

The Oracle logo is displayed in red, featuring the word "ORACLE" in a bold, sans-serif font with a registered trademark symbol (®) to the upper right of the "E".

**ORACLE®**

---

**ATG WEB COMMERCE**

Version 10.0.2

Page Developer's Guide

Oracle ATG  
One Main Street  
Cambridge, MA 02142  
USA

## ATG Page Developer's Guide

### Document Version

Doc10.0.2 PAGEDEVv1 4/15/2011

Copyright © 1997, 2011, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

#### U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark licensed through X/Open Company, Ltd.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/us/corporate/accessibility/index.html>.

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.



# Contents

1	Introduction	12
2	Creating JavaServer Pages	13
	Tag Libraries	14
	JSTL	14
	DSP	14
	Importing Tag Libraries	15
	Setting Expression Language Variables	15
	Accessing Standard JavaBeans	15
	Accessing Dynamic Beans	17
	dsp:getvalueof versus dsp:tomap	18
	Importing Nucleus Components	18
	EL Variable Scopes	19
	Setting Scripting Variables	19
	Displaying Dynamic Values	20
	Specifying a Default Value	20
	Using EL to Display Property Values	20
	Referencing Nested Property Values	21
	Displaying Values that Contain HTML	21
	Setting Component Properties	21
	Setting properties with dsp:setvalue	22
	Setting properties with dsp:a	22
	Using Expression Language to Set Properties	23
	Accessing Indexed Properties	23
	Specifying an Index with Another Property	24
	Specifying an Index with a Page Parameter	24
	Hashtable Property Values	25
	Tag Converters	25
	CreditCard	27
	currency	28
	Date	31
	map	32
	Nullable	33
	Number	34
	Required	34
	ValuesHTML	35
	Page Parameters	36



Passing Page Parameters	36
Page Parameter Scope and Inheritance	37
Displaying HTML of a Parameter Set to a URL	38
Reusing Application Components	38
Included Pages	39
Relative and Absolute URLs	40
Invoking Pages in Other Web Applications	41
Passing Parameters to Embedded Files	42
<b>3 Using ATG Servlet Beans</b>	<b>44</b>
Embedding ATG Servlet Beans in Pages	44
Using ForEach to Display Property Values	44
Servlet Bean Parameters	45
Parameter Scope	46
Accessing Servlet Bean Parameters with EL	47
Importing Servlet Beans	48
Passing Component Names with Parameters	48
Displaying Nested Component Properties with Servlet Beans	49
Nesting Servlet Beans	50
Other Commonly Used Servlet Beans	52
<b>4 Coding a Page for Multiple Sites</b>	<b>55</b>
Obtaining Product Sites	55
Site Selection Algorithm	56
Site Priority and Precedence	56
SiteIdForItem Example	56
Getting Site Information	58
Evaluating Site Sharing	59
Changing Site Context	60
Generating Links to Other Sites	61
<b>5 Serving Targeted Content with ATG Servlet Beans</b>	<b>65</b>
Targeting Servlet Beans	65
Sorting Results	66
Controlling Event Logging	67
Limiting Result Sets	68
Using Slots	68
Setting Pages To Support Scenarios	69
FormSubmission Event	69
Clicks a Link	69
<b>6 Integrating XML With ATG Servlet Beans</b>	<b>71</b>
XML, XSLT, and DOM	71
XML Transformation	72
Processing XML in a JSP	73
Transforming XML Documents with Templates	76



	Applying a Template to an XML Document	77
	XMLTemplateMap Component	78
	XML Document Cache	78
	XML Transformation and Targeting	79
	Passing Parameters to XSL	80
	XSL Output Attributes	82
	Encoding	83
	Method	83
	MIME Type	83
	Multiple XMLTransform Servlet Beans in a Page	84
	Nested XMLTransform Servlet Beans	84
	Serialized XMLTransform Servlet Beans	85
<b>7</b>	<b>Forms</b>	<b>87</b>
	Form Basics	87
	DSP Tags and HTML Conventions	88
	Embedding Pages in Forms	89
	Setting Property Values in Forms	90
	Scalar (Non-Array) Properties	91
	Array Properties	94
	Map Properties	96
	Set Property Values via Hidden Inputs	97
	Submitting Forms	97
	Submit Input Tags	97
	Order of Tag Processing	99
	Synchronizing Form Submissions	99
	Preventing Cross-Site Attacks	100
<b>8</b>	<b>Form Handlers</b>	<b>101</b>
	Form Handler Classes	101
	Resetting a Form	102
	Form Error Handling	102
	Displaying Form Exceptions	103
	Detecting Errors	103
	Alerting Users to Errors	103
	Redirecting on Form Submission	104
<b>9</b>	<b>Search Forms</b>	<b>105</b>
	Creating SearchFormHandler Components	106
	Basic SearchFormHandler Properties	106
	Multisite SearchFormHandler Properties	107
	Keyword Search	108
	Keyword Search Properties	108
	Logical Operators	109
	Single-Value Property Searches	110
	Multi-Valued Property Searches	110



Quick Searching	110
Text Search	110
Text Search Properties	111
Enable Full-Text Searches	112
Hierarchical Search	112
Advanced Search	113
Advanced Search Properties	113
How Advanced Searches Operate	115
Searching a Range of Values	116
Combination Search	116
Search Form Submit Operations	117
Executing the Search	117
Clearing Search Query Data	117
Presenting Search Options	118
Managing Search Results	118
Search Results Properties	119
Displaying Results Data	119
Linking to Search Results	120
Organizing Query Result Items	120
<b>10 RepositoryFormHandler</b>	<b>123</b>
RepositoryFormHandler Properties	124
Required RepositoryFormHandler Properties	124
Optional RepositoryFormHandler Properties	124
RepositoryFormHandler Submit Operations	126
Updating Item Properties from the Value Dictionary	126
RepositoryFormHandler Navigation Properties	127
Updating Multi-Valued Properties	128
Modifying Properties that Refer to Items by repositoryId	129
Managing Map Properties	130
<b>11 User Profile Forms</b>	<b>135</b>
ProfileFormHandler	135
ProfileFormHandler Submit Operations	135
Setting Profile Values	136
ProfileFormHandler Properties	137
Multi-Valued Properties	139
ProfileFormHandler Navigation Properties	141
Single Profile Form Handler Error Messages	142
ProfileFormHandler Scope	142
Multi-Profile Form Handlers	143
Multi-Profile Form Handler Submit Operations	143
Setting Values for Multiple Profiles	144
MultiProfileAddFormHandler Properties	145
Updating Values in Multiple Profiles	147
MultiProfileUpdateFormHandler Properties	147



Multi-Profile Form Handler Navigation Properties	148
Multi-Profile Form Handler Error Messages	149
Multi-Profile Form Handler Scope	150
Profile Form Examples	151
Create a User Profile	151
Create Multiple User Profiles	154
Update Multiple User Profiles	157
 12 Displaying Repository Data in HTML Tables	 161
TableInfo Basics	161
Localizing Column Headings	163
Managing Sort Criteria	164
Simple Table Example	166
User-Sorted Tables	169
Adding Graphic Sort Indicators	172
TableInfo Example	174
 Appendix A: DSP Tag Libraries	 179
dsp:a	181
dsp:beginTransaction	184
dsp:commitTransaction	185
dsp:contains	186
dsp:containsEJB	188
dsp:demarcateTransaction	189
dsp:droplet	191
dsp:equalEJB	192
dsp:form	193
dsp:frame	195
dsp:getvalueof	196
dsp:go	197
dsp:iframe	199
dsp:img	200
dsp:importbean	201
dsp:include	202
dsp:input	203
dsp:layerBundle	209
dsp:link	210
dsp:oparam	212
dsp:option	213
dsp:orderBy	214
dsp:page	215
dsp:param	216
dsp:postfield	218
dsp:property	220
dsp:rollbackTransaction	221
dsp:select	223



dsp:setTransactionRollbackOnly	225
dsp:setvalue	226
dsp:setxml	228
dsp:sort	229
dsp:tagAttribute	232
dsp:test	233
dsp:textarea	235
dsp:tomap	236
dsp:transactionStatus	238
dsp:valueof	241

## **Appendix B: ATG Servlet Beans** **243**

Servlet Beans by Category	243
Standard Servlet Beans	243
Database and Repository Access Servlet Beans	246
Multisite Servlet Beans	248
XML Servlet Beans	249
Transaction Servlet Beans	250
Personalization Servlet Beans	250
Business Process Tracking Servlet Beans	254
AddBusinessProcessStage	254
AddMarkerToProfile	256
BeanProperty	258
ArrayIncludesValue	261
Cache	262
CanonicalItemLink	265
CollectionFilter	267
Compare	269
ComponentExists	271
ContentDroplet	272
ContentFolderView	273
CurrencyConversionFormatter	276
CurrencyFormatter	277
EndTransactionDroplet	279
ErrorMessageForEach	280
For	284
ForEach	286
Format	289
GetDirectoryPrincipal	291
GetSiteDroplet	293
HasBusinessProcessStage	294
HasEffectivePrincipal	296
HasFunction	298
Invoke	299
IsEmpty	300
IsNull	302
ItemLink	303





ItemLookupDroplet	306
MostRecentBusinessProcessStage	308
NamedQueryForEach	309
NavHistoryCollector	312
NodeForEach	313
NodeMatch	315
PageEventTriggerDroplet	316
PipelineChainInvocation	317
PossibleValues	319
ProfileHasLastMarker	323
ProfileHasLastMarkerWithKey	325
ProfileHasMarker	328
ProtocolChange	331
Range	332
Redirect	336
RemoveAllMarkersFromProfile	337
RemoveBusinessProcessStage	338
RemoveMarkersFromProfile	340
RepositoryLookup	342
RQLQueryForEach	344
RQLQueryRange	348
RuleBasedRepositoryItemGroupFilter	352
SharingSitesDroplet	353
SiteContextDroplet	355
SiteIdForItem	356
SiteLinkDroplet	360
SitesShareShareableDroplet	362
SQLQueryForEach	364
SQLQueryRange	368
Switch	373
TableForEach	375
TableRange	378
TargetingArray	383
TargetingFirst	385
TargetingForEach	388
TargetingRandom	390
TargetingRange	393
TargetPrincipalsDroplet	397
TransactionDroplet	399
UserListDroplet	400
ViewPrincipalsDroplet	402
WorkflowInstanceQueryDroplet	404
WorkflowTaskQueryDroplet	407
XMLToDOM	410
XMLTransform	412



<b>Appendix C: SimpleSQLFormHandler</b>	<b>415</b>
SimpleSQLFormHandler Properties	415
Submit Handler Methods	417
SimpleSQLFormHandler Navigation Properties	417
SQL Form Handler Example	418
<b>Appendix D: Managing Nucleus Components</b>	<b>421</b>
Viewing Components in Nucleus	421
Module View	421
Path View	423
Module and Path View Synchronization	424
Searching for a Component	424
Creating Nucleus Components	424
Configuring Nucleus Components	425
Editing Property Values	426
Invoking Methods	427
Changing Registered Event Listeners	427
Changing a Component's Scope and Description	428
Viewing Configuration Layers and Properties Files	428
Starting and Stopping Nucleus Components	429
Rebuilding the Component Index	430
Managing Site Pages	430
Creating Document Folders	431
Deleting Folders	431
Searching for Documents	432
Creating Documents	432
<b>Appendix E: ATG Document Editor</b>	<b>435</b>
Document Editor Window	435
Document Outline	437
Text Editor	437
Dynamic Element Editor	437
Creating a JSP	439
Anatomy of a JSP	439
Displaying Property Values	441
Importing Components	441
Inserting a dsp:valueof tag	442
Previewing JSPs	442
Embedding Documents	443
Inserting ATG Servlet Beans	443
Inserting a Targeting Servlet Bean	444
Adding Slots	445
Inserting Forms	445
Creating Profile Form Pages	445
Using the Insert Form Wizard	446
Using a Form Template	447



Index	449
-------	-----



# 1 Introduction

The *ATG Page Developer's Guide* describes how to work with Nucleus components and JavaServer Pages—two core building blocks of any Web site.

Nucleus components are server-side JavaBeans and servlets that perform the back-end functionality of the Web application—for example, enabling database connectivity, logging, scheduling, and handling HTTP requests. Java developers create these components and link them together as applications in ATG's development framework, Nucleus.

Site visitors browse and interact with JavaServer Pages (JSPs), dynamic documents that conform to the JavaServer Pages 2.0 specification (<http://jcp.org/aboutJava/communityprocess/final/jsr152/>). The ATG platform's DSP tag library provides specialized markup tags to render content dynamically by linking Nucleus components directly to JSPs. The DSP tag library lets you connect JSP content to back-end Java code. Thus, the presentation layer and site application logic can be separately developed and maintained.

Nucleus and the DSP tag library let you accomplish the following tasks:

- Display component property values.
- Connect HTML forms to component property values, so information entered by the user is sent directly to the components.
- Reuse HTML content by embedding the contents of one page in other pages.
- Embed ATG servlet beans that display the servlet's output as a dynamic element in a JSP.



## 2 Creating JavaServer Pages

When a browser requests a JavaServer Page (JSP), the ATG platform locates the associated document, compiles it into Java code (unless it is pre-compiled), then converts the code into an HTML page for browser display. Because the HTML is generated at runtime, it can incorporate dynamic elements that customize the page for each user who requests it. JSP pages that run on the ATG platform can pass information to JavaBeans, servlets, and other Java conventions and receive the output of these operations.

### *Summary of JSP Functionality*

An application built from JSPs has the following built-in capabilities available to it:

- Display property values of Nucleus components. See [Displaying Dynamic Values](#) in this chapter.
- Display forms that connect to Nucleus component properties, so current property values can appear in input fields, and user-entered values can be saved to components. See the [Forms](#) chapter.
- Invoke ATG servlet beans at specified points in a page, returning the output of the servlet as a dynamic element of the page. Components that generate HTML are derived from the Java Servlet standard. See [Included Pages](#) in this chapter and [Embedding ATG Servlet Beans in Pages](#) in the [Using ATG Servlet Beans](#) chapter.
- Invoke servlet components with HTML passed in as parameters. In this way, servlet components can dynamically generate HTML, so specific HTML tags do not need to be embedded in the Java code. Thus, the HTML-generated display and Java-generated functionality can be maintained in separate files. See [Embedding ATG Servlet Beans in Pages](#) in the [Using ATG Servlet Beans](#) chapter.
- Set Nucleus components to a scope, which determines its availability to other application elements. See [Changing a Component's Scope and Description](#).
- Extract data stored in XML Data Object Models, transfer it to Nucleus components and render it in a JSP. See [Integrating XML with ATG Servlet Beans](#).

This chapter shows how to build JSPs on the ATG platform, focusing on special features that are provided by the DSP tag library. The chapter contains these sections:

- [Tag Libraries](#)
- [Setting Expression Language Variables](#)
- [Setting Scripting Variables](#)
- [Displaying Dynamic Values](#)



- [Setting Component Properties](#)
- [Accessing Indexed Properties](#)
- [Tag Converters](#)
- [Page Parameters](#)
- [Reusing Application Components](#)

## Tag Libraries

The ATG platform supports JSPs as its primary authoring format. The ATG platform ships with two tag libraries, located at <ATG10dir>/DAS/taglib:

- [JSTL](#)
- [DSP](#)

Additional tag libraries might be provided with other ATG products and solutions.

### JSTL

The JavaServer Pages Standard Tag Library (JSTL) from Sun Microsystems handles generic Web application tasks that are supported by all application servers. JSTL is comprised of four tag libraries:

- core: evaluate and display data
- xml: manage XML code
- fmt: handle internationalization
- sql: interact with SQL data sources

For more information, see <http://java.sun.com/products/jsp/jstl/>.

### DSP

The DSP tag library lets you access all data types in ATG's Nucleus framework. Other functions provided by these tags manage transactions and determine how to render data in a JSP. You should use tags from the DSP tag library only for tasks that involve ATG resources. For generic Web application tasks, use JSTL tags.

In pages that import the DSP tag library, you should generally favor DSP tags over equivalent JSP tags. In general, the corresponding DSP tag library tags provide enhanced functionality, such as support for the passing of object parameters between pages. In particular, use [dsp:include](#) and [dsp:param](#) rather than their JSP equivalents.

The DSP tag library supports both scripting and the JSP Expression Language. For example, the following two tags are equivalent:

```
<dsp:valueof param="<%= currentCourse %>"/>
<dsp:valueof param="${currentCourse}"/>
```



## Importing Tag Libraries

A JSP must import the ATG platform's DSP tag library so it access Nucleus components. The following page directive specifies the DSP tag library location and the dsp prefix to use for its tags:

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
```

Below the DSP tag libraries import statements, [dsp: page](#) tags enclose the remaining document. These tags facilitate ATG page processing and must be included in each page as they are in this example.

### Changing the tag library directive

There are two ways to deploy a tag library. By default, the DSP tag libraries `tl d` files are kept in the tag library JAR file inside `WEB-INF/lib`. The URI used in JSP page directives are defined in the `tl d` itself.

The URI in the page directive must match the URI that is specified in either the tag library's `tl d` file (as is the case with the DSP tag libraries) or `web.xml`. In order to define a different URI for the DSP tag libraries, specify the new URI and `tl d` file location in `web.xml`. Then, the JSPs can use either the default or the new URI.

For information about making your J2EE application ready for deployment, see the pertinent documentation of your application server.

## Setting Expression Language Variables

You can set any Nucleus component and Dynamic Bean to a variable that can then be referenced by the JSP Expression Language (EL). The DSP tag library provides three tags for setting components to EL variables:

- `dsp: getvalueof` enables access to standard JavaBean components, page parameters and constant values.
- `dsp: tomap` enables access to non-standard Dynamic Beans—for example, a `RepositoryItem`.
- `dsp: importbean`, which imports Nucleus components into a JSP, can also be used to set standard Java Bean components to EL variables.

### Accessing Standard JavaBeans

`dsp: getvalueof` lets you save a [JavaBean component](#), [page parameter](#), or [static value](#) to a variable in the specified scope. Other tags such as `dsp: valueof` can then reference that value. The variable is also accessible to the JSP Expression Language.

`dsp: getvalueof` has the following syntax:



---

```
<dsp: getval ueof
    source-value var=" var-name" [scope=" scope-spec" ] >
</getval ueof>
```

---

where *source-value* specifies the value source as a [JavaBean component](#), [page parameter](#), or [static value](#):

- bean=" *nucleus-path/component-name*"
- param=" *param-name*"
- value=" *static-value*"

### **JavaBean component**

dsp: getval ueof can make JavaBean components available to EL. The following dsp: getval ueof tag sets the student variable to the Nucleus component /atg/samples/Student\_01. The tag omits the scope attribute, so the default pageScope is applied:

```
<dsp: getval ueof bean="/atg/samples/Student_01" var="student"></dsp: getval ueof>
```

The student variable provides access on the current JSP to all Student\_01 component properties. For example, you can use EL to render this component's age property:

```
You are <c: out value="${student.age}"/> years old.
```

### **Page parameter**

dsp: getval ueof can make page parameters available to EL. In the next example, dsp: getval ueof sets the requestScope variable gender to reference the page parameter sex:

```
<dsp: getval ueof param="sex" var="gender" scope="request"> </dsp: getval ueof>
```

After you set this variable, it is accessible to other objects in the same request scope. In the following example, the c: when tag evaluates requestScope. gender in order to determine which content to display:

---

```
<c: choose>
  <c: when test="${gender == 'female'}"/>
    Formal dresses are on sale!
  </c: when>

  <c: when test="${gender == 'male'}">
    Tuxedos are on sale!
  </c: when>
</c: choose>
```

---

### **Static value**

dsp: getval ueof can save a static value to an EL variable, as in the following example:





```
<dsp: getvalueof value="student" var="userType"> </dsp: getvalueof>
```

Subsequent references to the `userType` variable resolve to the literal string `student`.

## Accessing Dynamic Beans

`dsp: tomap` can save any dynamic value—JavaBean, Dynamic Bean or page parameter—as an EL variable. `dsp: tomap` has the following syntax:

---

```
<dsp: tomap var=var-name
  bean="namespace-path/component-name.property[.property]" . . .
  [scope="scope-spec"]
/>
```

---

For example:

```
<dsp: tomap bean="/atg/userprofiling/Profile" var="userProfile"/>
```

This tag creates the variable `userProfile` which can be used to reference all `Profile` component properties. The tag omits the `scope` attribute, so the default `pageScope` is applied. You can render this value later on the page as follows:

```
Welcome back, <c: out value="${userProfile.firstName}"/>!
```

### Mapping to properties

In the previous example, the `firstName` property is referenced through the `userProfile` variable. `dsp: tomap` also lets you set an EL variable to a component property, enabling direct access to the property itself:

---

```
<dsp: tomap bean="/atg/userprofiling/Profile.firstName" var="name"/>
Welcome back, <c: out value="${name}"/>!
```

---

### Accessing Dynamic Beans as properties

If a `dsp: tomap` tag sets its `recursive` attribute to `true`, all underlying Dynamic Beans are wrapped as Maps and can be accessed through the EL variable. This can help simplify code that accesses multi-dimensional data. For example, if the `RepositoryItem Profile` has a property `fundList` which is itself a `RepositoryItem`, you can provide direct access to the `fundList` property as follows:

---

```
<dsp: tomap bean="atg/userprofiling/Profile" var="profile" recursive="true"/>
I own these mutual funds: <c: out value="${profile.fundList}"/>
```

---

If desired, you can access the underlying Dynamic Bean rather than its Map wrapper through the `DynamicBeanMap` property `_realObject`. Thus, given the previous example, you might access the Dynamic Bean `fundList` as follows:

```
${profile.fundList._realObject}
```

## dsp:getvalueof versus dsp:tomap

DSP tags `dsp:getvalueof` and `dsp:tomap` overlap in functionality. The following considerations can help guide your choice of one over the other:

- While both `dsp:getvalueof` and `dsp:tomap` provides access to JavaBean components, `dsp:getvalueof` is more efficient. `dsp:tomap` wraps all components including JavaBeans in a Map, where component properties are keys and the corresponding standard bean properties are values. Conversely, a variable created by `dsp:getvalueof` directly references the source JavaBean.
- `dsp:tomap` lets you set variables to component properties, while variables set by `dsp:getvalueof` can reference only components.

## Importing Nucleus Components

You can import a Nucleus component with `dsp:importbean`, which is modeled after Java class imports:

---

```
<dsp:importbean bean="Nucleus-path" [var="attr-name" [scope="scope-spec"] ]
```

---

After a component is imported, references to it can omit its path and use its name only.

For example, the following tag imports the component `/samples/Student_01` and creates the EL variable `student`:

```
<dsp:importbean bean="/samples/Student_01" var="student"/>
```

This tag accomplishes two tasks:

- Imports the `Student_01` component into the `pageContext` so non-EL references to `Student_01` can exclude its full Nucleus path and just use the shorthand name (`Student_01`). For example:

```
My favorite subject is<dsp:select bean="Student_01.subject">
<dsp:option value="math">Math</dsp:option>
  <dsp:option value="music">Music</dsp:option>
  <dsp:option value="physical Education">Phys Ed</dsp:option>
  <dsp:option value="history">History</dsp:option>
</dsp:select>
```

- Makes the `Student_01` component available to EL through the variable `student`. For example:

```
My name is <c:out value="{student.name}"/>
```

By default, the `student` variable is set to page scope, which makes it accessible to any EL-enabled resources on the same page. You can define a larger scope for the `student` variable by setting the `scope` attribute to `request`, `session`, or `application`.



**Note:** `dsp:importbean` can import any JavaBean or Dynamic Bean component into a page; however, this tag can only set standard JavaBeans to an EL variable and make them accessible to EL. To make Dynamic Beans visible to EL expressions, use `dsp:tomap`.

## EL Variable Scopes

An EL variable can be set to one of several scopes:

- `pageScope` constrains access to the current JSP.
- `requestScope` constrains access to the current request. The variable is not accessible to other requests, even if they originate from the same user session.
- `sessionScope` enables access to the user session. The variable persists for the duration of that session.
- `applicationScope` enables access to all application resources and users. The state of the variable is the same for all users of the application.

The default scope is `pageScope`, which is used when the variable declaration omits scope specification. You should specify the narrowest scope possible, in order to minimize the chance that two variables with the same name have overlapping scopes.

## Setting Scripting Variables

You can use `dsp:getvalueof` and `dsp:tomap` tags to extract data from a component or parameter and place it in a scripting variable, which can then be referenced by other library tags that support Java expressions.

In the following example, `dsp:getvalueof` sets the `bgcolor` variable to the value held by the `Profile.preferredColor` property. When the ATG platform processes the body color code, it retrieves the `bgcolor` value and uses it to set the current page background color:

---

```
<dsp:getvalueof id="bgcolor" bean="Profile.preferredColor">
  <body bgcolor="<%=bgcolor%>">
</dsp:getvalueof>
```

---

DSP tags can also reference scripting variables in Object runtime expressions. In the following example, `dsp:getvalueof` sets the scripting variable `profile` to the `Profile` component. The `dsp:param` tag sets the page parameter `userRole` from the component's `Role` property, which it obtains from the runtime expression `<%=profile.getRole()%>`:

---

```
<dsp:getvalueof id="profile" bean="atg/userprofile/Profile">
  <dsp:param userRole="value" value="<%=profile.getRole()%>" />
</dsp:getvalueof>
```

---



## Displaying Dynamic Values

You can use `dsp: val ueof` to display dynamic values—for example, the value of a Nucleus component property, or a page or servlet bean parameter. `dsp: val ueof` tag uses this syntax:

---

```
<dsp: val ueof val ue-source >
  [default t-val ue]
</dsp: val ueof>
```

---

where *val ue-source* specifies a dynamic value as follows:

- `bean="nucleus-path/component-name. property[. property]. . ."`
- `val ue="$EL-variable[. property]. . ."`
- `param="param-name"`

### Specifying a Default Value

`dsp: val ueof` can specify a default value to display by embedding it between the open and close tags. This value displays only if the component property or EL variable is empty, or undefined/null.

If no default value is required, you can use this simplified syntax:

---

```
<dsp: val ueof dynamic-val ue-spec />
```

---

For example, you can access properties of the `Student_01` component as follows:

```
Name: <dsp: val ueof bean="Student_01. name"/><p>
Age: <dsp: val ueof bean="Student_01. age"/><p>
```

### Using EL to Display Property Values

If the `Student_01` component is referenced by an EL variable, you can use EL to display its property values, as in the following example. When the page is served, `dsp: val ueof` tags are replaced by the values of the specified component properties. If a value is null, the default value `No Name` displays instead.

---

```
<dsp: getval ueof val ue="/samples/Student_01" var="student"> </dsp: getval ueof>
. . .
Name: <dsp: val ueof val ue="${student. name}">No Name</dsp: val ueof>
Name: <dsp: val ueof val ue="${student. age}">No Name</dsp: val ueof>
```

---

If no default value is required, you can use EL directly to specify display of the component property, as follows:



---

```
<dsp: getval ueof val ue="/samples/Student_01" var="student"> </dsp: getval ueof>
...
Name: <c: out val ue="${student. name}"/>
Name: <c: out val ue="${student. age}"/>
```

---

## Referencing Nested Property Values

JavaBean property values can themselves be JavaBeans with their own property values. You can reference nested property values in JSPs with the following syntax:

*component-name. property. nested-property[. nested-property]. . .*

- *property* specifies a property of *component-name* that refers to another component
- *nested-property* specifies a property of the parent property

When you reference a property in a JSP, you can also reference a property of that property, with multiple levels of indirection.

For example, the JavaBean `PersonManager` has the property `currentPerson`. The value of this property is an instance of a `Person` class object which has an `age` property. A JSP can reference the age of the current person as follows:

```
<dsp: getval ueof val ue="/samples/PersonManager" var=pMgr></dsp: getval ueof>
<c: out val ue="${pMgr. currentPerson. age}"
```

## Displaying Values that Contain HTML

The `dsp: val ueof` tag's `val uei shtml` attribute lets you display the value of a property or page parameter that contains HTML. For example, the `Student_01` component might have a `homepage` property whose value is a link to a Web page:

```
<dsp: val ueof bean="Student_01. homepage" val uei shtml ="true"/>
```

This tag displays the value of the `homepage` property by interpreting its tags as HTML tags, rather than display the tags themselves.

## Setting Component Properties

Two DSP tags let you set component properties on a JSP page:

- `dsp: setval ue` sets a bean property or a page parameter on the current page.
- `dsp: a` specifies a hyperlink that sets a property on activation.

## Setting properties with dsp:setvalue

`dsp:setvalue` sets a bean property or page parameters:

---

```
<dsp:setvalue target-spec [source-spec] />
```

---

where *target-spec* specifies the target to set in one of the following ways:

- *bean=property-spec*
- *param=param-name*

and *source-spec* specifies the source value in one of the following ways:

- *beanvalue=property-spec*
- *paramvalue=param-name*
- *value=value*

The value of the property is assigned by calling the appropriate accessor method on the target object. If no source value is specified, the target is set to null value.

For example, the following JSP fragment sets the bean property `Student_01.name` to the string value `Brian`:

---

```
<dsp:importbean bean="/samples/Student_01"/>
<p><dsp:setvalue bean="Student_01.name" value="Brian"/>
```

---

## Setting properties with dsp:a

The anchor tag `dsp:a` lets you set a property on activation of the hyperlink:

---

```
<dsp:a href="url" bean="property-spec" source-spec> link-text </dsp:a>
```

---

where *source-spec* specifies the source value in one of the following ways:

- *beanvalue=property-spec*
- *paramvalue=param-name*
- *value=value*

For example:

---

```
<dsp:a href="page2.jsp" bean="/samples/Student_01.name" value="Ron">
  If your name is Ron, click here!
</dsp:a>
```

---



When this link is clicked, the property Student\_01. name is set to Ron before the browser opens the link.

### Setting multiple properties

In order to set multiple properties within a dsp:a tag, use the following syntax:

---

```
<dsp: a href="url "
  <dsp: property bean=property-spec" source-spec />
  <dsp: property bean=property-spec" source-spec />
  [...]
  link-text </dsp: a>
```

---

**Note:** A [dsp: a](#) tag can set a property only if the destination URL specifies a JSP in the same Web application.

### Using Expression Language to Set Properties

The tags dsp: setvalue and dsp: a can use EL to set properties. For example:

---

```
<dsp: importbean bean="/samples/Student_01" var="student"/>
<dsp: importbean bean="/samples/Teacher1"/>

<dsp: setvalue bean="Teacher1. oldestStudent" value="${student. name}"/>
```

---

**Note:** EL can only be used to express the source value; the target expression must directly specify the component property to set.

## Accessing Indexed Properties

JavaBeans supports indexed property values—that is, properties that store an ordered set of values in an array. In order to retrieve and set indexed properties, a JavaBean implements accessor methods that take an additional integer index parameter, as follows:

```
Object getX(int index);
void setX(int index, Object value);
```

where X is the property name. You can get and set the values of indexed properties by referencing the indexed property with this syntax:

*property-spec*[*array-index*]

where *array-index* is one of the following:

- integer expression

- bean: *property-spec*
- param: *page-param*

*array-index* can also be expressed in EL, where the expression resolves to an integer value, JavaBean property or page parameter.

In all cases, the array index must resolve to an integer between 0 and the array size. This integer is passed as an index argument to the accessor method for that property, which gets or sets the specified property element.

For example, the JavaBean `Student_01` might have a `Hobbies` property that manages a list of hobbies. The JavaBean defines the following accessor methods:

```
String getHobbies(int which);
void setHobbies(int which, String hobby);
```

Given these methods, you can use `dsp: valueof` to display a given hobby from this property:

```
<dsp: valueof bean="Student_01.hobbies[0]">no hobby</dsp: valueof>
```

## Specifying an Index with Another Property

The value of the array index can be set from the value of another property. For example, if the `Student_01` object also has a `favoriteHobby` property that stores an integer value, it can be used to specify which hobby to display:

---

```
<dsp: getvalueof bean="/samples/Student_01" var="student"/>

<dsp: valueof value="{student.hobbies[student.favoriteHobby]}">
I have no favorite hobby.
</dsp: valueof>
```

---

When the page is rendered, it uses the array index specified by `favoriteHobby` and returns the property element's current value. All subsequent changes made to the array index property after the page is rendered to the browser have no effect on form submission.

**Note:** You can use array notation in some cases for a property that does not have the indexed `setX` and `getX`—for example, a property whose value is an array or a vector. However, if an ATG Servlet Bean needs to change the value of an element of your property because you referenced this property in a `setvalue` tag, it must have an indexed `setX` method.

## Specifying an Index with a Page Parameter

The value of an array property index can also be set from a page parameter. For example, if a JSP has the page parameter `IatestHobby` and it contains an integer value, the JSP can use that parameter as an index into the array property `Student_01.hobbies`:

```
<dsp: valueof bean="Student_01.hobbies[param:IatestHobby]">
```





As with properties, you can also use EL to express the page parameter as an array index. The previous example might be modified as follows:

```
<dsp: getval ueof param="IatestHobby" var="hobby"></dsp: getval ueof>

<dsp: param name="IatestHobby" value="3"/>
<dsp: val ueof bean="Student_01.hobbies[${hobby}]" />
```

**Note:** While you can use a parameter as an array index, a parameter cannot itself be indexed.

## Hashtable Property Values

You can reference property values in a Hashtable or Dictionary. If you have a property whose type implements the Dictionary interface, you can access its values as properties of the property itself. For example, if you have a bean called MyBean with a Dictionary (or Hashtable) property called myHashtabl e, you can access the value that corresponds to the key named myKey of that Hashtable as follows:

```
<dsp: val ueof bean="MyBean.myHashtabl e.myKey">not set</dsp: val ueof>
```

## Tag Converters

ATG provides tag converter classes that let you explicitly control how form data is converted on input and displayed on output, and determine when exceptions are thrown. Certain DSP tags such as `dsp: i nput` and `dsp: val ueof` can specify these tag converters.

For example, the ATG installation provides a Credi tCard tag converter, which lets you control how to mask a portion of the String that represents a credit card number. The following `dsp: val ueof` tag specifies the Credi tCard tag converter and two of its attributes, `maskcharacter` and `numcharsunmasked`, which specify to use # as the mask character and to expose the credit card number's last six digits, respectively:

```
<dsp: val ueof param="paymentGroup.ccNum"
  converter="Credi tCard" maskcharacter="#" numcharsunmasked="6"
  No number availabl e
</val ueof>
```

The following DSP tags can specify tag converters:

```
dsp: a
dsp: i nput
dsp: param
dsp: postfi el d
dsp: sel ect
dsp: setval ue
dsp: textarea
dsp: val ueof
```



### Tag Converter Syntax

You can invoke a tag converter explicitly with the following syntax:

```
<dsp-tag converter="converter-name" [attribute-name="value"]... />
```

- *dsp-tag* is the DSP tag that specifies one of the [standard tag converters](#) or [custom tag converters](#).
- *converter* specifies the name of the converter.
- *attribute-name* optionally specifies an attribute defined in the tag converter. You can specify multiple attributes for a given tag converter.

For example, the following `dsp:input` tag specifies the tag converter `Date` and supplies its date attribute. The date attribute is set to a format supported by `java.text.SimpleDateFormat`:

```
<dsp:input bean="Semi nar. startDate" converter="Date" date="M/dd/yyyy" />
```

### Implicit Converters

A DSP tag can omit the `converter` attribute and implicitly invoke a tag converter by specifying an attribute that the tag converter defines as automatic. Each tag converter can define one automatic attribute. In the following example, the `dsp:input` tag implicitly invokes the date converter through its automatic attribute `date`:

```
<dsp:input bean="Semi nar. startDate" date="M/dd/yyyy" />
```

### Standard Tag Converters

The following table lists tag converters that are provided with the ATG platform. These are discussed in detail later in this chapter.

Converter	Input data type	Output data type	Function
<a href="#">CreditCard</a>	numeric string	integer, float, etc.	Determines how a credit card number is displayed.
<a href="#">currency</a> currencyConversion euro	numeric string	string	Displays currency in a format appropriate for the locale
<a href="#">Date</a>	String	Date	Parses strings into Dates; displays Dates in a variety of formats.
<a href="#">map</a>	Map	Map	Ensures that key-value pairs are parsed correctly for the specified map property.

Converter	Input data type	Output data type	Function
<a href="#">Nullable</a>	any	any	If field is left empty, corresponding property is set to null
<a href="#">Number</a>	numeric string	integer, float, etc.	Displays numeric data-types in a variety of formats
<a href="#">Required</a>	any	any	Throws a form exception if the user specifies a null or empty value
<a href="#">Value HTML</a>	string	string	Displays a string as formatted HTML text rather than escaped text.

### Custom Tag Converters

If your installation includes custom tag converters, a DSP tag can invoke these as follows:

```
<dsp-tag converter="converter-name" converterattributes="attr-list" />
```

*attr-list* is a list of semi-colon-delimited attributes that are defined by the tag converter, and their values. The *converterattributes* attribute allows use of attributes that are unknown to the DSPJSP tag library ([http://www.atg.com/taglibs/daf/dspjspTaglib1\\_0](http://www.atg.com/taglibs/daf/dspjspTaglib1_0)).

For example, given a custom tag converter *Warthog* that defines two attributes, *recommendations* and *widirection*, a *dsp:input* tag can specify this tag converter as follows:

```
<dsp:input type="text"
  bean="NuclearOrderFormHandler.address1"
  converter="warthog"
  converterattributes="recommendations=splunge;widirection=west" />
```

For information about defining tag converters, see the [ATG Programming Guide](#).

### CreditCard

Formats a credit card number.

#### Attributes



Attributes	Description
Credi tCard	Automatic Set to true or false to indicate whether to format this value as a credit card.
maskcharacter	The masking character. The default character is X.
numcharsunmasked	The number of characters to unmask, from right to left.
groupi ng si ze	The number of characters to group within the credit card string.

### Usage Notes

You can display a credit card number as a series of number groupings. You determine the number of items in each grouping by setting the groupi ng si ze attribute. For example:

```
<dsp: val ueof bean="Credi tCard. number" converter="Credi tCard" groupi ng si ze="4" />
```

This example might render a credit card number with four numbers in each grouping as follows:

7845 5624 5586 1313

In general, you should display the entire credit card number only when it is initially entered, in order to facilitate confirmation. Afterward, you typically display only a portion of the number, so users can recognize it while ensuring confidentiality. You can control number masking with two attributes:

- numcharsunmasked: The number of digits to unmask from right to left.
- maskcharacter:

For example:

Tag	Result
<pre>&lt;dsp: val ueof bean="Credi tCard. num"   converter="Credi tCard" numcharsunmasked="4" /&gt;</pre>	XXXXXXXXXXXX3456
<pre>&lt;dsp: val ueof bean="Credi tCard. num" converter="Credi tCard"   maskcharacter="*" numcharsunmasked="12" /&gt;</pre>	***567890123456

### currency

Renders a value in the currency pattern of the recognized locale.



Attributes	Description
currency	Automatic/Required Set to true or false.

### Usage Notes

The currency converter renders a value in the currency pattern of the recognized locale. In the following example, no locale is explicitly specified:

```
<dsp: valueof bean="PriceInfo.amount" converter="currency"/>
```

If no locale is specified, the locale is determined as follows, in descending order of precedence:

1. From a locale request parameter, either a `java.util.Locale` object or a String that names a locale.
2. From the `RequestLocale`.

A specified locale always has precedence, as in the following example:

```
<dsp: valueof bean="PriceInfo.amount" locale="en_US" converter="currency"/>
```

Each locale provides a default setting that determines:

- A currency symbol that is supported by ISO Latin-1 (ISO 8859-1)
- Separator, such as a comma or period
- Format, such as extending the decimal to the tenths position

### Tag Converter Extensions: euro and currencyConversion

ATG provides two converters, `currencyConversion` and `euro`, which let a site display prices in multiple currencies. The classes of these converters extend the currency tag converter class `atg.droplet.CurrencyConverter`:

- `currencyConversion` (`atg.droplet.CurrencyConversionTagConverter`) performs a currency conversion. By default, this converter converts the currency of a specified locale to the currency that is used in the user's locale.
- `euro` (`atg.droplet.EuroTagConverter`) formats a value as euros if that is the locale's currency.

These tag converters can take the following attributes:



Attributes	Description
symbol	Valid for euro and currencyConversion converter.  This attribute substitutes the specified HTML entity—typically, &euro; — for the euro currency symbol. This attribute is generally useful for applications that use Latin-1 (ISO 8859-1) encoding, which does not support the euro currency symbol.
reverse	Valid for currencyConversion, this attribute reverses the conversion, so it converts from the currency of the user's locale to the currency of the specified locale.

The following example uses the euro converter to display a price in Euros. The locale attribute specifies how to format the output.

```
<p>Price in Euros: <dsp: valueof locale="en_UK"
    bean="PriceInfo.amountInEuros" converter="euro" symbol="&euro;" />
```

In the next example, the dsp: valueof tag uses the currencyConversion converter to display the price in British pounds. The exchange rate is obtained from the resource file atg/droplet/ExchangeRates.properties:

```
<p>Price in British Pounds: <dsp: valueof locale="en_UK"
    bean="PriceInfo.amountInEuros" converter="currencyConversion" />
```

If locale indicates a non-euro country such as the United States, the currency converter returns null and the dsp: valueof tag returns its default value, if any is provided.

The euro and currencyConversion converters use the same algorithm as currency to discover a locale. In the following example, no locale is specified. If the locale is a country that uses the euro, the value of priceInfo.amountInEuros is converted to the appropriate currency (using the official exchange rates) and displayed in the appropriate format for that locale. If the locale is not a country that uses the euro, the price is displayed as \$5.00.

```
<dsp: valueof bean="priceInfo.amountInEuros" converter="currencyConversion">
    Current locale does not use Euros
</dsp: valueof>
```

The currencyConversion provides a reverse attribute that converts from a locale-specific European currency (for example, British pounds) into Euros:

```
<dsp: valueof locale="en_UK_EURO" bean="priceInfo.amountInEuros"
    converter="currencyConversion" reverse="true" symbol="&euro;" />
```



```
no price
</dsp: valueof>
```

In this example:

- The amount stored in priceInfo.amountInEuros is converted to euros.
- The symbol attribute supplies the HTML entity for displaying the euro currency character.
- The locale attribute is set to en\_UK\_EURO value, so the euro value is formatted for the United Kingdom accordingly.

## Date

Parses and formats dates.

Attributes	Description
date	Automatic/Required  Specifies a desired date format supported by the Java class <code>java.text.SimpleDateFormat</code> .
maxdate	Optional, specifies the maximum date allowed for input. The format of this setting must conform to the format specified by the date attribute.
mindate	Optional, specifies the minimum date allowed for input. The format of this setting must conform to the format specified by the date attribute.

### Usage Notes

You can specify how dates are formatted in a form with the date tag converter. The date converter uses the formatting and parsing methods and formats of the `java.text.SimpleDateFormat` class. For example:

This date converter:	Renders output like this:
date="MMM d, yyyy"	November 12, 1998
date="M/dd/yy"	11/12/98

See the JDK documentation on `java.text.SimpleDateFormat` for details on how the converter parses strings into Dates and formats Dates into strings for display.

Because the date converter does not flag an error if a two-digit value is entered when a four-digit value is expected for the year, your form validation logic might require error checking to handle this case. Also, if



the field expects a two-digit year entry, the year is interpreted as being no more than 80 years before or 20 years after the current date.

The following example uses the date converter to format a date:

```
<dsp:input bean="NewUser.lastActivityDate" date="MMMM d, yyyy"/>
```

The following example defines a parameter as a Date:

```
<dsp:param name="lastActivityDate" value="3/4/98" date="M/dd/yy"/>
```

If a form contains a date input field, you might want to place a note near that field that indicates the expected format. If a date is entered in the wrong format, it is interpreted according to the format specified in the date converter, and as a result might store an incorrect date.

### ***mindate/maxdate***

You can use the date converter's optional `mindate` and `maxdate` attributes individually or together to restrict the range of dates a field accepts. For example, if you have a field for the user's birth date, you typically want to require that the user enter a four-digit year to make sure the date the customer enters is unambiguous, as in this tag:

```
<dsp:input bean="NewUser.birthDate" date="M/dd/yyyy"/>
```

If users enter a two-digit year instead, the date might be interpreted differently than expected. For example, 2/16/59 is interpreted as February 16, 0059. By using the `mindate` attribute, however, you can reject dates that are earlier than a certain date you specify. For example, the following tag includes a `mindate` (in the format specified by the date attribute) of January 1, 1900:

```
<dsp:input bean="Employee1.birthDate" date="M/dd/yyyy" mindate="01/01/1900"/>
```

In this case, if the user enters 2/16/59, the entry is rejected, because February 16, 0059 is earlier than the date specified by `mindate`.

When a user enters a date that is outside of the bounds specified by `mindate` and `maxdate`, the converter rejects the value and throws a `TagConversionException` with an `invalidDate` error code. You can include error handling in your form to display a message instructing the user to enter a valid date.

### **map**

Enables entry of Map data on a form.

Attributes	Description
<code>value</code>	Required, specifies one or more key/value pairs separated by the separator character—by default, a comma.





keyval ueseparator	Optional, the character that separates a Map key and its value. Default: equals sign (=)
separator	Optional, the character that separates multiple key/value settings. Default: comma (, )

### Usage Notes

ATG's built-in map converter facilitates input of Map property values on forms. You can use it with `dsp:input` as follows:

```
<dsp:input bean="FormHandler.map-property" converter="map"
  value="key=value" [, ...] />
```

**Note:** The map converter is not required by the [RepositoryFormHandler](#), which can handle Map data in its own way (see [Managing Map Properties](#) in the RepositoryFormHandler chapter).

The map converter lets you input multiple key/value pairs. For example:

Please verify the following address: <br/><br/>

```
Street: &nbsp;  <c: out val ue="\${emp. street}"/> <br/>
Ci ty: : &nbsp;  <c: out val ue="\${emp. ci ty}"/> <br/>
State: : &nbsp;  <c: out val ue="\${emp. state}"/> <br/>
Zi p code: : &nbsp;  <c: out val ue="\${emp. zi p}"/> <br/>
```

```
<dsp:input type="hidden"
  bean="FormHandler.address" converter="map"
  value="street=${emp.street},city=${emp.city},state=${emp.state},zip=${emp.zip}"
/>
```

For more information on using the map converter, see [Map Properties](#) in the [Forms](#) chapter.

## Nullable

Attributes	Description
nul l abl e	Automatic/Required Set to true or fal se.



### Usage Notes

By default, an empty form field leaves the corresponding property unchanged on form submission. The `nullable` converter lets you change this behavior and set the property to null. For example:

```
<dsp:input type="text" bean="Person.address" nullable="true"/>
```

You can use the `nullable` converter only with data types that are classes such as `Integer`; `nullable` converters cannot be used with primitive data types such as `int`. If you use the `nullable` converter with a property whose data type is a Java primitive, and the user leaves the field empty, an exception is thrown.

`nullable` can also be specified as an attribute of another tag converter such as [Date](#) or [Number](#). For example:

```
<dsp:input bean="NewUser.age" number="#" nullable="true"/>
```

## Number

Attributes	Description
number	Automatic/Required  Specifies a desired numeric format supported by the Java class <code>java.text.DecimalFormat</code> .

### Usage Notes

The `Number` converter uses the formatting and parsing methods and formats of the `java.text.DecimalFormat` class. For example:

This converter:	Formats the numeric value as follows:
number="#"	Truncates decimals in the rendered value.
number="###,###.##"	Uses comma as a grouping separator, and displays up to two decimal places.
number="#.00"	Always displays two decimal places, even there is no decimal value.
number="0.00"	Displays a number with two decimal places and at least one digit before the decimal point, even if any of the digits is zero.

## Required



Attributes	Description
required	Automatic/Required Set to true or false.

### Usage Notes

The Required converter requires a user to populate a form field before the form can be processed. This converter only affects the corresponding property's setX operation. You can use it in the following form tags:

```
dsp:input  
dsp:select  
dsp:postfield  
dsp:textarea
```

If the value in a required field is an empty string or contains white space, a `DropLetFormException` for this element is thrown.

For example, the following input tag creates a form exception if the user submits the form with a null or empty value in this field:

```
<dsp:input type="text" bean="Person.userName" required="true"/>
```

**Note:** Other tag converters such as date and number can specify required as an optional attribute. For example, this tag specifies a date format, and makes the field required:

```
<dsp:input bean="NewUser.lastActivityDate" date="MMM d, yyyy" required="true"/>
```

### ValueIsHTML

Attributes	Description
valueIsHTML	Automatic/Required Set to true or false.

### Usage Notes

When a JSP is converted into HTML by the page compiler, any value that is provided by a DSP tag is automatically enclosed in escape tags so it is treated as non-HTML code, and any HTML tags are rendered as raw text. The ValueIsHTML tag converter lets a DSP tag display a value as formatted HTML code. For example:

```
<dsp:param name="boldCode" value="<b>bold</b>"/>  
Bold text looks <dsp:valueof param="boldCode" valueIsHTML="true"/>
```



A browser displays the `dsp: val ueof` tag as follows:

Bol d text looks **bol d**

## Page Parameters

A page parameter is a named value that one page can pass on to another. A page parameter can be set to a component property, another parameter, or a constant value. Page parameters are typically appended to a URL as query strings or are submitted with a form. Tags that invoke other pages, such as `dsp: a` and `dsp: i ncl ude`, can set these query strings. A JSP can explicitly define its own page parameters through the tag `dsp: param`. An included page also inherits the page parameters of its parent page.

### Passing Page Parameters

The following DSP tags let you pass parameters to the page that they invoke:

- `dsp: a`
- `dsp: frame`
- `dsp: i frame`
- `dsp: i mg`
- `dsp: i ncl ude`
- `dsp: l i nk`

You can pass parameters to the target page by embedding one or more `dsp: param` tags between the open and close tags. For example, you might set the `dsp: a` tag to create an anchor tag that passes a user's city as a query parameter:

---

```
<dsp: a href="homepage. j sp">Go to your home page
  <dsp: param name="ci ty" beanval ue="Profi l e. homeAddress. ci ty"/>
</dsp: a>
```

---

When a user clicks on this hyperlink, the tag resolves to a hyperlink string that embeds the parameter `ci ty` as a query string. For example:

```
<a href="homepage. j sp?ci ty=Boston"></a>
```

When the target page `homepage. j sp` is invoked by this hyperlink, it can access `ci ty` as a page parameter. In the following example, the page tests this parameter with the JSTL tag `c: choose`:

---

```
<c: choose>
  <c: when test="${param. ci ty == Boston}">
    A new store i s coming soon near you!
```



```

</c: when>

<c: when test="{param. ci ty == Atlanta}">
    There' s a sale at your local store!
</c: when>

<c: when test="{param. ci ty == Seattle}">
    The local store is going out of business. Visit the clearance sale.
</c: when>

<c: otherwi se>
    Check our new products! Now you can purchase more products from the
    Web si te.
</c: otherwi se>
</c: choose>

```

The query string `ci ty` is held as an attribute of the `param` implicit object and referenced as `param. ci ty`. Other operations that involve obtaining content from the request or response objects follow a similar format.

Click here to learn about our other stores:

```

<dsp: a href="{pageContext. request. requestURI }"?ci ty= atl anta>Atl anta</dsp: a>
<dsp: a href="{pageContext. request. requestURI }"?ci ty= boston>Boston</dsp: a>

```

In this example, when the user clicks the `boston` link, the current page reloads only this time it contains the query parameter `boston` so the page reflects information about the Boston location.

**Note:** Tags that invoke other pages like `dsp: a` and `dsp: i ncl ude` can pass component property values to the target page; however, they cannot pass the component itself. In the previous example, the anchor tag passes the value of the JavaBean property `Game. score` to `resul ts. j sp`; in order to obtain the component `Game`, the page must explicitly import it with `dsp: i mport`.

## Page Parameter Scope and Inheritance

Page parameters are accessible only to the current page and its embedded child pages. A page that is embedded in another page automatically inherits the parameters that are visible to its parent page. Conversely, parameters that are defined on an included page are not automatically visible to its parent pages.

### Overriding parameter values

If you use a DSP tag such as `dsp: i ncl ude` to pass page parameters to a page, these override values inherited from parameters of the same name.

A page can also override an inherited parameter's value either by resetting it through `dsp: setval ue`, or by declaring a parameter of the same name with `dsp: param`.

To change the value of a parameter within a page, use `dsp: setval ue`. For example:



```
<dsp:setvalue param="price" beanvalue="member.discount"/>
```

### ***Precedence of parameter values***

A parameter's value is always determined by its most proximate setting. In the following example, the value rendered by `dsp:valueof` tag varies according to the latest setting of the `price` parameter:

---

```
<dsp:importbean bean="/catalog/tools"/>
<dsp:setvalue param="price" beanvalue="hammers.KLDeluxe.price"/>

<p>KL Hammers now on sale! </p>
<p>KL Deluxe: <strike>
    $<dsp:valueof param="price"/>
</strike>&nbsp; &nbsp;

<dsp:setvalue param="price" beanvalue="hammers.KLDeluxe.discount_price"/>
This week: only $<dsp:valueof param="price"/>!
```

---

This might yield the following display:

Hammers now on sale!  
KL Deluxe: ~~\$19.95~~ This week: only \$14.99!

### **Displaying HTML of a Parameter Set to a URL**

If a parameter is set to a URL, you can specify to display the HTML of the specified link by setting the `valueishtml` attribute with the `dsp:valueof` tag. For example:

```
<dsp:valueof param="person.homepage"
valueishtml="true"></dsp:valueof>
```

This tag displays the value of `person.homepage` and interprets any HTML tags in the value, rather than showing the tags themselves.

## **Reusing Application Components**

The ATG platform helps you build an application from reusable components. A JSP can embed components such as other JSPs or images, or access them via hyperlinks. For example, it is generally good practice to create a single JSP fragment that contains page header data, and embed in multiple JSPs across the application. Reusing this component provides two benefits:

- To change the header content, you only need to edit a single file.
- The embedded file can respond dynamically to each file context via parameters that it obtains from the calling JSP.



This section covers the following topics:

- [Included Pages](#)
- [Relative and Absolute URLs](#)
- [Invoking Pages in Other Web Applications](#)
- [Passing Parameters to Embedded Files](#)

## Included Pages

A JSP can include one or more child pages. The included page can be one of two types:

- A JSP fragment with a `.jspf` extension included with the `JSP include` directive, and compiled together with the parent page.
- Another JSP included with either `dsp: include` or `jsp: include`, and compiled independently of the parent page before it is included.

The DSP library also provides tags for embedding other HTML elements, such as frames ([dsp: frame](#)) and images ([dsp: img](#)).

### JSP Fragments

A simple header that contains static elements can be stored as a JSP fragment (JSPF). The `JSP include` directive inserts the fragment into the parent page so it is processed as part of the parent page. For example:

```
<%@include file="header.jspf"%>
```

Dynamic elements in the child fragment can access values provided by the parent page, excluding those elements that are constrained to page or request scope. All URLs in `header.jspf`, including the calling URL (`file="header.jspf"`) resolve relative to the parent page. If the child page should resolve its URLs relative to itself, consider including it by calling `dsp: include`.

### JSPs

A JSP can embed another JSP by calling [dsp: include](#) or `jsp: include`. In both cases, the page compiler processes the child page before inserting it into the parent page. The included page must be a self-sufficient JSP with all required tags, including `dsp: page`. For example:

```
<dsp: include page="RegistrationForm.jsp" />
```

`dsp: include` offers these benefits over `jsp: include`:

- It can pass page parameters from the parent page to the child page, including dynamic objects.
- Its `page` attribute lets you set absolute URLs. See [Absolute URLs](#) for more information.
- URLs resolve relative to the current page, which removes the burden of managing URL paths in descendant pages.

**Note:** Always use `dsp: include` to embed a page that uses runtime expressions.

## Relative and Absolute URLs

DSP tags that invoke other application components can reference those components with [relative URLs](#) or [absolute URLs](#). The following DSP tags support both URL types:

- [dsp: a](#)
- [dsp: frame](#)
- [dsp: i frame](#)
- [dsp: i mg](#)
- [dsp: i ncl ude](#)
- [dsp: l i nk](#)

### **Relative URLs**

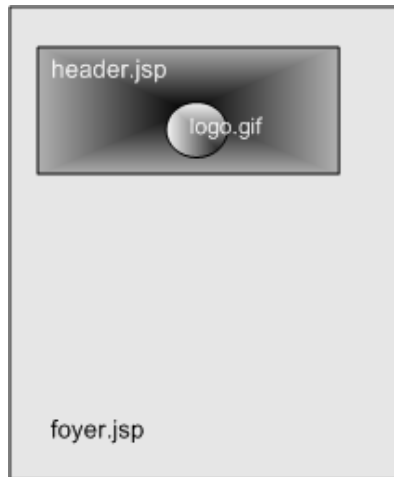
The `src` and `page` attributes of DSP tags such as `dsp: i ncl ude` and `dsp: a` lets you specify relative URLs. The ATG platform rewrites all relative links to embedded JSP files before including the contents of the file in the parent page. Relative links are resolved as relative to the page where they are defined. This allows entire directories of the document tree to be designed independently, so they can be reused by pages in separate directories or moved to another portion of the tree.

For example, the following file tree describes the relationship between the parent JSP `foyer. j sp` and two embedded files: `header. j sp` and `headeri mage. gi f`:

```
/testPages
  foyer. j sp
  /parts
    header. j sp
    l ogo. gi f
```

`foyer. j sp` embeds `header. j sp`, which in turn embeds the graphic file `l ogo. gi f`:





You can implement this presentation with two DSP tags that reference the embedded files with relative paths, as follows:

- In `foyer.jsp`:  
`<dsp:include page="parts/header.jsp"/>`
- In `header.jsp`:  
`<dsp:image src="./headerimage.gif"/>`

The header file `header.jsp` includes the image in `logo.gif`, which is in the same directory. Because the header file is likely to be included by other files in various locations, the reference in `header.jsp` to the image always to be relative to `header.jsp` itself.

In this example, when the ATG platform compiles `foyer.jsp` and resolves its links, it rewrites the `img` tag so it is embedded within `foyer.jsp` as follows:

```
<dsp:image src="/testPages/parts/headerimage.gif"/>
```

### **Absolute URLs**

Tags that invoke other files such as `dsp:include` can set their page attribute to an absolute path. Absolute paths typically begin in the Web application's root directory (`war` file); however, they can begin in any location that is specified in the `war` file's `web.xml`. During page compilation, the ATG platform retrieves the request object's context root and adds it to the supplied page attribute value. Thus, the tag is rewritten to include the complete URL so the page holding the URL is not limited by complex hierarchies.

## **Invoking Pages in Other Web Applications**

[dsp:include](#) lets you designate a target page in any J2EE Web application. You can do this in three ways:

- [Use the WebAppRegistry](#) to look up the application's context root.
- [Use the src attribute.](#)

- [Use attributes src and otherContext.](#)

### Use the WebAppRegistry

You can access a page in another Web application by setting the `src` attribute to a string that starts with the name of a Web application registered in the WebAppRegistry, as follows:

```
src="webAppName: /path"
```

where the application's WebAppRegistry maps *webAppName* to a context root. The *path*-specified page is searched within that context root.

### Use the src attribute

To access a page in another Web application, set the `src` attribute to a full path that starts with a context path defined in `web.xml`. For example, this tag might appear in a JSP:

```
<dsp:include src="/Qui ncyFunds/en/guesthome.jsp">
```

where `/Qui ncyFunds` is a context path that is defined in `web.xml`.

You can configure the ATG platform to support multiple context paths. For example, the following fragment from `web.xml` defines two context paths delimited by colons: `/Qui ncyFunds` and `/dyn`:

---

```
<context-param>
  <param-name>atg.dynamo.contextPaths</param-name>
  <param-value>/Qui ncyFunds: /dyn</param-value>
</context-param>
```

---

**Note:** If you modify `web.xml`, you must reassemble, redeploy and restart the ATG platform application before changes take effect.

### Use attributes src and otherContext

Used together, the `src` and `otherContext` attributes let you access pages in other Web application by identifying a context path that is not specified in the application's `web.xml`. For example:

```
<dsp:include otherContext="/Qui ncyFunds" src="/en/guesthome.jsp">
```

This tag should resolve correctly to the desired JSP whether or not the including application's `web.xml` defines the context path `/Qui ncyFunds`.

## Passing Parameters to Embedded Files

An application typically contains embedded HTML and JSP files, which often rely on exchanging parameters. For example, two bicycle shops under the same ownership—Spinning Wheel and Power Wheel—run on the same site. The pages for each store use a header whose format and functionality are virtually identical, except for some customized content that identifies the individual store. In order to



share the same header file, the JSP for each store calls the header file with `dsp: include`, which passes a parameter that identifies the given store.

For example, the applicable code for the Spinning Wheel site's JSP looks like this:

```
<dsp: include page=header.jsp>
  <dsp: param name="storename" value="Spinning Wheel" />
</dsp: include>
```

The code for the sister store Power Wheel looks like this:

```
<dsp: include page=header.jsp>
  <dsp: param name="storename" value="Power Wheel" />
</dsp: include>
```

At runtime, the `dsp: include` tag passes in `storename` as a page parameter to `header.jsp`. This parameter is available to DSP tags in the header file, as in the following example:

```
<h1>Welcome to <dsp: valueof param="storename" /></h1>
```

At runtime, the `valueof` tag's `param` attribute resolves to the value supplied by the parent page—Spinning Wheel or Power Wheel—and displays in the included page header accordingly.

## 3 Using ATG Servlet Beans

Most applications require a way to dynamically generate HTML from a Java object. The `dsp:droplet` tag lets you do this by embedding an ATG servlet bean. The servlet bean produces output that is included in the HTML page.

ATG provides a standard set of ATG servlet beans that you can use in your JSPs. ATG product such as ATG Commerce provide additional servlet beans. Engineers at your site can also create servlet beans and make them available to you, as described in the [ATG Programming Guide](#).

This chapter focuses on several ATG servlet beans that are frequently used in JSPs, with an emphasis on concepts that apply to all servlet beans. For full information about these and other ATG servlet beans, see [Appendix B: ATG Servlet Beans](#).

This chapter includes the following topics:

- [Embedding ATG Servlet Beans in Pages](#)
- [Passing Component Names with Parameters](#)
- [Displaying Nested Component Properties with Servlet Beans](#)
- [Nesting Servlet Beans](#)
- [Other Commonly Used Servlet Beans](#)

### Embedding ATG Servlet Beans in Pages

This section shows how to use ATG servlet beans in JSPs with an example that uses [ForEach](#) servlet bean.

#### Using ForEach to Display Property Values

The [ForEach](#) servlet bean lets you iterate over the elements of an array property, and specify the desired HTML for displaying each element. ForEach also provides these options:

- Specify the HTML to display before and after array processing,
- Specify the HTML to display if the array has no elements.

For example, the following JSP fragment uses the ForEach servlet bean to loop through the elements in the array property `Student_01.subjects`:




---

```
<dsp: dropl et name="/atg/dynamo/dropl et/ForEach">
  <dsp: param name="array" bean="/sampl es/Student_01. subj ects"/>
  <dsp: oparam name="outputStart">
    <p>The student is registered for these courses: </p>
  </dsp: oparam>
  <ul >
    <dsp: oparam name="output">
      <li><dsp: val ueof param="el ement"></dsp: val ueof></li>
    </dsp: oparam>
  </ul >
</dsp: dropl et>
```

---

## Servlet Bean Parameters

Each servlet bean has a predefined set of parameters that control the servlet bean's behavior. Servlet beans have three types of parameters:

- [Input](#)
- [Output](#)
- [Open](#)

### *Input*

Input parameters are passed into the servlet bean. For example, the ForEach servlet bean requires the input parameter array to identify the type of data to process—in the following extract, the component array property Student\_01.subj ects:

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/ForEach">
  <dsp: param name="array" bean="/sampl es/Student_01. subj ects"/>
  <dsp: oparam name="outputStart">
    <p>The student is registered for these courses: </p>
  </dsp: oparam>
  <ul >
    <dsp: oparam name="output">
      <li><dsp: val ueof param="el ement"></dsp: val ueof></li>
    </dsp: oparam>
  </ul >
</dsp: dropl et>
```

---

### *Output*

Output parameters are set by the servlet bean. In the previous example, el ement is an output parameter that contains the value of the current array element. As the ForEach bean cycles through the subj ects array, el ement is successively set to each element value. Thus, if there are five elements in the array, on each loop iteration el ement is set to five different values.

## Open

Open parameters are marked by dsp: oparam tags, and specify code to execute at different stages of servlet processing. The previous example uses three ForEach servlet bean open parameters:

- outputStart executes just before loop processing begins. In the example, the open and close tags enclose HTML text and the <ul> tag to open the bulleted list.
- output executes for each array element. In the example, the open and close tags enclose a dsp: value tag that specifies to display the content of the current property element.
- outputEnd executes after loop processing is complete. In the example, it contains the close tag </ul> to close the bulleted list.

The HTML that is rendered by the example's open parameters might look like this:

---

```
<p>The student is registered for these courses:
<ul>
<li>Linguistics</li>
<li>Biology</li>
<li>Victorian Poetry</li>
<li>Latin</li>
<li>Philosophy</li>
</ul>
```

---

While output parameters are set by the servlet bean, they are rendered on the page only if they are referenced in an open parameter. In this example, the output parameter element is set by the servlet bean, but the dsp: valueof tag in the open parameter is used to display the element values.

**Note:** Certain HTML tags cannot overlap different open parameters. For example, the open and close tags of dsp: form—<dsp: form> and </dsp: form>—must both be embedded in the same open parameter. This constraint applies to the following DSP tags:

```
<dsp: a href="...">
<dsp: form ...>
<dsp: textarea ...>
<dsp: select>
```

## Parameter Scope

In general, the values of input, output, and open parameters are local to the servlet bean that uses them, and are not directly accessible outside the open and close dsp: droplet tags.

If the servlet bean has a parameter with the same name as the page parameter, the servlet bean parameter value temporarily overrides the page parameter within the servlet bean.



## Accessing Servlet Bean Parameters with EL

You can make servlet bean parameters available to other EL-enabled tags by setting the servlet bean's `var` attribute:

---

```
<dsp: dropl et name="servl et-bean-path" var="attr-name">
  ...
</dsp: dropl et>
```

---

The `var` attribute creates a Map that is composed of the parameters and their values defined in the enclosing `dsp: dropl et` tag, and provides a name to that Map.

For example:

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/ForEach" var="fe">
  <dsp: param name="array" bean="/sampl es/Student_01. subj ects"/>
  <dsp: oparam name="outputStart">
    <p>The student is registered for these courses:
    <ul >
  </dsp: oparam>
  <dsp: oparam name="output">
    <li><c: out val ue="${fe. el ement}"/></li>
  </dsp: oparam>
  <dsp: oparam name="outputEnd">
    </ul >
  </dsp: oparam>
</dsp: dropl et>
```

---

In this example, the JSTL Core tag `c: out` replaces the `dsp: val ueof` tag and uses EL syntax to render the property contents.

A servlet bean's `var` attribute can be set either to `page` (the default) or `request` scope. In general, it is good practice to confine the scope of a servlet bean's `var` attribute to the current page.

### *Scope of servlet bean parameters*

If you set EL variables for nested servlet beans, each variable is accessible only within the bean where it was set and to its parameters, even if several variables use the same name. For example, in the following example, the `dsp: dropl et` tags for both servlet beans set the EL variable `fe`. Each instance of `fe` has scope only within its own bean; so, the first-set `fe` is accessible only to outer-bean parameters, while the second instance of `fe` is accessible only to inner-bean parameters:

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/ForEach" var="fe">
  ...
  <dsp: dropl et name="/atg/dynamo/dropl et/ForEach" var="fe">
    ...
  </dsp: dropl et>
```

---



```
...  
</dsp: droplet>
```

---

For more information, see [Nesting Servlet Beans](#).

## Importing Servlet Beans

You can use `dsp: importbean` to import a servlet bean into a JSP, in order to reference it more easily. For example:

```
<dsp: importbean bean="/atg/dynamo/droplet/ForEach" />
```

Subsequent references to the servlet bean can omit its full path, as follows:

```
<dsp: droplet name="ForEach">
```

## Passing Component Names with Parameters

The previous example is hard-coded to display properties of the `Student_01` component. You can modify this example so it displays properties of any component of class `Student`. This is typically done through a query string in the calling URL that contains the desired property value—obtained, for example, via a form—which is accessible to the JSP as a page parameter.

The following example obtains the page parameter `StudentID` and uses `dsp: getvalueof` to set the EL variable `currentStudent`. This parameter is used to access a specific `Student` component and use its `subjects` property to set the servlet bean's array input parameter:

---

```
<dsp: param bean="/samples/Student1" name="currentStudent" />  
<p>Student's name: <dsp: valueof param="currentStudent.name" />  
<p>Student's age: <dsp: valueof param="currentStudent.age" />  
  
<dsp: droplet name="/atg/dynamo/droplet/ForEach">  
  <dsp: param param="currentStudent.subjects" name="array" />  
  <dsp: oparam name="outputStart">  
    <p>The student is registered for these courses:  
    <ul>  
  </dsp: oparam>  
  <dsp: oparam name="output">  
    <li><dsp: valueof param="element" /></li>  
  </dsp: oparam>  
  <dsp: oparam name="outputEnd">  
    </ul>  
  </dsp: oparam>  
</dsp: droplet>
```

---





**Note:** The `currentStudent` parameter is a page parameter available throughout the page. It is not a parameter of the servlet bean itself, although in this example it is used to set the value of the `ForEach` servlet bean's array parameter. In this example, the value of the `currentStudent` parameter is set explicitly, but in an actual site you would typically set the parameter dynamically, based on values the user enters in a form, other pages visited, or other criteria.

## Displaying Nested Component Properties with Servlet Beans

Previous examples show how to pass an array of `String` values to the `ForEach` servlet bean and display elements of that array. This section shows how to pass an array of `Nucleus` components into a `ForEach` servlet bean, then use the servlet bean to display a specific property of each component.

For example, the `Student` components used in earlier examples might be complemented by `Dormitory` components, which contain two properties:

- `name`: a `String` that stores the name of the dormitory
- `students`: an array of `Student` components

The `Dormitory` class instantiated by dormitory components looks like this:

---

```
public class Dormitory {
    String name;
    Student[] students;

    public Dormitory () {}
    public void setName (String name) { this.name = name; }
    public String getName () { return name; }
    public void setStudents(Student[] students) { this.students = students; }
    public Student[] getStudents() { return students; }
}
```

---

In the following example, the `ForEach` servlet bean iterates through an array of `Student` components that are in a `Dormitory` component, and displays each `Student` component's `name` property:

---

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
    <dsp: param name="array" bean="/dorms/MarkIey.students"/>
    <dsp: param name="sortProperties" value="+age"/>
    <dsp: oparam name="outputStart">
        <p>The following students live in
            <dsp: valueof bean="/dorms/MarkIey.name"/>: <ul>
        </dsp: oparam>
    <dsp: oparam name="output">
        <li><dsp: valueof param="element.name"/> &nbsp; &nbsp;
```

```

      dsp: val ueof param="element.age"/> </li>
    </dsp:oparam>
    <dsp:oparam name="outputEnd">
      </ul>
    </dsp:oparam>
  </dsp:droplet>

```

In this example, the array input parameter points to `Mark1` (e.g., `students`), an array of `Student` components. As the `ForEach` servlet bean iterates through this array, each `element` output parameter points to the next `Student` component; the `jsp:val ueof` tag displays the `name` property of each `Student` component.

## Sorting elements on properties

This example uses the input parameter `sortProperties`, which specifies to sort elements on the `Student` component's `age` property, in ascending order.

In the compiled JSP, the `ForEach` servlet generates output like this:

```
<p>These students live in Markley Hall:  
<ul>  
<li>Susan Williams 19</li>  
<li>Frank Thomas 20</li>  
<li>Dwayne Hickman 23</li>  
</ul>
```

**Note:** If servlet beans are nested, an inner servlet beans inherits from the outer servlet beans the most proximate `sortProperties` setting, unless it has its own `sortProperties` setting.

## Nesting Servlet Beans

The HTML produced by open parameters can contain dynamic elements such as `dsp: val ueof` tags. The open parameters of one servlet bean can also include other servlet beans.

The following example specifies to display a list of students who live in a dormitory, and under each student name list registered courses. It does this by nesting ForEach servlet bean: the outer servlet bean displays the student names, and the inner servlet bean displays the courses.

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param bean="/dorms/MarkIey.students" name="array"/>
  <dsp: oparam name="output">
    <p><dsp: val ueof param="element.lastName"/>&nbsp;
      <dsp: val ueof param="element.firstName"/>&nbsp; ; </p>
```



```

<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param param="element.subjects" name="array" />
  <dsp: oparam name="outputStart"> <ul>
    <dsp: oparam name="output">
      <li><dsp: valueof param="element" /> </li>
    </dsp: oparam>
  <dsp: oparam name="outputEnd"> </ul>
</dsp: droplet>

</dsp: oparam>
</dsp: droplet>

```

---

When this page is compiled, the following actions occur:

1. The outer ForEach sets its array parameter to `MarkIey.students`, an array of Student components. The outer ForEach element points to the first Student component in this array.
2. The outer ForEach bean displays the current value of `element.name`, the name property of the current Student component.
3. The inner ForEach bean sets its array parameter to `element.subjects`. This parameter points to the first subjects String in the current Student component.
4. The inner ForEach bean iterates over the elements of its array parameter, displaying each String in `element.subjects`.

**Note:** The `element` parameter of a given servlet bean has scope only within that bean; changes to its value have no effect on the `element` parameters of servlet beans that are nested outside and inside the current bean.

5. After the inner ForEach processes all elements in `element.subjects`, it returns control to the outer ForEach.
6. The outer ForEach element parameter points to the second element of the `MarkIey.students` array.
7. Steps 3-5 repeat, this time using the next Student component.
8. This process continues for each component in the `MarkIey.students` array.

The previous example generates HTML as follows:

```

<p>Williams, Susan: </p>

<ul>
<li>Linguistics</li>
<li>Biology</li>
<li>Latin</li>
<li>Philosophy</li>
</ul>

<p>Thomas, Frank: </p>

```



```
<ul>
<li>Physics
<li>American History
<li>Psychology
<li>French
</ul>
```

...

---

### ***Referencing parameters in outer servlet beans***

Servlet bean parameters have scope only within the bean itself. When nesting occurs, the parameters within each bean are unaffected by corresponding parameters in the outer- and inner-nested beans. However, an inner bean can access the values of an outer bean parameter by using to define another parameter in the outer servlet bean, and setting it to the value of a parameter in the current bean that the inner beans can access.

For example, you can modify the JSP shown earlier and obtain similar output, as follows:

---

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param bean="/dorms/Markley.students" name="array"/>
  <dsp: oparam name="output">
    <dsp: setvalue param="outerElement" paramvalue="element"/>

    <dsp: droplet name="/atg/dynamo/droplet/ForEach">
      <dsp: param param="element.subjects" name="array"/>
      <dsp: oparam name="outputStart">
        <p><dsp: valueof param="outerElement.lastName"/>&nbsp;  ,
          <dsp: valueof param="outerElement.firstName"/>&nbsp;  ; </p>
        <ul>
          <dsp: oparam name="output">
            <li><dsp: valueof param="element"/> </li>
          </dsp: oparam>
        <dsp: oparam name="outputEnd"> </ul>
      </dsp: droplet>

    </dsp: oparam>
  </dsp: droplet>
```

---

## **Other Commonly Used Servlet Beans**

The ForEach servlet bean used in the previous sections is one of many servlet beans that ATG provides. Among those most often used



- [Looping servlet beans](#)
- [Switch servlet bean](#)

### ***Looping servlet beans***

Several standard servlet beans are similar to `ForEach`: they also provide ways to loop through array elements. They differ from `ForEach` primarily in the portion of the array they render, or how they format output. For example:

- [Range](#) renders a subset of array elements.
- [TableForEach](#) produces output in a two-dimensional format such as an HTML table.
- [TableRange](#) renders a subset of array elements, and produces its output in a two-dimensional format.

For a descriptions of all ATG servlet beans, see [Appendix B: ATG Servlet Beans](#).

### ***Switch servlet bean***

[Switch](#) is a commonly used servlet bean that tests multiple open parameters against the input parameter `val ue`; if the open parameter and `val ue` parameter are identical, the servlet executes the open parameter's body.

For example, your site might recognize different user types and render content accordingly. You typically use Switch servlet beans to handle this.

Switch takes an input parameter called `val ue` and renders the open parameter whose name matches `val ue`. A Switch servlet bean can have any number of open parameters, depending on the number of possible values of `val ue`. You can optionally include an open parameter named `default` that is rendered if `val ue` does not match the name of any other open parameter.

In the following example, the Switch servlet bean gets the value of `currentStudent.gradtype`. If the value of this parameter is `grad` or `undergrad`, Switch executes the Redirect servlet in the corresponding `oparam` body and displays the appropriate JSP. If no match is found in any open parameter, Switch renders the `default` open parameter, which redirects the user to the welcome page:

---

```
<dsp: droplet name="/atg/dynamo/droplet/Switch">
  <dsp: param name="val ue" param="currentStudent.gradtype"/>

  <dsp: oparam name="default">
    <dsp: droplet name="/atg/dynamo/droplet/Redirect">
      <dsp: param name="url" val ue="http://www.umvs.edu/welcome.jsp"/>
    </dsp: droplet>
  </oparam>

  <dsp: oparam name="undergrad">
    <dsp: droplet name="/atg/dynamo/droplet/Redirect">
      <dsp: param name="url" val ue="http://www.umvs.edu/enrollment_ugrad.jsp"/>
    </dsp: droplet>
  </oparam>
```



```
<dsp:oparam name="grad">
  <dsp:droplet name="/atg/dynamo/droplet/Redirect">
    <dsp:param name="url" value="http://www.umvs.edu/enrollment_grad.jsp"/>
  </dsp:droplet>
</oparam>
</dsp:droplet>
```

---

See Switch in [Appendix B: ATG Servlet Beans](#) for more information.



## 4 Coding a Page for Multiple Sites

You can code JSPs for a multisite application so they are sensitive to site context. Site context determines access to site-aware repository items, and especially to ATG Commerce product items, which are encoded with site context IDs. Site context can also affect component properties.

By default, a JSP executes within the current site context. You can encode a page—entirely or in portions—to switch site context. For example, a page that opens in one site context can temporarily switch site context for a specific portion of the page in order to display products from another site.

This chapter discusses the following page coding tasks:

- [Obtaining product sites](#)
- [Getting site information](#)
- [Evaluating site sharing](#)
- [Changing site context](#)
- [Generating links to other sites](#)

**Note:** This chapter assumes that you are familiar with multisite requirements, architecture, and terminology. For more information, see:

- [ATG Programming Guide](#): Describes multisite request processing.
- [ATG Multisite Administration Guide](#): Provides a general overview of multisite architecture and detailed information about setting up a multisite environment.

### Obtaining Product Sites

Because a product can be accessed on multiple sites, it is often important for a page to obtain the correct site ID before enabling user access to product information. For example, you might define cross-sell and upsell lists that include items that belong to multiple sites. When a user selects an item, the application must be able to determine the best site ID to return for the selected item.

Given a product item, the servlet bean `SiteldForItem` returns a site ID from a list of site IDs that are stored in the item's `siteIds` property. The servlet bean chooses a site ID according to various input parameters and site priority settings, as described below. This site ID can be used to obtain useful information about the site itself. It can also be used to change site context and generate links to the site.

## Site Selection Algorithm

You can modify `SiteldForItem`'s site selection algorithm by setting one or more input parameters. By default, `SiteldForItem` only returns an enabled site. If desired, you can allow return of a site that is both inactive and disabled site by setting the input parameters `includeInactiveSites` and `includeDisabledSites` to `true`.

Among all sites that are eligible to be returned, `SiteldForItem` determines its selection by prioritizing the sites as follows:

1. The only site in the item's `sitelids` list.
2. The current site, if input parameter `currentSiteFirst` is set to `true`.
3. The site ID specified by input parameter `siteId`.
4. A site in a site group that shares the `ShareableType` specified by input parameter `shareableTypeId`.

## Site Priority and Precedence

A site's configured priority can affect its precedence in the `SiteldForItem` selection process. For example:

- Product item `myProduct` has its `sitelids` property set to `siteA` and `siteB`.
- `siteA` is the current site.
- `siteB` has a higher priority than `siteA`:
 

```
siteA
  sitePriority=2

siteB:
  sitePriority=1
```
- `siteA` and `siteB` are both active.

Priority settings being equal, the following example of `SiteldForItem` generally gives precedence to the current site—`siteA`—over any other site. However, given `siteA` and `siteB`'s priority settings, `SiteldForItem` gives precedence to `siteB` over `siteA` and returns `siteB`'s site ID:

---

```
<dsp:importbean bean="/atg/dynamo/droplet/multisite/SiteldForItem"/>
<dsp:droplet name="SiteldForItem">
  <dsp:param name="item" param="myProduct"/>
  <dsp:param name="currentSiteFirst" value="true"/>
  ...
</dsp:droplet>
```

---

## SiteldForItem Example

The following example shows how you might use `SiteldForItem` with [SiteLinkDroplet](#) to identify a product's site ID, then generate an absolute URL to the product on that site. The generated URL is used to set a request-scope variable that other JSPs can access:





```

<dsp: page>
<%--
    Generate a cross site URL and set in request scope parameter 'siteLinkUrl' so
    it's accessible to the calling JSP

    Expected page parameters:
        product - product item
        siteId - the site to use in generated URL; if not supplied, use SiteIdForItem
                  to obtain the site ID from the product item.
--%>
<dsp: importbean bean="/atg/dynamo/droplet/multiSite/SiteLinkDroplet"/>
<dsp: importbean bean="/atg/commerce/multiSite/SiteIdForItem"/>

<dsp: getvalueof var="product" param="product"/>
<dsp: getvalueof var="siteId" param="siteId"/>

<c: choose>

    <%-- No site ID supplied, so get one from SiteIdForItem --%>
    <c: when test="${empty siteId}">
        <dsp: droplet name="SiteIdForItem">
            <dsp: param name="item" param="product"/>
            <dsp: param name="shareableTypeId" value="atg.ShoppingCart"/>
            <dsp: param name="currentSiteFirst" value="true"/>
            <dsp: oparam name="output">
                <dsp: getvalueof var="productSiteId" param="siteId"/>

                <dsp: droplet name="SiteLinkDroplet">
                    <dsp: param name="siteId" value="${productSiteId}"/>
                    <dsp: param name="path" param="product.template.url"/>
                    <dsp: oparam name="output">
                        <dsp: getvalueof var="siteLinkUrl" scope="request" param="url"/>
                    </dsp: oparam>
                </dsp: droplet>
            </dsp: oparam>
        </dsp: droplet>
    </c: when>

    <%-- Site ID supplied --%>
    <c: otherwise>
        <dsp: droplet name="SiteLinkDroplet">
            <dsp: param name="siteId" value="${siteId}"/>
            <dsp: param name="path" param="product.template.url"/>
            <dsp: oparam name="output">
                <dsp: getvalueof var="siteLinkUrl" scope="request" param="url"/>
            </dsp: oparam>
        </dsp: droplet>
    </c: otherwise>
</c: choose>

```



```
</dsp: page>
```

---

## Getting Site Information

Given a valid site ID, the servlet bean [GetSiteDroplet](#) returns a site object—an implementation of interface `atg.multiplesite.Site`—which encapsulates a site configuration. The output parameter enables access to that site's properties. If you supply null for the site ID, `GetSiteDroplet` returns the current site.

The following JSP code uses `GetSiteDroplet` to obtain the site `mySite`'s configuration. The output parameter `site` provides access to all site configuration properties. In this example, the code obtains the site's `closingDate` property and compares it to the current date to determine whether the site is active:

---

```
<!-- Get a site's configuration and look at its closingDate property --%>
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ page import="java.util.Date;"%>
<dsp: page>

<dsp: droplet name="/atg/dynamo/droplet/multiple/GetSiteDroplet">
  <dsp: param name="siteId" value="mySite"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="closingDate" param="site.closingDate"
      vartype="java.util.Date">
      <c: choose>
        <c: when test="${System.currentTimeMillis() < closingDate.getTimeInMillis()}">
          Site is still active
        </c: when>
        <c: otherwise> Site is no longer active </c: otherwise>
      </c: choose>
    </dsp: oparam>
  </dsp: droplet>
</dsp: page>
```

---

### Getting the Current Site

You can obtain the current site and its configuration through the request-scoped Nucleus component `/atg/multiple/Site`. For example, you can modify the previous example and obtain the current site's `closingDate` property as follows:

---

```
<!-- Get the current site configuration and look at its closingDate property --%>
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ page import="java.util.Date;"%>
<dsp: page>
```

---



```
<droplet bean="/atg/dynamo/droplet/ComponentExists">
  <oparam name="true">
    <dsp: tomap bean="/atg/multisite/Site.id" var="siteId"/>
    <dsp: getvalueof var="closeDate" param="siteId.closeDate"
      vartype="java.util.Date">
      <c: choose>
        <c: when test="${System.currentTimeMillis() < closeDate.getTimeInMillis()}">
          Site is still active
        </c: when>
        <c: otherwise> Site is no longer active </c: otherwise>
      </c: choose>
    </oparam>
    <oparam name="false">SiteContextManager not found</oparam>
  </droplet>
</dsp: page>
```

---

## Evaluating Site Sharing

Two servlet beans let you test which sites are in the same site group and share data as specified by the `ShareableType` components that are configured for the group:

- [SitesShareShareableDroplet](#) tests whether two sites are in the same sharing group and therefore have equal access to the data of objects that are shared within the group. For example, before changing the site context on a given page, you might first want to check whether the two sites are in a sharing group that accesses the same shopping cart.
- [SharingSitesDroplet](#) identifies a sharing group and returns its member sites.

For detailed information, see the documentation for these servlets in [Appendix B, ATG Servlet Beans](#).

### *SitesShareShareableDroplet Example*

The following example uses `SitesShareShareableDroplet` to test whether the current site and another site are in the same sharing group and access the same shopping cart:

---

```
<dsp: droplet name="SitesShareShareableDroplet">
  <dsp: param name="shareableId" value="ShoppingCart"/>
  <dsp: param name="otherSiteId" value="rcSite"/>
  <dsp: oparam name="true">
    The current site shares a shopping cart with Really Cool Site\!
  </dsp: oparam>
  <dsp: oparam name="false">
    The current site doesn't share a shopping cart with Really Cool Site\!
  </dsp: oparam>
  <dsp: oparam name="error">
    Only called if the SiteGroupManager isn't set.
```



The following error occurred:

```
<dsp: valueof param="errorMessage" />
</dsp: oparam>
</dsp: droplet>
```

---

## Changing Site Context

In a multisite environment, it might be desirable to provide access to another site while handling the current site's request. For example, selection of a given product might trigger display of related products from another site. You can allow users to retrieve additional product information from that site by temporarily switching the current page's site context.

A page can switch site context through the servlet bean [SiteContextDroplet](#). This servlet bean resets the site context as directed by the supplied site ID, and its output parameter acts as a wrapper for the new site context. Before the output parameter executes, the `SiteContextManager` uses the site ID to push a new `SiteContext` object onto the `SiteContext` stack and establish it as the new site context.

The changed site context remains in effect until `SiteContextDroplet` exits; the `SiteContextManager` then pops the site context off the stack and restores the previous site context to the page. For detailed information about site context, see the [ATG Programming Guide](#).

### Empty Site Context

You can use `SiteContextDroplet` to set an empty site context by setting the input parameter `emptySite` to `true`. For the duration of output parameter execution, the page's site context is empty and the page has access to repository data of all sites.

If desired, you can set an empty site context for an entire page by wrapping it entirely inside `SiteContextDroplet`'s output parameter and setting `emptySite` to `true`.

### SiteContextDroplet Example

The following page code uses `SiteContextDroplet` to change the site context to `site1`. It then invokes the servlet bean [TargetingFirst](#) within the `SiteContextDroplet` output parameter:

---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<dsp:importbean bean="/atg/dynamo/droplet/multisite/SiteContextDroplet"/>
<dsp:importbean bean="/atg/targeting/TargetingFirst"/>

<dsp:page>
<html>
<body>

<dsp:droplet name="SiteContextDroplet">
  <dsp:param name="siteId" param="site1"/>
  <dsp:oparam name="output">
```



```

<dsp: droplet name="TargetingFirst">
  <dsp: param name="targeter" bean="/atg/multi site/SiteTargeter"/>
  <dsp: oparam name="output">
    <BR/>
  </dsp: oparam>
  <dsp: oparam name="empty">
    No content returned from the Targeter<BR/>
  </dsp: oparam>
</dsp: droplet>
</dsp: oparam>
<dsp: oparam name="error">
  Unable to set site context. Error: &nbsp;
  <dsp: valueof param="errorMessage" />
</dsp: oparam>
</dsp: droplet>

</body>
</html>
</dsp: page>

```

## Generating Links to Other Sites

The servlet bean [SiteLinkDroplet](#) can generate URLs that reference other sites. You can use this servlet bean to write page code that generates links to different sites and paths, depending on the current site context.

SiteLinkDroplet uses input parameters `siteid` and `path` to generate a URL as follows:

If input parameter...	Contains...	Then...
<code>siteid</code>	Site ID	Use production URL configured for this site.
<code>siteid</code>	(parameter omitted)	Use the current site.
<code>path</code>	Leading forward slash (/)	Replace old path with new path relative to site root.
<code>path</code>	No leading forward slash (/)	Make specified path relative to current page.
<code>path</code>	(parameter omitted)	Retain old path relative to site root.

The output parameter `url` contains the generated URL, which you can use to set a link to the desired site.

SiteLinkDroplet also provides the following optional input parameters:

- `queryParams`: Appends the specified query parameters to the generated URL.



- `protocol` : Specifies the protocol to use—`http` or `https`—in the generated URL. If omitted, the protocol is set from the `SiteURLManager` property `defaultProtocol`.
- `include`: If `SiteLinkDroplet` is invoked inside an included page, you can set `include` to `true` in order to specify that the generated URL contains the included page's path relative to the parent page.

The following examples demonstrate two use cases:

- [Site ID specified, path omitted](#)
- [Site ID omitted, path replaced](#)

### ***Site ID specified, path omitted***

In the following example, the following conditions apply:

- Current page URL:  
`http://FirstSite.com/dir/index.jsp`
- Production site URL of site `mySite`:  
`SecondSite.com`.

---

```
<dsp: droplet name="SiteLinkDroplet">
  <dsp: param name="siteId" value="mySite"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="newUrl" param="url" vartype="java.lang.String">
      <dsp: a href="{newUrl}">
        Click Here! </dsp: a>
      </dsp: getvalueof>
    </dsp: oparam>
  </dsp: droplet>
```

---

This generates the following link:

```
<a href="http://SecondSite.com/dir/index.jsp">Click Here! </a>
```

### ***Site ID omitted, path replaced***

If the current page is `http://FirstSite.com/dir/index.jsp`, the following code generates this link:

```
<a href="http://FirstSite.com/home/index.jsp">Click Here! </a>
```

---

```
<dsp: droplet name="SiteLinkDroplet">
  <dsp: param name="path" value="/home/index.jsp"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="url" param="url" vartype="java.lang.String">
      <dsp: a href="{url}">
        Click Here! </dsp: a>
      </dsp: getvalueof>
    </dsp: oparam>
  </dsp: droplet>
```

---



```
</dsp:oparam>  
</dsp:droplet>
```

---







## 5 Serving Targeted Content with ATG Servlet Beans

ATG Personalization lets you customize content for specific users and events, primarily through two mechanisms:

- Content targeters that match content to specific users.
- Slots that display their content in the context of certain events, or *scenarios*.

For example, a site that covers sporting events might use a content targeter to show users articles about those sports that match the locale specified in their individual profiles. The targeter might have these rules:

```
Show this content:  
  redsoxnews.jsp  
to these people:  
  people in group BaseballLovers  
  whose homeAddress.city is Boston
```

The same site might also use slots in order to determine the content that users see when they navigate to certain pages—for example, slots that highlight certain items that are relevant to the current page, such as new items of sports clothing, or clearance items for camping equipment.

This chapter describes servlet beans for incorporating targeted content into JSPs. Each targeting servlet bean gathers the items defined by a specific targeter, and displays all or a subset of them.

For specific information about targeting servlet beans, see [Appendix B: ATG Servlet Beans](#). For information about creating targeted content components such as content targeters and slots, see the [ATG Personalization Guide for Business Users](#).

### Targeting Servlet Beans

The ATG platform provides five targeting servlet beans. Four of them use a targeter to gather a set of items from a content repository, and render the items on the page. They differ only in how many items they actually display:

- [TargetingForEach](#) displays all of the items returned by the targeter.
- [TargetingFirst](#) displays the first *n* items, where *n* is a number you specify.



- [TargetingRange](#) displays a range of items, such as the third through the seventh.
- [TargetingRandom](#) displays  $n$  items chosen randomly.

[TargetingArray](#) differs from the others in that it does not format the output of the targeting operation. Instead, you must use another servlet bean to format the output.

The following sections describe how to control the output of these servlet beans, covering the following topics:

- [Sorting results](#)
- [Controlling event logging](#)
- [Limiting result sets](#)

## Sorting Results

ATG Personalization has two ways to sort the results of a targeting operation:

- The targeter's rules set, which can set the sort order for output from the targeting operation. If specified, the targeter's sort order determines the order in which items are retrieved from the repository.
- The targeting servlet bean can sort the result set through its `sortProperties` parameter. A sort order specified in the targeting servlet bean with the `sortProperties` parameter determines the order the items are presented for display.

It might be desirable to specify sorting in both places. For example, a targeter can retrieve the first five matching items in order of publication date, then sort publications by author as follows:

1. Create a targeter that retrieves a set of items and sorts them by publication date.
2. Use the `TargetingFirst` servlet bean with this targeter and specify to retrieve five items, and to sort output by author.

The tags for the servlet bean might look like this:

---

```
<dsp: droplet name="/atg/targeting/TargetingFirst">
  <dsp: param name="targeter"
    bean="/atg/registry/RepositoryTargeters/Features/AllFeatures"/>
  <dsp: param name="howMany" value="5"/>
  <dsp: param name="sortProperties" value="+author"/>
  <dsp: oparam name="outputStart"> <ul></dsp: oparam>
  <dsp: oparam name="output">
    <li><dsp: valueof param="element.headline"/></li>
  </dsp: oparam>
  <dsp: oparam name="outputEnd"> </ul></dsp: oparam>
</dsp: droplet>
```

---



You can set the value of the `sortProperties` parameter to a string that specifies how to sort the list of items in the output array. Sorting can be performed on JavaBean and Dynamic Bean properties, and on Dates, Numbers, and Strings.

### **Sorting on JavaBeans and Dynamic Beans**

To sort on JavaBean and Dynamic Bean properties, specify the value of `sortProperties` as a comma-separated list of property names, as follows:

```
<dsp: param name="sortProperties" value="{+|-}property-name[, ...]" />
```

The first instance of *property-name* specifies the primary sort, the second specifies the secondary sort, and so on. Each property name is prepended by a plus (+) or minus (-) sign to indicate whether to sort in ascending or descending order.

The following example specifies to sort an output array of JavaBeans by its `title` property in ascending order, then by its `size` property in descending order:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

### **Sorting on Dates, Numbers and Strings**

To sort on Dates, Numbers, and Strings, set `sortProperties` to a + (ascending) or – (descending) sign, as in the following example, which sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

### **Sorting on a Map key**

In order to sort elements of a Map, set `sortProperties` to the Map key as follows:

```
<dsp: param name="sortProperties" value="-_key" />
```

## **Controlling Event Logging**

Each targeting servlet bean described in this chapter triggers a content view event and a content-type view event for each content item returned by the targeter. For example, a page with a `TargetingForEach` servlet bean returns an array of seven content items. When the ATG platform serves this page, seven content view events and seven content-type view events are triggered and sent to the Event Distributor's content channel and content-type channel, respectively.

You can disable this behavior for any targeting servlet bean by setting one or both of the following parameters in the servlet:

```
<dsp: param name="fireContentEvent" value="false" />
<dsp: param name="fireContentTypeEvent" value="false" />
```

For more information, see the *Personalization Module Logging* chapter of the [ATG Personalization Programming Guide](#).

## Limiting Result Sets

Some targeting operations might return a large number of repository items. To prevent a servlet bean from loading large results sets into memory, limit the result set by setting its `maxNumber` input parameter. For example, you can set the maximum number of items returned by a `TargetingForEach` servlet bean as follows:

```
<dsp: param name="maxNumber" value="100" />
```

By default, the `maxNumber` property is set to `-1`, which specifies no limit to the targeting result set size.

## Using Slots

A slot is a Nucleus component that you can use, in conjunction with scenarios, to set up and display a series of personalized content items on one or more Web pages. Slots are containers that you use to display and manage dynamic items on your Web site. You use targeting servlet beans to include slots in site pages, and you use scenarios to fill them with content.

Slots provide some powerful features not available through targeters:

- Slots can be integrated with scenarios, providing more flexibility and control over delivery of dynamic content. With scenarios, you can set up an empty slot that generates its own request for content when a site visitor displays the page.
- Slots can display content other than repository items.
- Slots have better caching capabilities than targeters, facilitating faster display.

You typically create a slot in the ATG Control Center, as described in the *Using Slots* chapter in the [ATG Personalization Programming Guide](#). After you create the slot, you integrate the slot into your application as follows:

1. Add the slot to the JSP through the appropriate targeting servlet bean—for example, `TargetingForEach`.
2. [Create a scenario for the slot](#) that specifies the content to display in the slot and the circumstances when it appears.

### Create a scenario for the slot

A scenario can perform these tasks:

- Defines the events and circumstances that cause content to appear in the slot.
- Specifies the content that appears.

**Note:** Instead of using a scenario to determine slot content, you can use an existing content targeter.

For example, you might want to display a series of messages—a Welcome Back message and a list of new pages—to site members who log on to your site after an absence of eight or more weeks. To do so, you create a `WelcomeBack` slot and a scenario that performs these tasks:



- Tests whether eight weeks elapsed since a user's last login.
- If the first condition evaluates to true, displays the appropriate items in the WelcomeBack slot.

**Note:** A slot can contain only items of the same type from a single repository. (The content item type is a setting that your database administrator or application developer defines when he or she sets up a repository. For more information, see the [ATG Repository Guide](#).) The item type that you can include in a particular slot is specified by the `itemDescriptorName` property in the slot component. The repository is specified by the `repositoryName` property in the slot component.

You use the Control Center to create and enable scenarios for slots. For more information, see the [ATG Personalization Guide for Business Users](#).

## Setting Pages To Support Scenarios

Creating scenarios is a task that is often assigned to business users. Some scenarios use elements that listen for events fired from pages, or respond to changes in component properties. Such scenarios require you to design pages with these tasks in mind.

### FormSubmission Event

A scenario can include a `FormSubmission` element, which listens for a `FormSubmission` event and responds by triggering the next scenario element. A `FormSubmission` event occurs on activation of a form submit control that has a form handler method associated with it. If the form handler's `sendMessage` property is set to true, the form handler method fires the `FormSubmission` event, and the scenario's `FormSubmission` element responds accordingly.

You can set the `sendMessage` property to true by defining a hidden input as follows:

```
<dsp:input type="hidden" bean="MyFormHandler.sendMessage" value="true" />
```

### Filtering FormSubmission events

A `FormSubmission` element can be configured to listen only for messages sent by a specific form handler component, such as `ProfileFormHandler`. Each `FormSubmission` event sets a `formName` parameter that is set to the name of its form handler's component name. This parameter enables `FormSubmission` scenario elements to respond only to the desired events.

For more information, see the *FormSubmission Event* section of the *Using Scenario Events* chapter of the [ATG Personalization Guide for Business Users](#).

### Clicks a Link

`Clicks a Link` scenario elements can track links that site visitors click, and their navigation habits. A `Clicks a Link` scenario element is enabled by setting the `dsource` parameter in a `dsp: a` anchor tag. For example:



```
<dsp: a href="content/month_discounts.jsp">
  <dsp: param name="dsource" value="OnSale, MemberDiscount" />
  Click here to see this month's discounts!
</dsp: a>
```

The values that you specify for the `dsource` parameter—in this example, `OnSale` and `MemberDiscount`—are used to name a collection of data stored about that link. You can use a different value for each link or reuse a value such as `onsale` for similar links.

The Clicks a Link element contains the following optional parameters:

This parameter...	Triggers the Clicks a Link element when...
anywhere	A site visitor clicks any link whose anchor tag contains a <code>dsource</code> parameter.
from page	A site visitor clicks any link on the specified page, if the link contains a <code>dsource</code> parameter.  Examples: <i>Clicks a link from page named /welcome.jsp</i> <i>Clicks a link from page in folder /content/today/</i>
to page	A site visitor clicks any link that goes to the specified page, if the link contains a <code>dsource</code> parameter.  Examples: <i>Clicks a link to page named /thismonthonly.jsp</i> <i>Clicks a link to page in folder /content/monthlyspecials</i>
where source name list	A site visitor clicks a link with a specific <code>dsource</code> parameter.  Examples: <i>Clicks a link where source name list includes OnSale.</i> <i>Clicks a link where source name list includes any of [OnSale, MemberDiscount, MonthlyPromo]</i>

For more information, see the *ClickThrough Event* section of the *Using Scenario Events* chapter of the [ATG Personalization Programming Guide](#).



## 6 Integrating XML With ATG Servlet Beans

The ATG platform provides servlet beans for transforming data in XML documents. These servlet beans let you display all or part of an XML document in a JSP, or make the data in an XML document available to the ATG platform as a DOM document.

This chapter includes the following sections:

- [XML, XSLT, and DOM](#)
- [XML Transformation](#)
- [Transforming XML Documents with Templates](#)
- [XML Transformation and Targeting](#)
- [Passing Parameters to XSL](#)
- [XSL Output Attributes](#)

### XML, XSLT, and DOM

XML is a markup language for documents that contain structured information. An XML document is a tree of nodes, where each node can have one or more attributes. It can also have one or more child nodes, each of them a different type than its parent.

XML specifies the structure of a document; it does not specify how to display it. XSLT is a language for transforming XML documents—for example, into a human-readable format such as HTML. ATG products provide XSLT and JSP templates that transform XML documents for display.

For more about XML and XSLT, see [www.w3.org/XML](http://www.w3.org/XML) and [www.w3.org/TR/xslt](http://www.w3.org/TR/xslt).

#### ***XML and DOM***

The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML and XML formats. It defines the logical structure of documents and how they are accessed and manipulated.

The DOM represents an XML document as a tree of nodes, where a DOM document is an instance of `org.w3c.dom.Document`. Each node in a DOM document is an instance of some subclass of the Node class `org.w3c.dom.Node`. Node subclasses include `Element`, `Text`, and `Comment`. Node properties

include `nodeName`, `nodeType`, `nodeValue`, `childNodes`, `firstChild`, and `lastChild`. The `Node` interface provides methods that let you determine whether a node has children, its type, and its value. You can use these methods to iterate over, manipulate, and display a DOM document and its nodes.

For more information about DOM, see the Document Object Model Level 2 specification at <http://www.w3.org/DOM/DOMTR>.

## XML Transformation

The ATG platform includes four servlet beans that let you create and manipulate DOM components:

- **XMLToDOM**: Parses an XML document and transforms it into a DOM document, so the document's nodes and attributes are available as objects and object properties.
- **NodeForEach**: Given a DOM node, selects nodes that match a specified pattern and iterates over the selected nodes.
- **NodeMatch**: Given a DOM node, selects the next node that matches a specified pattern.
- **XMLTransform**: Given an XML document and an XSLT or JSP template, transforms and outputs the XML document, with the formatting provided by the template.

You can also accomplish these tasks with a non ATG-specific tag library such as Apache Xtags (<http://jakarta.apache.org/taglibs/doc/xtags-doc/intro.html>).

For example, an extranet application might let clients view the status of orders that are fulfilled through a supply chain. Because the supply chain can involve other companies, you establish a standard XML format to communicate order status via documents like this:

---

```
<?xml version="1.0"?>

<widget-orders version="1">

  <order id="ID101010" account="86666">
    <order-status>
      "Shipped"
    </order-status>
  </order>

  <order id="ID939393" account="86667">
    <order-status>
      "Awaiting shipment"
    </order-status>
  </order>

  <order id="ID101011" account="86666">
    <order-status>
```





```

        "Shi pped"
      </order-status>
    </order>

  </wi dget-orders>

```

In this example, the XML document is composed of several order nodes. Each node has two attributes:

- `i d` identifies the order.
- `account` identifies the customer who placed the order.

Each order node has an `order-status` child node, which defines the order's status.

**Note:** This XML document omits an encoding declaration because it uses the default iso-8859-1 encoding. A document that uses another character encoding must include an encoding declaration. For example, if the document uses the UTF encoding, the first line must be:

```
<?xml versi on="1. 0" encodi ng="UTF-8"?>
```

For more information about character encoding, see the [ATG Programming Guide](#).

## Processing XML in a JSP

Given the previous XML document, you can use a JSP to read the XML document and retrieve data from it for display. The JSP performs these tasks:

1. Obtains the XML document and converts it into a DOM document.
2. Finds every order node with account number 86666.
3. Displays an HTML table with a row for each order node that satisfies the selection criteria as follows:

Order ID	Order Status
ID101010	Shipped
ID101011	Shipped

### Transforming XML to DOM

The JSP processes the XML document by calling the servlet bean `XMLToDom`. `XMLToDom` obtains the XML document `wi dget-orders` and converts it into a DOM document. The servlet bean's `i nput` parameter is set to `orders. xml` —the relative URL of the XML document:

```

<%-- Convert the XML document into a DOM object --%>
<dsp: dropl et name="/atg/dynamo/dropl et/xml /XMLToDOM">

```



```
<dsp: param name="input" value="orders.xml" />
<dsp: param name="validate" value="false" />

<%--if conversion succeeds--%>
<dsp: oparam name="output">
...
</dsp: oparam>

<%--if conversion fails--%>
<dsp: oparam name="failure">
...
</dsp: oparam>

</dsp: droplet>
```

---

XMLToDom has two open parameters whose bodies conditionally execute, depending on whether the transformation succeeds or fails:

- `output` specifies how to process the DOM document when transformation succeeds.
- `failure` specifies the message to display if the transformation fails.

### ***Processing DOM nodes***

If XMLToDOM succeeds, it executes the contents of its open parameter `output`. In this example, the `output` body calls the servlet bean [NodeForEach](#), which iterates over the specified DOM nodes and performs these tasks:

- Gets each order node whose `account` attribute value is 86666.
- Gets the node's `id` attribute and sets its value in the appropriate HTML table cell.




---

```

<!-- if conversion succeeds --%>
<dsp:oparam name="output">

  <!-- from first order node, select each node where account = 86666 --%>
  <dsp:droplet name="/atg/dynamo/droplet/xml/NodeForEach">
    <dsp:param name="node" param="document.firstChild"/>
    <dsp:param name="select"
      value="/widdget-orders[position()=1]/order[@account='86666']"/>

    <!-- Case where none found --%>
    <dsp:oparam name="empty">
      No orders found.
    </dsp:oparam>

    <!-- if data found, start output --%>
    <dsp:oparam name="outputStart">
      <table>
        <tr>
          <td>Order ID</td>
          <td>Order Status</td>
        </tr>
      </dsp:oparam>
      <!-- output value of order node's id attribute --%>
      <dsp:oparam name="output">
        <tr>
          <td> <dsp:valueof param="element.id.nodeValue"/> </td>

          <!-- get child node order-status, display its value --%>
          <dsp:droplet name="/atg/dynamo/droplet/xml/NodeMatch">
            <dsp:param name="node" param="element"/>
            <dsp:param name="match" value="order-status/text()"/>
            <!-- Case where match is successful --%>
            <dsp:oparam name="output">
              <td> <dsp:valueof param="matched.nodeValue"/> </td>
            </dsp:oparam>
          </dsp:droplet>
        </tr>
      <!-- end NodeForEach output --%>
    </dsp:oparam>

    <!-- close table --%>
    <dsp:oparam name="outputEnd">
      </table>
    </dsp:oparam>

    <!-- close NodeForEach droplet --%>
  </dsp:droplet>
<!-- close XMLToDOM output --%>
</dsp:oparam>

```

---

### ***Handling DOM conversion failure***

If the XML-to-DOM conversion fails, the XMLToDOM servlet bean executes its open parameter `failure`:

---

```
<%-- If conversion fails --%>
<dsp:oparam name="failure">
  Failure to parse XML document: <dsp:valueof param="input"/> <br/>
</dsp:oparam>
```

---

## **Transforming XML Documents with Templates**

The servlet bean `XMLTransform` (`/atg/dynamo/drop1et/xml/XMLTransform`) lets you use templates to transform XML documents. The template can be an XSLT template or a JSP. The document to transform, accessible as a URL, DOM document, or input stream, is specified by the servlet bean's `input` parameter.

You can specify the template to use in several ways. The ATG platform uses the template specified according to the following order of precedence:

1. The input parameter `template` that is set by the `XMLTransform` servlet bean.
2. The global `XMLTemplateMap Component`—specified by the `xmlTemplateMap` property of the component `/atg/dynamo/drop1et/xml/XMLTransform`—which defines a mapping from document type to template, or from URL to template.
3. A globally defined default template, specified by the `defaultTemplate` property of the `XMLTemplateMap Component`.
4. The template defined by the XML document itself, using an embedded `xml-stylesheet` processing instruction.

For more details on the embedded `xml-stylesheet` directive, see the specification at <http://www.w3.org/TR/xml-stylesheet>. An XSL directive must have a recognized XSL type, such as `text/xml` or `text/xsl`, to distinguish it from CSS stylesheet directives. If the source document contains multiple XSL directives, the stylesheets are concatenated before the transformation is applied. If you want an embedded stylesheet directive to take effect, the `xmlTemplateMap` must not produce a match. You can achieve this by setting the `xmlTemplateMap` property to `null`, or having an empty match table and empty default value.

In the case of a JSP template, the `XMLTransform` servlet bean passes a DOM document as a parameter local to the execution of the JSP. The name of this parameter can be controlled by the `XMLTransform` servlet bean parameter `documentParameterName`.

The transformation yields a DOM object that is bound to the `document` parameter. The name of this parameter can be controlled by the `XMLTransform` parameter `documentOutputName`.

Unless you specify otherwise with the output parameter, `XMLTransform` outputs the transformed document to the page.



## Applying a Template to an XML Document

The following example uses the `widget-orders` XML document shown earlier, and this XSLT template:

---

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="widget-orders">
  <table>
    <tr>
      <td>Order Id</td>
      <td>Order Status</td>
    </tr>
    <xsl:for-each select="order[@account='86666']">

      <xsl:sort select="@id"/>
      <tr>
        <td> <xsl:value-of select="@id"/> </td>
        <td> <xsl:value-of select="order-status"/> </td>
      </tr>

    </xsl:for-each>
  </table>

</xsl:template>
</xsl:stylesheet>
```

---

The example uses the servlet bean `XMLTransform` as follows:

- The servlet bean's `input` parameter specifies the XML document to transform—`widget-orders`—by supplying its relative URL `orders.xml`.
- The `template` parameter supplies the XSLT template shown earlier, `orders-template.xsl`.
- No output parameter is specified, so the servlet bean displays the transformed XML document in the current JSP.

---

```
<dsp:droplet name="/atg/dynamo/droplet/xml/XMLTransform">
  <dsp:param name="input" value="orders.xml"/>
  <dsp:param name="template" value="orders-template.xsl"/>

  <dsp:oparam name="failure">
    Failure to transform XML document: <dsp:valueof param="input"/>
  <br/>
  </dsp:oparam>
</dsp:droplet>
```

---



The `orders.xml` document is transformed by this template to yield the same output as shown in the XMLToDOM example shown earlier (in [Processing XML in a JSP](#)). The XSLT template performs the same selection and formatting as the servlet beans XMLToDOM, NodeForEach, and NodeMatch.

## XMLTemplateMap Component

The ATG platform includes a global component `/atg/dynamo/service/xml/XMLTemplateMap` whose mappings enable ATG servlet beans to locate templates. XMLTemplateMap has the following service Map properties:

- `urlTemplateMap` maps an absolute URL/URI of an XML file to the absolute URI/URL of a template. For example:  
`urlTemplateMap=/xml/orders.xml=/xml/orders-template.jsp`
- `docTypeTemplateMap` maps a DTD document-type name to a template.
- `defaultTemplate` specifies a template to use when no template is found for an XML file. You can set this property to either a URL or a URI in the document root.

## XML Document Cache

Transforming XML to DOM can incur considerable overhead. The ATG platform includes a separate document cache for XML documents, located at `/atg/dynamo/service/xml/DocumentCache`. This component can be used to cache the results of an XML to DOM parser. The servlet bean XMLTransform uses a document cache for XML input documents, and another for XSL stylesheet templates.

The XML document cache is similar to ATG's file cache. It uses an LRU (least-recently-used) policy to maintain the cache, determining which entries to flush based upon expiration times and timestamps.

The XML document cache has the following configurable properties. A value of -1 indicates that the value is unlimited.

Property	Default	Type	Description
<code>maximumCacheEntries</code>	-1	<code>int</code>	Maximum number of cache entries.
<code>maximumCacheSize</code>	2000000	<code>int</code>	Maximum size of the cache, in bytes
<code>maximumEntryLifetime</code>	-1	<code>long</code>	Time in milliseconds before a cache entry expires.
<code>maximumEntrySize</code>	-1	<code>int</code>	Maximum size for any single entry in the cache, in bytes.

The following read-only properties provide information about cache efficiency:



Property	Description
accessCount	Total number of cache accesses.
hitCount	Number of cache hits.
hitRatio	Number of cache hits divided by total number of cache accesses.
timeoutCount	Number of times out of date entries were invalidated.
usedCapacity	Ratio of cache entries to maximum cache entries.
usedMemory	Amount of memory in bytes used by cache entries (includes keys).

## XML Transformation and Targeting

You can use XML transformation servlet beans together with targeting servlet beans. The following example applies XMLTransform to the results obtained by TargetingForEach, as follows:

1. The targeter retrieves a result set from an XML repository, where each member of the result set is an XML document.
2. The TargetingForEach output parameter output invokes XMLTransform.
3. XMLTransform processes each element of the result set: the servlet bean transforms the XML document into a DOM document, and displays the document as specified by the template.

**Note:** This example assumes that a display template is associated with the XML documents. For more information about XML transformation and templates, see [Transforming XML Documents with Templates](#).

```
<dsp: droplet name="/atg/targeting/TargetingForEach">
  <dsp: param bean="/atg/registry/RepositoryTargeters/myXml Repository"
    name="targeter"/>
  <dsp: oparam name="outputStart">
    <b> Formatted Output </b>
    <ul>
  </dsp: oparam>

  <dsp: oparam name="output">

    <dsp: droplet name="/atg/dynamo/droplet/xml/XMLTransform">
      <dsp: param name="input" param="element.document"/>
      <dsp: oparam name="failure">
        Failure to transform XML document: <dsp: valueof param="input"/>
        <br/>
      </dsp: oparam>
    </dsp: droplet>
```



```
</dsp: oparam>
<dsp: oparam name="outputEnd">
  </ul>
</dsp: oparam>
</dsp: droplet>
```

---

### **Optimizing performance**

Transforming an XML document into a DOM object can incur considerable overhead. To obtain better performance, you can make some part of an XML document an attribute of the content item. In this way, you can access these attributes directly, rather than first transform the XML to a DOM and manipulate the DOM, or apply a template to the whole document.

For example, in the Quincy Funds demo application, the Funds list at `fundList.jsp` uses a targeter to locate and display the names and symbols of all stock funds. The fund descriptions are XML documents. Rather than transform the XML document into DOM objects and extract their fund name and symbol properties, the Funds repository treats the `fundName` and `symbol` as attributes of the repository item. Parsing an XML document into a DOM object for display occurs only when a user wants to view the entire XML fund description.

---

```
<dsp: droplet name="/atg/targeting/TargetingForEach">
  <dsp: param bean="/atg/registry/RepositoryTargeters/Funds/stockFunds"
    name="targeter"/>
  <dsp: oparam name="outputStart">
    <b>Stock Mutual Funds </b>
    <ul>
  </dsp: oparam>
  <dsp: oparam name="output">
    <dsp: a href="fund.jsp">
      <dsp: param name="ElementId" param="element.repositoryId"/>
      <dsp: valueof param="element.fundName"/></dsp: a> -
      <dsp: valueof param="element.Symbol"/><br/>
    </dsp: oparam>
  <dsp: oparam name="outputEnd">
    </ul>
  </dsp: oparam>
</dsp: droplet>
```

---

## **Passing Parameters to XSL**

You can pass parameters from the JSP environment through the XMLTransform servlet bean to an XSL stylesheet. The parameters are matched by name to top-level `<xsl:param>` elements in the stylesheet. The XMLTransform servlet bean parameter `passParam` determines which parameters are passed to the stylesheet. To pass parameters, set `passParams` parameter to one of these scopes, listed in order of precedence:





Scope	Parameters that are passed
none (default)	No parameters are passed to the XSL stylesheet.
local	Parameters defined in the XMLTransform servlet bean.
query	URL query parameters from a GET request.
post	Form parameters from a POST request.
all	Every parameter that is in scope for XMLTransform. Local parameters have highest precedence, followed by enclosing scopes within the same JSP.

If more than one parameter exists for a given name, only the first is passed. If `passParam` is set to `all`, local parameters have the lowest precedence, up through other enclosing scopes, and eventually to the request query and post parameters.

For more information about XSL param elements, see <http://www.w3.org/TR/xslt>.

### Sample stylesheet

The following stylesheet shows how to define top-level parameters for an XSL stylesheet; this stylesheet is used in later examples, to show how to set parameter values from URL query parameters and local JSP variables.

This simple stylesheet has two parameters, `p1` and `p2`. The parameters are just displayed in the page, but they can be used to build other values, or tested for conditional branches.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" encoding="iso-8859-1" media-type="text/html"/>
  <xsl:param name="p1"/>
  <xsl:param name="p2" select="'p2 default text'"/>
  <xsl:template match="/">
    <html>
      <body>
        <p>P1: <xsl:value-of select="$p1"/></p>
        <p>P2: <xsl:value-of select="$p2"/></p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

The param `p1` does not have a default value, but `p2` has a default text string. Note the use of single-quotes inside the `select` attribute. These can be omitted when the string is inside the element body:

```
<xsl:param name="p2">p2 default text</xsl:param>
```



You can invoke this stylesheet in two ways.

- [Use URL query parameters](#)
- [Use local parameters in the XMLTransform servlet bean scope](#)

### ***Use URL query parameters***

You can invoke a stylesheet through query parameters on a GET request URL. The JSP looks like this, with the `passParams` set to query scope:

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/xml/XMLTransform">
  <dsp: param name="i nput" val ue="dummy. xml "/>
  <dsp: param name="templ ate" val ue="param. xsl "/>
  <dsp: param name="passParams" val ue="query"/>
</dsp: dropl et>
```

---

An appropriate URL might look like this:

`param-query. j sp?p1=563&p2=not+defaul t+val ue`

### ***Use local parameters in the XMLTransform servlet bean scope***

You can also invoke a stylesheet by using local parameters in the scope of the XMLTransform servlet bean: the `passParams` value is set to `l ocal`, and the two parameters, `p1` and `p2`, are defined as DSP tag library `<dsp: param>` elements.

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/xml/XMLTransform">
  <dsp: param name="i nput" val ue="dummy. xml "/>
  <dsp: param name="templ ate" val ue="param. xsl "/>
  <dsp: param name="passParams" val ue="l ocal "/>
  <dsp: param name="p1" val ue="563"/>
  <dsp: param name="p2" val ue="not defaul t val ue"/>
</dsp: dropl et>
```

---

## **XSL Output Attributes**

The ATG platform supports output properties that are defined by the `<xsl : output>` element in an XSL stylesheet. The XSLT standard specification (<http://www.w3.org/TR/xslt>) and the ATG platform implementation differ in some important respects. The default values for several of the attributes are changed to match the existing ATG platform behavior, so XSL stylesheets without an `<xsl : output>` element use default values from the request-handling pipeline, instead of those defined in the XSLT standard. The most important changes are for the `encodi ng`, `method` and `medi a-type` attributes.



## Encoding

Encoding is used for character conversion on the output stream. It is written to the `charset` field of the `contentType` HTTP response header, and it is part of the XML declaration at the head of the serialized output. Encoding can be explicitly set in the stylesheet through the `encoding` attribute: `<xsl:output encoding="x" />`. If none is provided, the ATG platform checks the `contentType` property of the `DynamoHTTPServletResponse` component to see if it was already set by a JSP. If it is null, the ATG platform uses the default encoding specified for the active locale—for example, `iso-8859-1` for English.

It is generally desirable that the encoding type for the XML file match the encoding used by the JSP that embeds it, depending on the contents of both files. If the JSP renders text, the XML in it might be unreadable if it uses a different character set. In this case, you might want to set encoding in both `<xsl:output>` and the JSP that uses the servlet bean, or not set it in either place so both use the default encoding.

You can set encoding for a JSP by invoking the `setContentType` method either in the page directly so it applies to the proceeding contents in the page or in the page directive:

```
<%@ page contentType="text/html ; charset=iso-10646-1"%>
```

For more information, see *Internationalizing an ATG Web Site* in the [ATG Programming Guide](#).

## Method

The XSL output method determines the default MIME type, and also some aspects of syntax in serialized tags:

- An explicit `<xsl:output method="..." />` in the stylesheet
- Otherwise, the default is HTML
  - It is not the usual XSL default of XML. If you want XML then make it explicit in the stylesheet

The XML declaration is written by default, for output method XML. This is appropriate for a complete XML document written from a single transform. If the XML output is written as a series of nested fragments, then the XML declaration must be disabled for the embedded output using an extra flag on the `<xsl:output>` element. For example:

```
<xsl:output method="xml" omit-xml-declaration="yes" />
```

## MIME Type

The Content-type HTTP response header specifies the MIME type. Different flavors of XML can have different MIME types. For example:

- XHTML: MIME type `text/html`
- SVG: MIME type `image/svg+xml`



The recent *XML Media Types* specification ([ftp://ftp.isi.edu/in-notes/rfc3023.txt](http://ftp.isi.edu/in-notes/rfc3023.txt)) suggests more variations, such as `image/rdf+xml` for RDF, and `image/xslt+xml` for XSLT.

The ATG platform determines which MIME type to use in the following order of precedence:

1. The MIME type that is set explicitly in the stylesheet via the `media-type` attribute:  
`<xsl:output media-type="a/b"/>`.
2. The MIME type indicated in the `DynamoHttpServletResponse` (response)
3. The default used by the application server for the specific file type. The default for JSPs is `text/html`.

Because the default MIME type is not the expected XSL default mechanism based on the method or the expected `text/xml`, it is advisable to set the MIME type explicitly in the stylesheet. For information about overriding the default MIME type, see the [ATG Installation and Configuration Guide](#).

## Multiple XMLTransform Servlet Beans in a Page

If a JSP has multiple XMLTransform servlet beans and each uses a different encoding/ MIME type value pair, the pairs might not be applied to their respective servlet beans as expected.

### Nested XMLTransform Servlet Beans

In the following example, one XMLTransform servlet bean encloses another to produce a two-stage chained page transformation. Each XMLTransform servlet bean has a MIME type and encoding pair set in their respective `<XSL: output>` that is distinct from the other:

```
<dsp:droplet name="/atg/dynamo/droplet/xml/XMLTransform">
  <dsp:param name="input" value="test-page.xml"/>
  <dsp:param name="template" value="../../../doc.xml"/>
  <dsp:oparam name="output">

    <dsp:droplet name="/atg/dynamo/droplet/xml/XMLTransform">
      <dsp:param name="input" param="document"/>
      <dsp:param name="template" value="../../../xhtml-xinclude.xml"/>
      <dsp:param name="passParams" value="query"/>
      <dsp:oparam name="failure">
        <HTML>
          <HEAD>
            <TITLE>XML/XSL Error</TITLE>
          </HEAD>
          <BODY>
            <H1>XML/XSL Error processing <dsp:valueof param="input"></dsp:valueof>
            </H1>
          </BODY>
        </HTML>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```



```

        </dsp: oparam>
    </dsp: droplet>
</dsp: oparam><!-- output --%>
<dsp: oparam name="failure">
    <HTML>
        <HEAD>
            <TITLE>XML/XSL Error</TITLE>
        </HEAD>
        <BODY>
            <H1>XML/XSL Error processing <dsp: valueof param="input"></dsp: valueof>
        </H1>
        </BODY>
    </HTML>
</dsp: oparam><!-- failure --%>
</dsp: droplet>

```

---

When the servlet beans are processed, the first XMLTransform writes its encoding and MIME type settings to the HTTP response object. Although those settings are applied to the content associated with that XMLTransform, they are overwritten immediately by the settings provided by the second XMLTransform. As a result, the settings provided by the second XMLTransform servlet bean are saved to the HTTP header and applied to the content associated to both the first and second XMLTransform servlet beans.

If the second XMLTransform references a stylesheet that does not explicitly set the encoding and MIME type, the values from the first XMLTransform persist and are applied to the content associated with the first and second XMLTransform servlet beans.

## Serialized XMLTransform Servlet Beans

To insert two XML fragments in a page, you include two XMLTransform servlet beans that are adjacent and equal in the page hierarchy:

---

```

<dsp: droplet name="/atg/dynamo/droplet/xml/XMLTransform">
    <dsp: param name="input" value="test-page.xml"/>
    <dsp: param name="template" value=".../doc.xsl"/>
    <dsp: oparam name="failure">
        <HTML>
            <HEAD>
                <TITLE>XML/XSL Error</TITLE>
            </HEAD>
            <BODY>
                <H1>XML/XSL Error processing <dsp: valueof param="input"></dsp: valueof>
            </H1>
            </BODY>
        </HTML>
    </dsp: oparam>
</dsp: droplet>

<dsp: droplet name="/atg/dynamo/droplet/xml/XMLTransform">

```



```
<dsp:param name="input" param="document" />
<dsp:param name="template" value=".. /xhtml -xi nclude. xsl" />
<dsp:param name="passParams" value="query" />
<dsp:oparam name="failure">
  <HTML>
    <HEAD>
      <TITLE>XML/XSL Error</TITLE>
    </HEAD>
    <BODY>
      <H1>XML/XSL Error processing <dsp:valueof param="input"></dsp:valueof>
    </H1>
    </BODY>
  </HTML>
</dsp:oparam>
</dsp:droplet>
```

---

Both servlet beans might be provided with their own encoding/ MIME type pair from their respective stylesheets. When the servlet beans are processed, each servlet bean uses its own encoding and MIME type pair to translate its content, but the pair used by the second XMLTransform are current at the end of the page so they are written to the HTTP header. This can cause an error if the content of the first XMLTransform is incompatible with the encoding enforced by the second XMLTransform. If encoding and MIME type are not set explicitly in the JSP, the settings provided by the second XMLTransform are applied to the entire page.



## 7 Forms

Many Web applications obtain information from users by having them fill out forms. A form might need to handle user input in a variety of formats, check input for validity, handle errors, and pass input to a servlet for processing, or to a database for storage.

Through the use of special attributes, JSPs can associate HTML form tags with Nucleus components properties:

- A page can automatically display the value of a Nucleus component property as the default value of a form element.
- When a form is submitted, its values can be written to component properties. A single form can set any number of properties in Nucleus components.
- Forms can interact directly with a SQL database. Form input can be stored in a database, or used to query a database for information to display.
- When a form is submitted, its input elements can trigger actions in the application.

This chapter shows how to use forms in JSPs. It covers the following topics:

- [Form Basics](#)
- [DSP Tags and HTML Conventions](#)
- [Embedding Pages in Forms](#)
- [Setting Property Values in Forms](#)
- [Submitting Forms](#)

### Form Basics

An ATG form is defined by the `dsp: form` tag, which typically encloses DSP tags that specify form elements, such as `dsp: input` that provide direct access to Nucleus component properties. For example, the following JSP embeds a form that accesses the properties in the `Student_01` component:



```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<dsp: page>

<html >
<head><title>Form Entry</title></head>
<body><h1>Form Entry</h1>

    <dsp: form action="/testPages/showStudentProperties.jsp" method="POST">
        <p>Name: <dsp: input bean="/samples/Student_01.name" type="text" />
        <p>Age: <dsp: input bean="/samples/Student_01.age" type="text" value="30" />
        <p><dsp: input type="submit" bean="/samples/Student_01.submit" />
            value="Click to submit" />
    </dsp: form>
</body>
</html >

</dsp: page>
```

In this form, two input tags are associated with the name and age properties of the Nucleus component `/samples/Student_01`. When this page is displayed, the field for the name property is filled in with its current value; the field for the age property specifies the `value` attribute, so that value is displayed instead of the current property value.

The form user can enter new values in these fields and submit the form. On submission, the values entered in the form replace the current values of the specified properties, and `showStudentProperties.jsp` is invoked, as specified by the form tag's `action` attribute.

**Note:** The `dsp: form` tag requires you to specify the `action` attribute.

## DSP Tags and HTML Conventions

The DSP library provides form processing tags that are similar to their HTML equivalents. In general, these tags support standard HTML attributes. On page compilation, these attributes are simply passed through to the HTML page. For example, the text input tag has a `size` attribute that specifies the width of the text input field. If you specify `size=25` for a text input tag, after page compilation the resulting field is 25 characters wide.

### **Name attribute**

Unlike standard HTML, which requires the name attribute for most input tags; the name attribute is optional for DSP form element tags:

- If an input tag omits the name attribute, the ATG platform assigns one when it compiles the page.





- If an input tag specifies the name attribute, the ATG platform uses it

**Note:** If name attributes are used, they must be unique for each property set in the form.

### **Attribute values**

Unlike HTML forms, ATG forms require an explicit value for all attributes. For example, an HTML input tag that creates a checkbox can specify the checked attribute without supplying a value, in order to specify that the checkbox is selected by default. For example, the following HTML specifies to check the primary checkbox:

---

```

Primary residence:
<input type="checkbox" name="re" value="primary" checked /><br />
Vacation home:
<input type="checkbox" name="re" value="vacation" /><br />
Other:
<input type="checkbox" name="re" value="other" />

```

---

This construction is invalid in JSP tags, where all attributes must be explicitly set to a value. Thus, the corresponding dsp: input tag must use this form:

---

```

Primary residence
<dsp:input type="checkbox" name="re" value="primary" checked="true" />
...

```

---

## Embedding Pages in Forms

The example in [Form Basics](#) shows a form that is defined in a single JSP. You might want to set the form's dsp: form tags in one JSP and define the form contents in another,, as in this example:

---

```

<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<dsp:page>
<html>
<head><title>Form Entry</title></head>
<body><h1>Form Entry</h1>

<dsp:form action="/testPages/servbeantest.jsp" method="POST">
  <dsp:include page="formbody.jsp"/>
</dsp:form>
</body>
</html>
</dsp:page>

```

---



The embedded JSP formbody.jsp contains the form's input elements:

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<dsp:page>

    <p>Name: <dsp:input bean="/samples/Student_01.name" type="text"/>
    <p>Age: <dsp:input bean="/samples/Student_01.age" type="text" value="30"/>
    <p><dsp:input type="submit" value="Click to submit"
        bean="/samples/Student_01.submit"/>

</dsp:page>
```

**Note:** An included page can contain element tags such as `dsp:input`, `dsp:textarea`, and `dsp:select` only if the page is a complete JSP—that is, it imports the required tag libraries and contains the `dsp:page` tag. Form elements cannot be embedded in a JSP fragment (.jspf) file).

By separating a form's `dsp:form` tag from its actual contents, you can customize the form for different user types. For example, the `dsp:form` tag might embed a [Switch](#) servlet bean that renders different forms for different genders—for example, specifies to include `formbodyFemale.jsp` for female users and `formbodyMale.jsp` for male users.

## Setting Property Values in Forms

HTML forms can set the values of Nucleus component properties by using the bean attribute to associate a form element with a Nucleus component property. This section describes DSP tags that extend HTML form elements to set property values.

A single form can set the properties of various Nucleus components. In general, only one input tag in a form can be associated with a given property or with a given element of an indexed or array property. One exception applies: multiple submit inputs on a form can be associated with the same property (see [Using Submit controls to Set Property Values](#)).

The following sections describe how to set values for the following property types:

- [Scalar \(non-array\) properties](#)
- [Array properties](#)
- [Map properties](#)

This section also shows how to [set property values via hidden inputs](#).



## Scalar (Non-Array) Properties

Several different form controls can set the values of scalar properties, depending on the data type of the property, and whether you want to provide the user with a predetermined set of choices.

### Checkboxes

Using a checkbox, a user can set the value of a boolean property. For example, the Student\_01 component might have a property called emailOK that indicates whether your site can send promotional email. Users can indicate their preference on a registration form that contains this input tag:

```
<dsp:input
  bean="/samples/Student_01.emailOK" type="checkbox"/>Send email
```

The current value of the property emailOK sets the checkbox's initial display: checked (true) or unchecked (false). To override the initial setting, set the checked attribute to true. For example, you might want the checkbox to be checked when the page displays, regardless of the property's current setting. The input tag looks like this:

```
<dsp:input
  bean="/samples/Student_01.emailOK" type="checkbox" checked="true"/>Send email
```

You can also use the default attribute to specify a value for the input element to display, in the event the property has no value:

```
<dsp:input
  bean="/samples/Student_01.emailOK" type="checkbox" default="true"/>Send email
```

### Radio Buttons

To create a set of radio buttons, set a dsp:input tag's type attribute to radio. The following example uses a group of radio buttons to present users with a set of choices, for setting the favoriteSport property:

---

```
<p>Choose your favorite outdoor sport:
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="swimming"/>Swimming
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="biking"/>Biking
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="hiking"/>Hiking
...
```

---

In contrast to standard HTML, the name attribute is optional; grouping is based on sharing the same bean attribute—in this example, Student\_01.favoriteSport.

Unless you specify otherwise, the page is initially displayed with the current value of favoriteSport, assuming it matches one of the three options specified here. You can override this behavior by using the checked attribute. The previous example can be modified so the hiking radio button is initially selected no matter how the favoriteSport property is set:



---

```
<p>Choose your favorite outdoor sport:
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="swimming" checked="false" />Swimming
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="biking" checked="false" />Biking
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="hiking" checked="true" />Hiking
...
```

---

Within a selection group, only one input tag can set its checked attribute to true; all others must set their checked attribute to false.

The default attribute specifies a value to use if the component property has no value. To specify a value as the default, set the default attribute of its input tag to true, and set default to false for all other input tags:

---

```
<p>Choose your favorite outdoor sport:
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="swimming" default="true" />Swimming
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="biking" default="false" />Biking
<dsp:input bean="/samples/Student_01.favoriteSport"
  type="radio" value="hiking" default="false" />Hiking
...
```

---

### Drop-Down Lists

To create a set of choices in a drop-down list, use a dsp:select tag. For example:

---

```
<dsp:select bean="/samples/Student_01.favoriteSport">
  <dsp:option value="swimming">Swimming</dsp:option>
  <dsp:option value="biking">Biking</dsp:option>
  <dsp:option value="hiking">Hiking</dsp:option>
  ...
</dsp:select>
```

---

Unless you specify otherwise, the control displays the bean's current value as the initial selection.. To override this behavior, set the dsp:select tag's nodefault attribute together with the dsp:option tag's selected attribute. For example, the following tags specify Hiking as the initial selection regardless of the current value in favoriteSport:

---

```
<dsp:select bean="/samples/Student_01.favoriteSport" nodefault="true">
  <dsp:option selected="false" value="swimming">Swimming</dsp:option>
  <dsp:option selected="false" value="biking">Biking</dsp:option>
  <dsp:option selected="true" value="hiking">Hiking</dsp:option>
```

---



```
...
</dsp: select>
```

Only one `dsp: option` tag can set `selected` to true; the others must set it to false.

**Note:** Use the `nodefault` attribute only to ignore a property's current value. If a form gathers data for a property that has no value—for example, for a new user profile—you can omit the `nodefault` attribute and set the desired selection through the `selected` attribute only.

`dsp: select` can also be used to create a [multiple-selection list box](#), through its `multiple` attribute.

### Text Entry Fields

You can create text entry fields with two DSP library tags:

- `dsp: input` creates a single-line text-entry field if its `type` attribute is set to `text`.
- `dsp: textarea` creates a multi-line area for longer text entries.

`dsp: input` creates a text entry field that is primarily used to enter text values; it can also be used for other types of data. For example, you can associate a text field with a property whose data type is Integer, such as `Student_01.age`. On form submission, the ATG platform converts the value from a String to the appropriate data type before it writes the value to the component property. If the conversion fails, the property remains unchanged.

A text input field displays the component property's current value unless the `dsp: input` tag also includes the `value` attribute. The `value` attribute overrides the property's current value. In the following example, the `dsp: input` tag for `Student_01.name` specifies to display the property's current value, while the tag for `Student_01.age` includes a `value` attribute that specifies the field to show a value of 30:

```
<p>Name: <dsp: input bean="/samples/Student_01.name" type="text" />
<p>Age: <dsp: input bean="/samples/Student_01.age" type="text" value="30" />
```

The user can change the values that appear in the fields, or leave them unchanged. When the user submits the form, the field entries are written to the specified properties.

`dsp: textarea` creates a multi-line field where users can enter longer text. For example:

```
<dsp: textarea bean="/samples/Student_01.evaluation" ></dsp: textarea>
```

`dsp: textarea` accepts standard HTML attributes such as `rows` and `cols`, which set the size of the text area on the page.

The text area initially contains the property's current value unless you supply the `default` attribute to override that value. The default text can be a String constant, the value of another property, or a page parameter. For example:

```
<dsp: getvalueof var="standardEval" bean="ReportCard.summary" ></dsp: getvalueof>
<dsp: textarea bean="/samples/Student_01.evaluation" default="{standardEval}" />
```

Alternatively:



```
<dsp: textarea  
  bean="/samples/Student_01. evaluation" default t="--Enter your evaluation here--"/>
```

You can set a textarea's value between its open and close tags. For example:

```
<dsp: textarea bean="/samples/Student_01. evaluation">  
--Enter your evaluation here--  
</dsp: textarea>
```

If a dsp: textarea tag uses both methods, the default t-specified value has precedence.

**Note:** To display the current property value, make sure no white space or other characters appear between the open and close tags <dsp: textarea> </dsp: textarea>; otherwise, those characters are interpreted as the default and override the property value.

## Array Properties

To set the values in an array property, the user needs a control that allows selection of multiple values. When the user submits the form, the selected values are written to array elements.

A form can set values in an array property through a grouped set of checkboxes or a multiple-selection list box. Both controls are functionally equivalent, each possessing different layout advantages: a drop-down list typically requires less space because it uses scrollbars for long lists; while checkboxes make all choices visible.

### Grouped Checkboxes

You can use a group of checkbox input tags to set an array of values. The following example creates a group of checkboxes that lists hobbies:

---

```
<dsp: input  
  bean="/samples/Student_01. hobbies" type="checkbox" value="swimming"/>Swimming  
<dsp: input  
  bean="/samples/Student_01. hobbies" type="checkbox" value="biking"/>Biking  
<dsp: input  
  bean="/samples/Student_01. hobbies" type="checkbox" value="climbing"/>Climbing  
<dsp: input  
  bean="/samples/Student_01. hobbies" type="checkbox" value="photo"/>Photography  
<dsp: input  
  bean="/samples/Student_01. hobbies" type="checkbox" value="fencing"/>Fencing
```

---

In this example, each checkbox is associated with the hobbies property, which is an array of Strings. When this page is displayed, any checkbox whose value attribute matches a value in the hobbies property appears as preselected. Each checkbox input tag can also set the checked attribute to true, in order to override current property settings.

Users can check or uncheck any combination of the checkboxes. On form submission, the values of all selected checkboxes are written to the array's elements.



Given the previous example, if the array property `hobbies` has two elements set to `climbing` and `fencing`, the climbing and fencing checkboxes are initially checked, while the others are unchecked:

- ☐ Swimming
- ☐ Biking
- ☒ Climbing
- ☒ Photography
- ☐ Fencing

The user can change these selections—for example, deselect climbing and photography, and select swimming, biking, and fencing checkboxes:

- ☒ Swimming
- ☒ Biking
- ☐ Climbing
- ☐ Photography
- ☒ Fencing

Given these selections, on form submission the `hobbies` array property is set to three elements that contain the values `swimming`, `biking`, and `fencing`.

You can group checkboxes with the `name` attribute only if all tags in the group supply the same `name` setting. You can save a value to the array property if no checkboxes are selected by setting the `default` attribute to `true` for one checkbox in the group. Only one checkbox tag in the group can set its `default` attribute to `true`; all other checkbox tags must set their `default` attribute to `false`.

### Multiple-Selection List Box

`dsp:select` creates a multiple-selection list box if its `multiple` attribute is set:

---

```
<dsp:select multiple="true" bean="/samples/Student_01.hobbies">
  <dsp:option value="swimming">Swimming</dsp:option>
  <dsp:option value="biking">Biking</dsp:option>
  <dsp:option value="climbing">Climbing</dsp:option>
  <dsp:option value="photo">Photography</dsp:option>
  <dsp:option value="fencing">Fencing</dsp:option>
</dsp:select>
```

---

In this example, each option is associated with the `hobbies` property, which is an array of `Strings`. When this page is displayed, any option whose `value` attribute matches a value in the `hobbies` array is initially selected. You can override these initial selections and determine which options to select on initial display by qualifying each `dsp:option` tag with the `selected` attribute, set to `true`.



The user can select any combination of options. On form submission, the values of all selected options are used to set the new elements of the array.

For example, the `hobbies` array property might have two elements, `climbing` and `fencing`; in that case, the climbing and fencing options are shown as selected when the form displays. If the user selects the swimming, biking, and fencing, on form submission the `hobbies` property is set to an array of three elements with those values.

## Map Properties

ATG's built-in `map` converter facilitates the entry of map property values on forms. You can use it with `dsp:input` as follows:

```
<dsp:input bean="FormHandler.map-property" converter="map" value="key=value" />
```

**Note:** The map converter is not required by the `RepositoryFormHandler`, which can handle Map data in its own way (see [Managing Map Properties](#) in the `RepositoryFormHandler` chapter).

For example:

---

```
<dsp:input bean="FormHandler.address" converter="map" value="street=" />
<dsp:input bean="FormHandler.address" converter="map" value="city=" />
<dsp:input bean="FormHandler.address" converter="map" value="state=" />
<dsp:input bean="FormHandler.address" converter="map" value="zip=" />
```

---

The `converter` attribute is set to the `map` tag converter, which ensures that on form submission, the Map settings are applied correctly to the target Map property.

You can also use a hidden input control to set a Map property with multiple key/value pairs, as in the following example:

---

Please verify the following address: <br/><br/>

```
Street: &nbsp;   <c:out value="{emp.street}"/> <br/>
City: : &nbsp;   <c:out value="{emp.city}"/> <br/>
State: : &nbsp;   <c:out value="{emp.state}"/> <br/>
Zip code: : &nbsp;   <c:out value="{emp.zip}"/> <br/>
```

```
<dsp:input type="hidden"
  bean="FormHandler.address" converter="map"
  value="street={emp.street}, city={emp.city}, state={emp.state}, zip={emp.zip}"
/>
```

---





## Set Property Values via Hidden Inputs

A component might contain one or more properties whose values are not visible on the form and which are set programmatically, often computed from other user-supplied values. A form can set these properties through hidden input tags. On form submission, these input tags write their values to the properties associated with them.

For example, the `Student_01.LastSubmit` property stores the timestamp of the last form submission. You can set this property from the ATG platform component `/atg/dynamo/service/CurrentDate` through a hidden form input, as follows:

```
<dsp:input type="hidden"
  bean="/samples/Student_01.LastSubmit"
  beanvalue="/atg/dynamo/service/CurrentDate.secondAsDate" />
```

Hidden input tags are commonly used to save and restore values of the properties of a request-scoped component from one request to the next. This lets you retain those values across multiple requests—typically, for the duration of the current session.

## Submitting Forms

On form submission—typically triggered when the user clicks the form's submit control—ATG sets the property values as specified in the form, and displays the page specified in the `action` attribute of the form tag.

### Submit Input Tags

The input tag for a form submit control can have one of these two formats:

```
<dsp:input type="submit"
  [bean="prop-spec" [submitvalue="value"] value="value"] />

dsp:input type="image" {src|image}=path
  [bean="prop-spec" submitvalue="value"] />
```

### Submit Control Attributes

You can set the following attributes for a submit control:

Attribute	Description
type	Determines whether the submit control displays as a button or image. If set to <code>submit</code> , the tag specifies to display a submit button; if set to <code>image</code> , it defines a submit control that uses the image specified by the <code>src</code> or <code>page</code> attribute.
bean	Specifies a component property to set when the user clicks the submit control.

val ue	Specifies the text to display on the submit button, if type is set to submit. If the attribute submit value is omitted, the value is also used to set the bean-specified property. This attribute is not valid if type is set to image.
submit value	Specifies a value to set on the bean-specified value when the user clicks on this submit control. If type is set to submit, this attribute overrides the value-specified value, which is otherwise used to set the property.

### Using Submit Controls to Set Property Values

A submit control can set the bean-specified property through the standard value attribute. The control can override its own value setting through the non-standard submit value attribute. Used together, value specifies only the button label text (if the input type is set to submit), and submit value sets the property value. For example, the following tag creates a submit button with the label Click Here, and specifies to set Student\_01.age to 34:

```
<dsp:input type="submit" bean="/samples/Student_01.age" submit value="34"
value="Click Here"/>
```

You typically set property values through data entry inputs and hidden inputs. Occasionally, however, you might want to associate property settings with submit controls. For example, a form might have multiple submit buttons, where each sets the same property to a different value. The following input tags define two submit buttons, each specifying the Boolean property Student\_01.emailOK:

```
<dsp:input type="submit" bean="/samples/Student_01.emailOK"
value="Send email" submit value="true" name="OK"/>
<dsp:input type="submit" bean="/samples/Student_01.emailOK"
value="Do not send email" submit value="false" name="notOK"/>
```

The first submit button sets the property to true; the second to false. Because the two input tags specify the same property, each requires a unique name attribute.

**Note:** In general, two controls cannot be associated with the same property. In the previous example, the submit buttons can be associated with the same property because only one of them can be executed for a given form submission.

### Using Images as Submit Controls

An image can act as a submit control. When the user clicks the image, the image input submits the form. The src or image attribute specifies the pathname of the image file. As with a submit button, the bean attribute specifies a property to set on form submission, and submit value specifies the value. For example:

```
<dsp:input bean="/samples/Student_01.emailOK" type="image"
src="clickpic.gif" submit value="true"/>
```

If you omit the submit value attribute, the image pixel coordinates are submitted as the value.



## Order of Tag Processing

On form submission, DAF sets all properties from form input tags in a predetermined order. The ATG platform uses its own algorithm to determine the default order for processing form tags. You can also explicitly set the sequence in which input tags are processed.

### *Default order*

The ATG platform assigns a default priority to all form controls, which determines the order in which they are processed on form submission. Every form element is assigned a positive or negative integer, or zero, as its priority value:

- Input elements of type `submit` and `image` are assigned priority -10. They are assigned a low priority in order to ensure that the `set` and `handle` methods associated with the element properties execute only after all other inputs are processed.
- All other form elements are assigned priority 0.

On form submission, elements are processed in descending order of priority. Elements with the same priority are processed in the order they are initially rendered—typically, the same order of their appearance on the page. However, use of conditional logic such as servlet beans can cause the order of rendering elements to differ from their order of appearance on the page.

### *Manually set order*

You can explicitly set the priority of specific input elements by setting their `priority` attribute. For example, you can ensure that a given property is always processed first by giving it a high priority setting as follows:

```
<dsp:input type="text" priority="10" bean="/samples/Student_01.name"/>
```

Any elements whose priority is not explicitly set use the default priority that the ATG platform assigns to them, as described earlier. Priority values must be literal integers; they cannot be represented by an expression.

## Synchronizing Form Submissions

When you use a form to set property values, take into account the scope of the components whose properties are being set. As a general rule, it is a good idea to use request-scoped components; this ensures that only one request at a time can set their properties.

If a form sets properties of components that are not request-scoped, you must ensure that multiple requests do not access the component at the same time. This is especially important for globally-scoped components, which are highly vulnerable to multiple simultaneous requests from different sessions; with a session-scoped component, multiple simultaneous requests occur only if the user submits the form twice in very rapid succession.

To ensure that multiple requests do not access the same component simultaneously, use the `synchronized` attribute in your form tag. With this attribute, the ATG platform locks the specified component before setting any properties, and releases the lock only after form submission is complete. Other form submissions can set the component's properties only after the lock is released.

For example, the following `dsp: form` tag locks the `Student_01` component during form submission:

```
<dsp: form
  method="post" action="servbeantest.jsp" synchronized="/samples/Student_01"/>
```

## Preventing Cross-Site Attacks

Cross-site scripting attacks take advantage of a vulnerability that enables a malicious site to use your browser to submit form requests to another site, such as an ATG-based site. In order to protect forms from cross-site attacks, you can enable form submissions to automatically supply the request parameter `_dynSessConf`, which identifies the current session through a randomly-generated long number. On submission of a form or activation of a property-setting [dsp: a](#) tag, the request-handling pipeline validates `_dynSessConf` against its session confirmation identifier. If it detects a mismatch or missing number, it can block form processing and return an error.

You can control session confirmation for individual requests by setting the attribute `requestSessionConfirmation` to `true` or `false` on the applicable [dsp: form](#) or [dsp: a](#) tag. If this attribute is set to `false`, the `_dynSessConf` parameter is not included in the HTTP request; and the request-handling pipeline skips validation of this request's session confirmation number. You can also set session confirmation globally; for more information, refer to *Appendix F: Servlets in a Request Handling Pipeline*, in the [ATG Programming Guide](#).



## 8 Form Handlers

The forms described in the previous chapter are suitable for simple applications. More often, your applications are likely to require more sophisticated tools to process forms.

For more complex form processing, the ATG platform provides form handlers that can perform these tasks:

- Validate data before it is submitted.
- Detect missing information and display appropriate messages to the user.
- Direct users to different pages depending on form submission results.
- Read and write database or repository data.

### Form Handler Classes

Form handlers are components that you typically build from one of several ATG form handler classes. All provided form handler classes are subclasses of `atg.repository.servlet.GenericFormHandler`, and inherit its properties and methods:

Form handler	Purpose	For more information...
<code>SimpleSQLFormHandler</code>	Works with form data that is stored in a SQL database.	<a href="#">Appendix C: SimpleSQLFormHandler</a>
<code>RepositoryFormHandler</code>	Saves repository data to a database.	<a href="#">RepositoryFormHandler</a>
<code>ProfileFormHandler</code>	Connects forms with user profiles stored in a profile repository.	<a href="#">User Profile Forms</a>
<code>SearchFormHandler</code>	Specifies properties available to a search engine.	<a href="#">Search Forms</a>



## Resetting a Form

Form handler components all support a cancel operation, which is invoked by a form's Cancel button. The input tag for a Cancel button uses the following format:

```
<dsp:input type="submit" value="button-label" bean="handler-component.cancel"/>
```

By default, the cancel operation redirects to the page specified in the form handler's cancel URL property. A form handler typically provides its own implementation of this operation in `handleCancel()`, often using it to empty the form fields when the user clicks Cancel. For more information, see the description of `handleCancel()` in the [ATG Programming Guide](#).

### Redirecting to another page

The form handler property cancel URL specifies the page where users are redirected after they click Cancel. You can edit this property directly; however, it is better practice for a JSP to set cancel URL through a hidden input tag attribute to a value that is appropriate to the current context. The URL that you specify in cancel URL is resolved relative to the page designated in the form's `action` attribute, so setting cancel URL and `action` on the same page can help avoid URL errors. For example:

```
<dsp:input type="hidden" bean="MyFormHandler.cancelURL" value=".. /index.jsp"/>
```

## Form Error Handling

A Web application must identify and gracefully handle errors that might occur on form submission. For example, a user enters an alphabetic character in a field where an integer is required. If the property set by this field is of type `int`, on form submission an exception is thrown when the application tries to set this property from the field value.

If the form uses a form handler of class `GenericFormHandler` or one of its subclasses, exceptions that occur during form processing are saved to one of these form handler component properties:

Property	Purpose
<code>formError</code>	Boolean that is set to true if any errors occur during form processing.
<code>formExceptions</code>	A vector of the exceptions that occur during form processing.
<code>propertyExceptions</code>	A read-only property that returns a Dictionary of subproperties, one for each property set by the form. For each property that generates an exception, a corresponding subproperty in the <code>propertyExceptions</code> Dictionary contains that exception. For each property that does not generate an exception, the corresponding subproperty in the <code>propertyExceptions</code> Dictionary is unset.



## Displaying Form Exceptions

If a form submission generates exceptions, you might want to display error messages that ask the user to correct those errors and resubmit the form. Form submission exceptions are of class `atg.droplet.DropletException`, or its subclass `atg.droplet.DropletFormException`. Both classes support the `errorCode` property, which indicates the type of error that occurred. The `DropletFormException` class has a `propertyName` property that indicates the form handler property associated with the exception.

To capture errors and display them to users, pass the value of the form handler component's `formExceptions` property to the ATG servlet bean [ErrorMessageForEach](#). This servlet bean displays messages that are keyed to the `errorCode` and `propertyName` properties.

## Detecting Errors

In the following example, the Switch servlet bean uses the form handler's `formError` property to determine whether any errors occurred when processing the form. If this property is true, the form handler's `formExceptions` property is passed to the servlet bean `ErrorMessageForEach`, which displays the message associated with each exception in the `formExceptions` vector.

---

```
<dsp: droplet name="Switch">
<dsp: param bean="MyFormHandler.formError" name="value"/>
<dsp: oparam name="true">
  <dsp: droplet name="ErrorMessageForEach">
    <dsp: param bean="MyFormHandler.formExceptions" name="exceptions"/>
    <dsp: oparam name="output">
      <dsp: valueof param="message"/>
    </dsp: oparam>
  </dsp: droplet>
</dsp: oparam>
</dsp: droplet>
```

---

**Note:** If your form handler is session-scoped, clear the `formExceptions` property after you display errors to the user.

## Alerting Users to Errors

Use the `propertyExceptions` property to determine which form handler properties have exceptions associated with them, and to display a message near each field where an exception occurred. In the following example, the form includes a field where users are required to enter an email address. If no email address is entered, an exception is thrown and the error code for that exception is `missingRequiredValue`—the error code commonly used by form handlers when a required field is empty. The Switch bean renders the `missingRequiredValue` open parameter, which displays an error message immediately below the field.

---

```
<p>Email address:
<dsp: input bean="MyFormHandler.value.email" size="32" type="text"
```

```
required="true"/>
<dsp: droplet name="Switch">
  <dsp: param bean="MyFormHandler.propertyExceptions.email.errorCode"
    name="value"/>
  <dsp: oparam name="missingRequiredValue">
    <br/>You must provide an email address.
  </dsp: oparam>
</dsp: droplet>
```

---

## Redirecting on Form Submission

ATG form handlers provide a number of submission operations that are specific to the form handler type—for example, the [RepositoryFormHandler](#) supports create, update, and delete operations. Each submission operation typically supports a pair of navigation properties that can be set to redirect the user to another URL on the success or failure of the operation. If set, these navigation properties override the form's `action` attribute. Navigation property names generally have the following format:

*operation*SuccessURL  
*operation*ErrorURL

Thus, the `RepositoryFormHandler`'s update operation supports two navigation properties: `updateSuccessURL` and `updateErrorURL`.

Navigation properties are set through hidden input tags. For example, the following hidden input tag redirects users to `updateStockFailure.jsp` in the event that an update operation fails:

---

```
<dsp: input type="hidden"
bean="atg/dynamo/droplet/MyRepositoryFormHandler.updateErrorURL"
value="updateStockFailure.jsp"/>
```

---

For more information about navigation properties, see later chapters on specific form handlers.

**Caution:** If you set an `errorURL` property to the name of the originating form and the corresponding operation fails, on redirect all fields on that form are emptied of their previous values. In order to avoid this, the `errorURL` must explicitly pass back the field values you wish to retain.





## 9 Search Forms

Search forms can help find products that satisfy a set of criteria. For example, a search form might help find recipes that contain substring such as Cayenne, or recipes that are tagged with a keyword such as Cajun.

You build search forms with the form handler class `atg.repository.servlet.SearchFormHandler`. This handler uses the search criteria to return items from one or more repositories, and one or more item types from a given repository. `SearchFormHandler` is a subclass of `atg.repository.servlet.GenericFormHandler`, so it inherits properties and handler methods common to all form handlers.

**Note:** Some ATG applications might provide subclasses of `SearchFormHandler`.

### ***Search types***

The `SearchFormHandler` supports several search methods, specified by setting one or more of the following Boolean properties:

Property	Search type
<code>doKeywordSearch</code>	<a href="#">Keyword search</a> : Uses user-supplied data to query the contents of string properties that store keywords
<code>doTextSearch</code>	<a href="#">Text search</a> : Takes a user-entered search string and performs text pattern matching on one or more text properties
<code>doHierarchicalSearch</code>	<a href="#">Hierarchical search</a> : Returns all parent and descendant items of the specified item.
<code>doAdvancedSearch</code>	<a href="#">Advanced search</a> : Provides search options for each property specified in the form handler's <code>advancedSearchPropertyNames</code> property.

Combined search types are discussed under [Combination Search](#).



## Creating SearchFormHandler Components

The ATG distribution provides no components of the `SearchFormHandler` class. Because there are many possible combinations of search options, you should create a unique search form handler component for each search that you use.

In order to provide a search form handler for your Web site, complete two tasks:

1. Create a component in the ACC whose `Class` and `Scope` properties are defined as follows:
  - `Class`: `atg.repository.servlet.SearchFormHandler`
  - `Scope`: `session`
2. Configure the [basic SearchFormHandler properties](#) that are common to all search types, then configure the properties that are specific to the desired search type.

## Basic SearchFormHandler Properties

All search form handler components generally require you to set the following properties:

Property	Description
<code>allowEmptySearch</code>	Determines whether a search proceeds or throws an error when no search criteria are specified. The default value is true.
<code>allowRefine</code>	Determines whether search criteria include previously entered criteria. See <a href="#">Clearing Search Query Data</a> for details. The default value is false.
<code>enableCountQuery</code>	Determines whether the ATG platform numbers the items located by a search. See <a href="#">Search Results Properties</a> for an explanation of related properties. The default value is false.
<code>errorURL</code>	URL that opens when a search fails. Set this property to the URL of a page that describes the error and provides tools for starting another search.
<code>itemTypes</code>	An array of item types to include in the search. Each item type must be defined in all repositories specified in the <code>repositories</code> property.
<code>previouslySubmitted</code>	Tracks whether a query that uses this <code>SearchFormHandler</code> occurred during the current session. When a search begins, this property is set to true.
<code>repositories</code>	A comma-delimited list of repositories to include in the search.
<code>successURL</code>	URL that opens when a search operation succeeds. This property should point to the URL for the first search results page.



## Multisite SearchFormHandler Properties

The SearchFormHandler class and subclasses such as atg.commerce.catalog.custom.CatalogSearchFormHandler define several properties that let you constrain search results to repository items that belong to one or more sites:

Property	Description
siteIds	Constrains the search to the list of site IDs, returns only items that belong to the specified sites.
siteScope	<p>Constrains the scope of searched sites, set to one of the following values:</p> <p>current (default) Returns matching repository items from the current site only.</p> <p>all Returns all matching repository items and does not filter based on site context. The result set can include items that do not belong to any site.</p> <p>any Returns matching repository items that belong to any site.</p> <p>none Returns matching repository items that do not belong to any site.</p> <p><i>shareable-type-ID</i> Returns matching repository items that belong to any sites that are in a sharing group with the current site, as defined by the ShareableType component ID. For example, you can return items that belong to sites that share a shopping cart with the current site.</p>
includeInactiveSites	A Boolean property, specifies whether to include active sites in the search. By default, only active sites are returned.
includeDisabledSites	A Boolean property, specifies whether to include disabled sites in the search. Set this property and includeDisabledSites to true in order to allow the search to return sites that are both inactive and disabled. By default, only enabled sites are returned.

The siteIds property has precedence over siteScope; if the first is set, the second is ignored. The SearchFormHandler method generateResultSet() adds multisite constraints to the generated query, as set by these properties.

See the [ATG Multisite Administration Guide](#) for examples of search forms that use multisite criteria to filter results.



## Keyword Search

A keyword search uses user-supplied data to query the contents of string properties that store keywords. For example, a keyword search might query items in a Recipes repository and return all items where the property Equipment contains the keyword "blender."

Keyword strings that contain embedded spaces must be enclosed by single or double quotes—for example, "summer squash".

### Keyword Search Properties

To enable keyword searches in the form handler, set the following properties:

- `doKeywordSearch` is set to `true` to enable keyword searching.
- `keywordSearchPropertyNamees` specifies one or more single or multi-valued properties to include in searches. If this property is empty, the form handler searches all string properties of each repository item, except enumerated or non-queryable properties.

This property is required to enable keyword searches on a content repository

You configure keyword searching with these properties:

#### Required Keyword Search Properties

Property	Description
<code>dokeywordSearch</code>	If true, enables this component for keyword searches.

#### Optional Keyword Search Properties

Property	Description
<code>excl udedStri ngPropertyNames</code>	<p>An array of string properties to exclude from a keyword search, specified in this format:</p> <p><i>i tem- type. property- name</i></p> <p>For example, the following setting specifies to exclude properties <code>someDescri ption</code> and <code>someStri ng</code> in items of type <code>descri ptor1</code> from keyword searches:</p> <pre>excl udedStri ngPropertyNames=\ descri ptor1. someDescri ption, \ descri ptor1. someStri ng</pre>



Property	Description
keywordInput	One or more keyword values, typically supplied by the search form user. Multiple keyword values are joined through the <a href="#">logical operators</a> AND, NOT, and OR.
keywordSearchPropertyNames	<p>The properties that are scanned during a keyword search, specified in this format:</p> <p><i>item-type. property-name</i></p> <p>If this property is empty, the keyword search looks at all string properties in the item types that are specified by the <code>itemTypes</code> property.</p> <p>If you specify multiple properties, the keyword search generates a query for each property, then ORs these queries together. If any query matches an item, the item is returned by the search operation.</p>
keywords	List of keywords that are always used in a search. If no keywords are specified for this property, the keywords must be supplied by the user.
toLowerCaseKeywords	If set to true, converts user-entered keyword to lowercase. If <code>toLowerCaseKeywords</code> and <code>toUpperCaseKeywords</code> properties are both set to true, <code>toLowerCaseKeywords</code> takes precedence. The default value is false.
toUpperCaseKeywords	If set to true, converts user-entered keyword to uppercase. The default value is false.

## Logical Operators

Keyword searches support complex search expressions that use the following logical operators, listed in order of precedence:

- NOT / !
- AND / &
- OR / |

For example, a user might enter these search criteria:

(summer AND Mexican OR southwestern) AND NOT "fresh cilantro"

Given these criteria, the `SearchFormHandler` retrieves repository data as follows:

1. Excludes all items that contain the keyword fresh cilantro.
2. Of the remaining items:



- Retrieves those that have both of these values: summer and Mexi can.
- Includes items that have the value southwestern.

## Single-Value Property Searches

If a property specified in `keywordSearchPropertyNames` is single-valued—for example, of type `String`—the keyword search algorithm uses the query `QueryBuilder.CONTAINS` to examine each target value and see if it appears anywhere within the property's current value. For example, given a `keywordString` property of type `String`, a keyword search for the values red, green, and blue generates this query:

```
keywordString CONTAINS "red"  
OR keywordString CONTAINS "green"  
OR keywordString CONTAINS "blue"
```

Because `CONTAINS` performs a substring match, this query returns true for an item whose `keywordString` has the value `reduced calorie`, because it contains the substring `red`.

## Multi-Valued Property Searches

If a property specified in `keywordSearchPropertyNames` is multi-valued—for example, of type `String[]`—the keyword search algorithm uses the query `QueryBuilder.INCLUDESANY` to perform a single search for an exact match between any property value and any search value. For example, given a `keywords` property of type `String[]`, a keyword search for the values red, green, and blue generates this query:

```
keywords INCLUDES ANY ["red", "green", "blue"]
```

`INCLUDES ANY` searches only for exact matches, so this query does not return an item whose keywords contain substrings of the specified values. For example, while `reduced fat` contains a substring of the search value `red`, it is not an exact match and does not satisfy the query.

## Quick Searching

A quick search is a hybrid of keyword and text search that uses a user-supplied value as a keyword and compares it to all text property values in a given repository. In effect, it is a keyword search that specifies all properties as keywords. To enable a quick search, leave the `keywordsSearchPropertyNames` property blank.

Unlike other search methods, a quick search and keyword search can share the same component, because the two searches are similar and the quick search requires only minor configuration.

# Text Search

A text search takes a user-entered search string and performs text pattern matching on one or more text properties—for example, finds all recipes whose `longDescription` property contains “sauté.”



In order to create a text search form handler you must:

- Configure the appropriate [text search properties](#).
- [Enable full-text searches](#) in your repository component and database.

## Text Search Properties

To enable text searches in a form handler, you set the following properties:

- `doTextSearch` is set to true to enable text searching.
- `textSearchPropertyNames` specifies which properties to query.

The search string is specified by the `textInput` property, typically entered by the user through a form input field.

You configure text searching with the following properties:

### Required Text Search Properties

Property	Description
<code>doTextSearch</code>	Set to true in order to enable this component to handle text searches. A text search compares the user-supplied value to a property value that is identified in <code>textSearchPropertyNames</code> .
<code>textSearchPropertyNames</code>	<p>Properties in the <code>itemTypes</code> property to be searched text intended as search criteria. If this property is empty, all properties are searched in the item types identified by the <code>itemTypes</code> property (see <a href="#">Basic SearchFormHandler Properties</a>).</p> <p>Set this property as follows:</p> <p><i>item-type. property-name</i></p>

### Optional Text Search Properties

Property	Description
<code>allowWildcards</code>	<p>Determines whether a query can include the wildcard symbol (*). A wildcard can replace any set of characters in a search string.</p> <p>Default: true</p>



Property	Description
searchStringFormat	<p>Format that the text should follow. Each repository component uses this property to specify the text format available to the database. If not set, the form handler uses the default format.</p> <p>Available options include:</p> <p>ORACLE_CONTEXT: Oracle ConText</p> <p>MSSQL_TSQ: Microsoft SQL Server</p> <p>DBS_TEXT_EXT: IBM DB2 with the Text Extender package</p>
textInput	<p>List of strings that are always used in a search. If no keywords are specified for this property, search strings must be supplied by the user.</p>

## Enable Full-Text Searches

The implementation of text searching is RDBMS-specific and uses the database's text search facility, if any. If your database supports a full-text search engine, you must configure it as required by your application, and set the repository component's `simulateTextSearchQueries` property to false.

### *Simulated text searches*

If a full-text search engine unavailable—for example, your RDBMS does not support one—the SQL repository can simulate full-text searching by using the LIKE operator to determine whether the target value is a substring of any of the text properties being examined. To enable this feature, set the repository component's `simulateTextSearchQueries` property to true.

**Note:** Simulated full-text searching is useful for development purposes; however, performance is liable to be inadequate in a production environment.

You also might want to set `simulateTextSearchQueries` to true in order to test full-text searching on the default ATG database SOLID, which disallows full-text searching. After you migrate content to a production database that can handle full-text searching, be sure to reset the property to false.

## Hierarchical Search

A hierarchical search returns all parent and descendant items of the specified item. A hierarchical search looks in a family of items, starting from a given item level and extending to that item's descendants.

Each item type must have a multi-valued property whose value is a complete list of its ancestor IDs. Hierarchical searching restricts the search results to items whose ancestor items include the item specified in the `ancestorId` property.





The `ancestorPropertyName` property specifies the name of the ancestor item property. You can specify ancestor categories by setting properties manually.

### ***Hierarchical search properties***

To configure a hierarchical search tool, set these properties:

Property	Description
<code>ancestorId</code>	Repository ID that represents an inheritance scheme or lineage. This property is set to the repository ID of the item whose descendants you want to search, typically obtained through a hidden input tag, or supplied by the form user through an input field
<code>ancestorPropertyName</code>	Name that represents an inheritance scheme or lineage.
<code>doHierarchicalSearch</code>	Set to true in order to enable the component for hierarchical searches.

## Advanced Search

Also called parametric search, an advanced search provides search options for each property specified in the form handler's `advancedSearchPropertyNames` property. The advanced query is built from options selected by the search form users to further refine the catalog search.

For example, an advanced search might be based on a description, chef, and set of ingredients: find all recipes with the keyword breadmachine where the chef is Grandma and the ingredients include whole wheat flour.

Advanced search lets you manipulate HashMaps; it can also allow queries that involve other data types such as string, numeric, Boolean, and date.

### **Advanced Search Properties**

To enable advanced searches, you set the following form handler properties:

- `doAdvancedSearch` property is set to true to enable advanced searching.
- `advancedSearchPropertyNames` specifies the set of properties to search. Advanced searches are limited to the set of properties named here.

In order to configure a component for advanced searches, set the following properties:

### ***Required Advanced Search Properties***



Property	Description
doAdvancedSearch	If set to true, enables advanced searching.
advancedSearchPropertyRanges	<p>Property range used to narrow search results by integer values. The specified range is inclusive and uses this format:</p> <p><i>i tem-type. property-name. max</i> <i>i tem-type. property-name. mi n</i></p> <p>You can leave the search range open at either end by leaving the maximum or minimum value undefined.</p>

### Optional Advanced Search Properties

Property	Description
advancedSearchPropertyRanges	<p>Property range used to narrow search results by integer values. The specified range is inclusive and uses this format:</p> <p><i>i tem-type. property-name. max</i> <i>i tem-type. property-name. mi n</i></p> <p>You can leave the search range open at either end by leaving the maximum or minimum value undefined.</p>
advancedSearchPropertyVal ues	<p>List of property values used in the search query. The format should appear as follows:</p> <p><i>i tem-type. property-name= val ue</i></p>
cl earQueryURL	<p>The URL of the page to display when the <code>handl eCl earQuery()</code> method is invoked. The specified page should have search tools. If this property is empty, <code>handl eCl earQuery()</code> returns control to the current page. See <a href="#">Clearing Search Query Data</a> for more information.</p>
di spl ayName	<p>The name displayed for an item type when it is used as search criteria. Each item type has a default display name; this property overwrites the default.</p>



Property	Description
propertyValuesByType	<p>Holds the properties in advancedSearchPropertyNames and their values in a HashMap key/value relationship. This only applies to those properties in advancedSearchPropertyNames that are enumerated, RepositoryItems, or collection of RepositoryItems.</p> <p>When the key is the name of a property, the value is a collection of possible values.</p> <p>When the key is a repositoryId, the value is a collection of values for a specific property.</p> <p>When the key is a collection of repositoryIds, the value is repositoryIds.</p> <p>The format should appear as follows:</p> <p><i>item-type. property-name</i></p>

## How Advanced Searches Operate

The SearchFormHandler property propertyValuesByType is a HashMap that contains one key/value pair for each property named in advancedSearchPropertyNames whose type is enumerated, RepositoryItem, or a collection of RepositoryItems. The key is the name of the property and the value is a collection of the possible values. propertyValuesByType is useful for building forms where search form users can select, for example, the size of an item, where size is an enumerated property with a set of predefined values such as Small, Medium, and Large.

The following example shows how to implement an advanced search:

```
<%-- Create a select field to choose size and a default option --%>
Size: <dsp: select bean="MyAdvancedSearchFormHandler.propertyValuesByType.size">

<%-- Create a default choice --%>
<dsp: option value="">Any size</dsp: option>

<%-- Now create an option for each defined value for size --%>
<dsp: droplet name="ForEach">
  <dsp: param name="array"
    value="MyAdvancedSearchFormHandler.propertyValuesByType.size"/>
  <dsp: oparam name="output">
    <dsp: option param="element"/>
    <dsp: valueof param="element">Unknown size</dsp: valueof>
  </dsp: oparam>
</dsp: droplet>

</dsp: select>
```



## Searching a Range of Values

The advanced search feature lets you specify a range of values, as in this example:

Find all recipes that take between 5 minutes and 25 minutes to cook

The `advancedSearchPropertyRanges` property is a `HashMap` that you can set to hold maximum and minimum values. You create a range by setting a property's `max` and `min` keys and their respective values. For example:

---

```
<dsp:input type="text" bean=
  "MyAdvSearchFH. advancedSearchPropertyRanges. recipe. cookTime. min" value="5"/>
<dsp:input type="text" bean=
  "MyAdvSearchFH. advancedSearchPropertyRanges. recipe. cookTime. max" value="25"/>
```

---

In this example, `recipe` is an item type defined for the repository that supports this search component (`MyAdvancedSearchFormHandler`). The `cookTime` property is a primitive string. If another primitive type were used, a tag converter would be inserted as part of the `value` phrase.

Range settings are inclusive: in this example, the search result can include recipes that take 5 and 25 minutes to cook. You can leave the search range open at either end by leaving the maximum or minimum value undefined. For example:

Find all recipes that use at least 8 tomatoes

The code to implement this search might look like this:

---

```
<dsp:input type="text"
  bean="MyAdvSearchFH. advancedSearchPropertyRanges. recipe. tomatoes. min" value="8"/>
```

---

## Combination Search

The `SearchFormHandler` class lets you specify multiple search types in a single request. For example, you can search on both keywords and text, or you can combine advanced searching with hierarchical searching to find only items in a particular category.

Search types are combined according to the following rules:

- Text and keyword searches are combined with the OR operator, so a match on either set of criteria selects an item for inclusion in the search results.
- Hierarchical and advanced searches are combined with the AND operator, limiting the scope of the search to items that satisfy the hierarchical or advanced search requirements in addition to any specified text or keyword search criteria.

The query can be expressed in this format:



*(KeywordConditions OR TextConditions) AND HierarchicalConditions AND AdvancedSearchConditions*

For example, you have a set of recipes and you configure a `SearchFormHandler` to allow all four types of searches. The site visitor enters the following search criteria:

```
keywords=appetizer
textSearchPropertyNames=ingredients
textInput="boston lettuce"
hierarchicalCategoryId=vegan
propertyValues.fatContent="5 grams"
```

The search locates all appetizers plus all recipes whose ingredients mention Boston lettuce, but returns only the subset of those recipes that are found in the vegan category and have 5 grams of fat.

## Search Form Submit Operations

The `SearchFormHandler` supports two submit operations:

- `search` launches the search.
- `clearQuery` removes all previous query data.

### Executing the Search

You can define a submit control that invokes the `SearchFormHandler`'s `handleSearch()` method, in one of the following ways:

```
<dsp:input type="submit" [value="Label"] bean="searchform-handler.search"/>
```

```
<dsp:input type="image" {page|src}=path bean="searchform-handler.search"/>
```

The search operation also stores search results in the handler's `searchResultsByItem` property and makes them available to the `searchResults` property for the duration of the component's scope (the current session).

### Clearing Search Query Data

You can define a submit control that clears query data by invoking the handler's `handleClearQuery()` method, in one of the following ways:

```
<dsp:input type="submit" [value="Label"] bean="searchform-handler.clearQuery"/>
```

```
<dsp:input type="image" {page|src}=path bean="searchform-handler.clearQuery"/>
```

The default implementation of `handleClearQuery()` flushes all previous search query criteria and opens the page specified by the `ClearQueryURL` property. Providing this option to users is especially important

if the property `allowRefine` is set to true. In this case, each successive query appends its search criteria to previous search criteria. Providing a Clear Query button allows users to start a fresh search.

## Presenting Search Options

When you create a search form, it is good practice to provide users with a set of drop-down lists where they can specify the values to include in their search. The [PossibleValues](#) component, an instance of `atg.repository.servlet.PossibleValues`, takes enumeration or `RepositoryItem` properties, queries the database for their values, and returns these values for use by a servlet bean such as `ForEach`, which can display the query results.

For example, you might want to query Recipe items on three properties: Appliance, PrepTime, and Course. You can construct three drop-down lists that display the complete set of property values:

<div>Appliance ▼</div> <div> Oven  Toaster  Rice Cooker  Hot Pot  Fondue Pot  Microwave  Oven </div>	<div>Prep Time ▼</div> <div> 15 min or less  16-30 min  31-45 min  46-60 min  60+ min </div>	<div>Course ▼</div> <div> Appetizer  Soup  Salad  Entrée  Dessert </div>
--	--	--

A component that supports advanced searches can generate a list of possible property values without using the `PossibleValues` servlet bean as follows:

- Set `propertyValues` to hold the desired properties.
- Set `propertyValuesByType` to hold each property and its possible values in a key/value relationship.

For more information, see [How Advanced Searches Operate](#).

## Managing Search Results

`SearchFormHandler` class properties let you control the display of search results and help users access them. With these properties, you can:

- Set the maximum number of items returned by a query.
- Set the maximum number of items displayed on a single page.
- Get the number of items returned and the number of results pages.
- Set the current results page number and the total number of results pages.



- Provide links that enable browsing among multiple result pages.

## Search Results Properties

In order to make query result properties accessible, set the enableCountQuery property to true. The SearchFormHandler defines the following search results properties:

Property	Description
currentResultPageNum	1-based page number of active page. Set this property in order to display a given page.  The default setting is 1.
enableCountQuery	Enables access to other query result properties.
endCount	1-based number of the last item on the current page.
endIndex	0-based index number of the last item on the current page. The form handler uses this property to calculate the endCount property.
maxResultsPerPage	Maximum number of items that display on a page.
maxRowCount	Maximum number of items returned from a search. A value of -1 (the default) returns all items that satisfy the search criteria.
resultPageCount	Number of pages that hold search results.
resultSetSize	Number of items returned from the search.
startCount	1-based number of the first item on the current page.
startIndex	0-based index number of the first item on the current page. The form handler uses this property to calculate the startCount property.

**Note:** These properties apply to a search that involves one item type. If your search form handler allows for multiple item types, these properties might not behave as expected.

## Displaying Results Data

An application can distribute search results across several pages and provide cues that facilitate access to those pages. The following example uses search results properties together with the Compare servlet bean to set the range of items that appear on the first results page:

```
<dsp: droplet name="Compare">
  <dsp: param bean="MyAdvancedNavigationSearch.resultSetSize" name="obj 1"/>
  <dsp: param bean="MyAdvancedNavigationSearch.maxResultsPerPage" name="obj 2"/>
  <dsp: oparam name="greaterthan">
    Results
```



```
<dsp: val ueof bean="AdvancedNavi gati onSearch. startCount"/>
&nbsp; -&nbsp;
<dsp: val ueof bean="AdvancedNavi gati onSearch. endCount"/>
of
<dsp: val ueof bean="MyAdvancedNavi gati onSearch. resul tSetSi ze"/>
</dsp: oparam>
</dsp: dropl et>
```

---

In this example, the Compare servlet bean compares the number of items returned from the search (resul tSetSi ze) to the maximum number of items set to appear on a page (maxResul tsPerPage). When the query returns more items than can be displayed on one page, the first results page displays the range of items with the properties startCount and endCount, and the total number of returned items with resul tSetSi ze.

For example, given the following property settings:

```
resul tSetSi ze=372
maxResul tsPerPage=25
startCount=1
endCount=25
```

The previous code yields this display:

Resul ts 1 - 25 of 372

## Linking to Search Results

The currentResul tPageNum property can be used to create links that enable access to multiple search results pages. For example:

```
<dsp: a
  href="SearchResul ts. j sp" bean="MySearchFH. currentResul tPageNum" val ue="3">3
</dsp: a>
```

## Organizing Query Result Items

After executing a query, the SearchFormHandl er makes the search results available in two properties, which organize the same information differently:

- searchResul ts contains all items returned by the query, undifferentiated by item type.
- searchResul tsByI temType is a HashMap with one key/value pair for each item type returned by the query. The key is the item type name—the value specified in the form handler's i temTypes property—and the value is a collection of items of that type that were returned by the query.

For example, if you search for cuisine and recipe items in the food repository, the searchResul ts property contains items of type cui si ne and reci pe. The searchResul tsByI tem type property has a





cuisine key whose value is a collection of cuisine items, and a recipe key whose value is a collection of recipe items.

Within each collection, the items are in the order of their retrieval from the database. To sort the display order of items, use a servlet bean such as `ForEach`.

The following example uses `ForEach` to iterate over the `searchResultsByItemtype` property, and display only the recipes returned by the search sorted by their display name:

---

```
<p>Your search returned the following products: </p>
```

```
<dsp:droplet name="ForEach">
```

```
<dsp:param name="array" bean="CatalogSearch.searchResultsByItemtype.recipe"/>
```

```
<dsp:param name="sortProperties" value="+displayName"/>
```

```
<dsp:oparam name="output">
```

```
<li><dsp:valueof param="element.displayName">Unknown recipe</dsp:valueof>
```

```
</dsp:oparam>
```

```
<dsp:oparam name="empty">
```

```
<p>No matching recipes were found.
```

```
</dsp:oparam>
```

```
</dsp:droplet>
```

---





# 10 RepositoryFormHandler

Web application forms are commonly used to view and update database data. The `atg.repository.servlet.RepositoryFormHandler` class provides methods and properties for working with repository items. You can use a component of this class to add, update, and delete repository items that use the same item descriptor. A `RepositoryFormHandler` can be used with any repository type: HTML, XML, LDAP, and SQL.

The ATG platform also provides the class `atg.droplet.sql.SimpleSQLFormHandler`, which supports direct interaction with an SQL database. For more information about this form handler, see [Appendix C: SimpleSQLFormHandler](#).

**Note:** Depending on the contents of your ATG software, your installation might include components that are based on a subclass of `RepositoryFormHandler`.

The `RepositoryFormHandler` offers several benefits:

- Requires only the repository item type for updates.
- Supports all repository types: HTML, XML, LDAP, and SQL.
- Caches repository data.
- Optimizes data access.

Before reading this chapter, you should have a basic understanding of repositories, components, forms, and form handlers. The following table shows where to review these topics:

Topic	Manual
Repositories	<a href="#">ATG Repository Guide</a>
Components, form handlers, <code>GenericFormHandler</code> class	<a href="#">ATG Programming Guide</a>

You might use a `RepositoryFormHandler` in a customer service application that requires internal employees to manage inventory. For example, a Web site that sells music might have an administration interface for tracking in-stock items. The site accesses a music repository, which defines three item descriptors: CDs, Artists, and Songs. This site requires three `RepositoryFormHandler` components, one for each item descriptor.



This chapter covers the following topics:

- [RepositoryFormHandler properties](#)
- [RepositoryFormHandler submit operations](#)
- [Updating Item Properties from the value Dictionary](#)
- [RepositoryFormHandler navigation properties](#)
- [Updating multi-valued properties](#)
- [Modifying properties that refer to items by repositoryId](#)
- [Managing map properties](#)

## RepositoryFormHandler Properties

When you start an application that includes the ATG platform, the application uses a properties file to instantiate RepositoryFormHandler components and to set initial values in them. The following sections describe these properties.

**Note:** Several [RepositoryFormHandler navigation properties](#) such as `updateSuccessURL` and `updateErrorURL` control navigation after a form operation is complete. These are discussed separately.

### Required RepositoryFormHandler Properties

The following properties must be set:

#### ***repository***

Specifies the repository where items are added, updated, and deleted. This property contains a pointer to the repository component. This property is required unless the handler's `repositoryPathName` property is set to the repository's Nucleus path.

#### ***itemDescriptorName***

A string that specifies the item descriptor items handled by this RepositoryFormHandler.

### Optional RepositoryFormHandler Properties

The following properties provide additional functionality to forms that use a RepositoryFormHandler:

#### ***checkForRequiredProperties***

A boolean, specifies whether the form handler checks that all required properties are present during operation.



### ***clearValueOnSet***

A boolean, specifies whether the value dictionary for an item is cleared and reset from the database when its repository property changes. When this property is set to true, data in the value dictionary reflects the item's database values. When set to false, the value dictionary can retain some data as the repository changes from one item to another.

If the `clearValueOnSet` property is set to true, any change to a repository ID clears all other properties for that item from the value dictionary. To preserve other property changes, set repository ID to false; alternatively, raise the priority for the repository ID input field, so updates from this field occur before updates to other properties are written to the item's value dictionary. For example:

---

```
<dsp:input type="hidden" priority=10
bean="atg/dynamo/dropLet/MyRepositoryFormHandler.repositoryID" value="Add CD"/>
```

---

### ***createTransientItems***

A boolean, specifies whether new items should remain transient. The default is false.

### ***extractDefaultValuesFromItem***

A boolean, specifies whether the default values for properties that use a value dictionary display in a form that renders these values available for modification.

### ***mapKeyValueSeparator***

By default set to = (equals), used when editing a Map through its [keysAndRepositoryIDs](#) property to set both the key and the map from a single form element. This string is used to separate the key from the repository id

### ***removeReferencesToDeletedItems***

A boolean, specifies whether deleting an item also deletes all references to that item. Set this property to false (the default) if the items to delete have no references, and avoids. This avoids unnecessary lookups and performance delays.

If set to true, an item deletion automatically invokes the `RepositoryUtils.removeReferencesToItem` method, which locates all items that reference the target item and deletes them first. For more information on this method, see the [Development, Testing and Debugging with the SQL Repository in the ATG Repository Guide](#).

### ***requireIdOnCreate***

A boolean, specifies whether an item ID must be user-supplied when an item is created. If set to true (the default), an error is thrown when the form is submitted and no ID exists for the new item. If set to false, the repository or database automatically generates the item ID.



## RepositoryFormHandler Submit Operations

The RepositoryFormHandler supports the following operations for form submission:

Operation	Function
create	Creates a repository item based on the value set in the form and adds it to the repository. If the repositoryId property is specified in the form, the new item is created with the given repository ID; otherwise, an auto-generated ID is used.
delete	Deletes from the repository the item specified by repositoryId.
update	Updates the item described by repositoryId from the form values.

You associate one of these operations with a submit input tag. For example, the following input tag defines a submit button that creates a repository item:

```
<dsp:input type="submit"
  bean="atg/dynamo/dropLet/MyRepositoryFormHandler.create" value="Add CD" />
```

## Updating Item Properties from the Value Dictionary

In order to use the RepositoryFormHandler to update item property values from a form, reference the item property in this format:

```
bean="nucl eus-path/ formhandl er-component. val ue. property"
```

For example:

```
<dsp:input type="input-type"
  bean="atg/dynamo/dropLet/MyRepositoryFormHandler. val ue. name" />
```

The RepositoryFormHandler's value property is a Dictionary of property/value pairs that temporarily stores all pending property values. The RepositoryFormHandler writes all values from this dictionary on form submission as a single transaction, thus preventing profile properties from being left in an inconsistent state.

### Updating hierarchical properties

In order to update a property that is referenced by another property, use the following notation:

```
bean=
  "nucl eus-path/ formhandl er-component. val ue. property. sub-property[. sub-property]. . ."
```



For example, the following input tag references the `city` property, which is referenced from the `address` property:

```
<dsp:input type="text" bean="MyRepositoryFormHandler.value.address.city"/>
```

In this case, the `address` property points to an ID that represents an address item with its own set of properties, one of which is `city`.

## RepositoryFormHandler Navigation Properties

After a form submission operation—create, update, or delete—is complete, the `action` attribute of the `dsp:form` tag specifies the page to display. You might want to specify several different pages, where the user's destination depends on the nature of the form operation and whether or not it succeeds.

A `RepositoryFormHandler` has a set of properties you can use to control navigation after a form operation. These properties specify the URLs where the user is redirected on certain error and success conditions. Each submit operation has the corresponding success and failure URL properties:

RepositoryFormHandler operation	Success/failure URL properties
create	createSuccessURL createErrorURL
update	updateSuccessURL updateErrorURL
delete	deleteSuccessURL deleteErrorURL

The value of each property is a URL relative to the page specified by the form's `action` attribute or a local URL defined absolutely from the root. The default value of each property is null, so you must set these properties explicitly. You can specify the values of these URL properties in the form handler component, or with hidden input tags in the JSP. If these property values are not set, the `action`-specified page is displayed on form submission.

For example, the following hidden input tag redirects users to `updateStockFailure.jsp` in the event that an update operation fails:

```
<dsp:input type="hidden"
bean="atg/dynamo/drop/et/MyRepositoryFormHandler.updateErrorURL"
value="updateStockFailure.jsp"/>
```



## Updating Multi-Valued Properties

By default, the `RepositoryFormHandler` updates a property value by overwriting its old value with the new one. In the case of multi-valued properties, you can modulate this behavior in the following ways:

- Add the form values to the current property values; or use the form values to remove any current property values that match.
- Specify the maximum number of values to add, update, or remove.

**Note:** These property options are available only if the `extractDefaultValuesFromItem` property is set to `true`.

### *updateMode*

The form handler property `updateMode` specifies how changes to a property value affect existing data. You can set `updateMode` to one of these values:

- `append` appends the new form values to the current ones.
- `prepend` prepends the new form values to the current ones.
- `replace` (the default behavior) overwrites old values with new values
- `remove` removes any existing values that match the selected form values.

For example, the list property `hobbies` has the following string values:

Mountain Biking  
Running  
Canoeing

The form contains a hidden input tag that sets the update mode for the hobbies property with the following format:

---

```
<dsp:input type="hidden"
  bean="myHandler.value.hobbies.updateMode" value="update-mode" />
```

---

The following table shows the various values for *update-mode* and how various input scenarios set the property's value:

updateMode value	User input	Resulting property value
append	Rafting Swimming	Mountain Biking Running Canoeing Rafting Swimming





updateMode value	User input	Resulting property value
prepend	Rafti ng Swi mmi ng	Rafti ng Swi mmi ng Mountai n Bi ki ng Runni ng Canoei ng
repl ace	Rafti ng Swi mmi ng	Rafti ng Swi mmi ng
remove	Mountai n Bi ki ng Canoei ng	Runni ng

### **numNewValues**

If updateMode is set to prepend or append, you must also set numNewValues to a positive integer. By default, it is set to 0. The following code example specifies to append a gift list to the list property giftLists:

Enter Gift List:

```
<dsp:input type="text" bean="MyRepositoryFormHandler.value.giftLists"/>
<dsp:input type="hidden"
  bean="MyRepositoryFormHandler.value.giftLists.updateMode" value="append"/>
<dsp:input type="hidden"
  bean="MyRepositoryFormHandler.value.giftLists.numNewValues" value="1"/>
```

## Modifying Properties that Refer to Items by repositoryId

When a RepositoryFormHandler uses a value dictionary, it adds the sub-property repositoryId to all multi-valued properties. This property references other items and their property values so you can update a collection of items using their IDs.

A form might use a RepositoryFormHandler to update user information, such as access controls provided through user roles and organizational groups. For example, you might want to enable an administration user to edit a user profile roles property, which holds a set of ID objects that reference role items. Each user profile has a property called currentOrganization, which maintains the sub-property relativeRoles. Because a user can have more than one role, relativeRoles is a set of role ID objects that designate user responsibilities and access requirements.

This form is designed to add the current organization to a user's profile so the user acquires some of the roles associated with that organization. The form performs these tasks:

- Accesses the contents of the current organization's relativeRoles property so the form user can select one or more roles.



- Saves the selected roles to a user profile's `roles` property.

In order to implement this form, a `ForEach` servlet bean iterates through the `currentOrganization.relativeRoles` property and presents each role ID it finds as a name in the form page, through a selection box:

---

```
<dsp: select bean="MyRepositoryFormHandler.value.roles.repositoryIds">
  <dsp: dropLet name="ForEach">
    <dsp: param name="array" bean="Profile.currentOrganization.relativeRoles"/>
    <dsp: oparam name="output">
      <dsp: option param="element.repositoryId">
        <dsp: valueof param="element.name">unnamed role</dsp:valueof>
      </dsp:oparam>
    </dsp:dropLet>
  </dsp:select>
```

---

In this example, a `dsp:option` tag is generated for each role in the `relativeRoles` property. The option value is the ID of the role repository item.

The form user selects several roles; on form submission, the `MyRepositoryFormHandler.values.roles.repositoryIds` contains the list of repository IDs checked by the user. These are used to update the user's `roles` property.

## Managing Map Properties

In most ATG operations, you update a Map property by updating a specific key value as if it were a servlet bean property:

```
<dsp:input type="text" bean="MyBean.value.addresses.city"/>
```

While `addresses` is a Map property and `city` is a key, this tag represents `addresses` and `city` as properties. Making a change to a key's value requires entering text into a text box and submitting the operation.

The `RepositoryFormHandler` and Maps have several properties that offer more flexibility in updating Map properties:

- `editMapsAsLists` is a `RepositoryFormHandler` property that, if set to true, lets you make changes to the Map keys and values.
- `keysAndRepositoryIds` is a Map property that you can use to update the Map with a series of key/repository ID pairs.
- `keysAndValues` is a Map property that you can use to update the Map with a series of primitive key/value pairs.



**Note:** The approach described in this section for updating Map properties applies only to the RepositoryFormHandler. For a generic approach to setting Map properties in all forms, see the [Map Properties](#) section in the [Forms](#) chapter.

### *editMapsAsLists*

You typically interact with Maps by treating them as servlet beans. For example, to expose a value, you specify the key as an input parameter. You cannot add new key/value pairs when you work with Maps in this way.

Alternatively, you can treat a Map as a list if the RepositoryFormHandler's property `editMapsAsLists` is set to true (the default setting). This lets you use `dsp: setValue` and `dsp: input` tags to expose and update Maps like a list property.

To use this feature:

1. Make sure `editMapsAsLists` is set to true.
2. Set the Map's `updateMode` property to append, prepend, replace, or remove. The default value is replace.

**Note:** Because Map elements are unordered, update modes prepend and append have the same effect. The Map itself determines the actual placement of new key-value pairs.

3. If the Map's `updateMode` property is set to prepend or append, also set its `numNewValues` property to a positive integer.
4. Set up the form input fields as required by the `updateMode` setting. For example, if `updateMode` is set to prepend or append, the form should contain text boxes that accept values for the new Map entries to be added.

**Note:** All keys in a Map must be unique. If `updateMode` is set to append or prepend and a key value in the form duplicates an existing Map key, on form submission the form values associated with the key overwrite the corresponding Map element.

For example, a page where users can enter home and work addresses might include this code:

---

```
<dsp:input type="hidden"
  bean="MyRepositoryFormHandler.value.addresses.numNewValues" value="1"/>
<dsp:input type="hidden"
  bean="MyRepositoryFormHandler.value.addresses.updateMode" value="append"/>
```

Enter Home Address:

```
<dsp:input type="hidden"
  bean="MyRepositoryFormHandler.value.addresses.keys[0]" value="home"/>
<br/>Street: &nbsp;
<dsp:input type="text"
  bean="MyRepositoryFormHandler.value.addresses[0].street" value=""/>
<br/>City: &nbsp;
<dsp:input type="text"
  bean="MyRepositoryFormHandler.value.addresses[0].city" value=""/>
```



```
<br/>State: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[0].state" value=""/>
<br/>Postal Code: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[0].postalCode" value=""/>
```

Enter Work Address:

```
<dsp: input type="hidden"
  bean="MyRepositoryFormHandler.value.addresses.keys[1]" value="work"/>
<br/>Street: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[1].street" value=""/>
<br/>City: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[1].city" value=""/>
<br/>State: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[1].state" value=""/>
<br/>Postal Code: &nbsp;
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.addresses[1].postalCode" value=""/>
```

This page supplies two sets of four text boxes, one set for home address data and identified by the key home, and the second set for work address and identified by the key work. On form submission, the RepositoryFormHandler uses the form values to add two new elements to the addresses Map.

### **keysAndRepositoryIds**

The keysAndRepositoryIds property lets you simultaneously set a Map key and value, where the Map holds a series of keys that map to repository IDs. Without keysAndRepositoryIds, performing this operation requires two input tags:

```
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.favorites.keys[0]" value=""/>
<dsp: input type="text"
  bean="MyRepositoryFormHandler.value.favorites.repositoryIds[0]" value=""/>
```

Alternatively, a single input tag that specifies the Map's keysAndRepositoryIds property can set each key-repository ID pair. This property is typically set with the following format:

*key=repository-id*

**Note:** The string used to separate the key and repository ID must be the value set by the RepositoryFormHandler property `mapKeyValueSeparator`—by default, = (equals).

You can use page parameters to set dynamic key repository ID values and configure a Format bean to provide the syntax. For example, you might want to track the brands each customer prefers. Each brand is



associated with a business, and one business can maintain several brands; so you want to organize the brands by parent business. Therefore, you display the brands as text boxes. When checked, you want the favorites Map to save a business name and brand ID as the key and repositoryId respectively. You might design your code like this:

---

```
<dsp: form action="keysands.js" method="post">
  <dsp: input type="text"
    bean="MyRepositoryFormHandler. value. favorites. updateMode" value="prepend"/>
  <br/>
  <dsp: droplet name="/atg/dynamo/droplet/RQLQueryForEach">
    <dsp: param bean="/atg/userprofiling/ProfileAdapterRepository"
      name="repository"/>
    <dsp: param name="itemDescriptor" value="business"/>
    <dsp: param name="queryRQL" value="all"/>
    <dsp: oparam name="output">
      <dsp: droplet name="/atg/dynamo/droplet/Format">
        <dsp: param name="key" param="index"/>
        <dsp: param name="repositoryId" param="element.repositoryId"/>
        <dsp: param name="format" value="{key}={repositoryId}"/>
        <dsp: oparam name="output">
          <dsp: input
            bean="MyRepositoryFormHandler. value. favorites. keysAndRepositoryIds"
            paramvalue="message" type="checkbox"/>
          <dsp: valueof param="message">Not set</dsp: valueof>
          <br/>
        </dsp: oparam>
      </dsp: droplet>
    </dsp: oparam>
  </dsp: droplet>
  <dsp: input bean="MyRepositoryFormHandler. update" type="submit" value="Update"/>
</dsp: form>
```

---

The RQLQueryForEach and the ProfileAdapterRepository retrieve the brands saved as business items in the repository. Assume that each brand displays in a text box in the page and when it is modified, the Format bean associates each business name and brand ID to the key and repositoryId properties of a Profile component favorites Map.

### **keysAndValues**

You can configure keys and values that are primitive data types through a Map's keysAndValues property. As with the [keysAndRepositoryIds](#) property, you typically set this property with this format:

*key=value*

The string used to separate the key and its value must be the value set by the RepositoryFormHandler property [mapKeyValueSeparator](#)—by default, = (equals).





# 11 User Profile Forms

This chapter shows how to use forms in JSPs to create and modify user profiles. It includes the following sections:

- [ProfileFormHandler](#) describes how to use the ProfileFormHandler, a component in ATG Personalization that associates values entered in forms with profile properties.
- [Multi-Profile Form Handlers](#) describes how to use the form handlers MultiProfileAddFormHandler and MultiProfileUpdateFormHandler. These form handlers let you create forms that interact with multiple user profiles.
- [Profile Form Examples](#) provides examples of various profile forms.

Before reading this chapter, you should be familiar with how the ATG platform works with forms, as described in the [Forms](#) chapter.

You can also use the ATG Control Center to create JSPs that contain profile forms for standard registration and login. For more information, see [Creating Profile Form Pages](#) in [Appendix E: ATG Document Editor](#).

For information about setting up a profile repository, see the *Setting Up a Profile Repository* chapter in the [ATG Personalization Programming Guide](#).

## ProfileFormHandler

Sites usually create and modify profiles through forms that invoke the ProfileFormHandler. Its Nucleus location is at:

```
/atg/userprofiling/ProfileFormHandler
```

You can use the ProfileFormHandler to set user profile properties without writing any Java or SQL code. The ProfileFormHandler handles the following tasks:

- Profile creation and updates
- User login and logout.
- Assignment of existing roles and organizations to individual users and groups of users.

### ProfileFormHandler Submit Operations

The ProfileFormHandler supports the following operations that can be specified for form submission:



Operation	Function
cancel	Cancels any changes the user has made to values in the form but has not yet submitted.
changePassword	Changes the password property of the profile to the new value entered by the user.
clear	Clears the value Dictionary.
create	Creates a permanent profile and sets the profile properties to the values entered in the form.
delete	Deletes the current profile.
login	Uses the login and password values entered by the user to associate the correct profile with that user.
logout	Resets the profile to a new anonymous profile and optionally expires the current session.
update	Modifies the properties of the current profile.

For example, when a user logs into a site, a form can invoke the login operation as follows:

```
<dsp:input bean="ProfileFormHandler.login" type="submit" value="Submit"/>
```

Each operation is associated with a handler method. For example, the previous example calls `handleLogin()`. For detailed information about `ProfileFormHandler` methods, see *Working with User Profiles* in the [ATG Personalization Programming Guide](#) for information.

## Setting Profile Values

Input fields on a registration page are used to set the `ProfileFormHandler`'s `value` property. The `value` property is a Dictionary of property/value pairs that temporarily stores pending values for an operation on the current profile. This enables the `ProfileFormHandler` to set all profile properties on form submission as a single transaction, so profile properties are not left in an inconsistent state.

For example, the following extract from a registration form gathers a user's name, email address, and gender. The `value` Dictionary stores user entries as property/value pairs. When the form is successfully submitted, the `ProfileFormHandler.create` operation creates a user profile and writes the contents of the `value` Dictionary to the corresponding profile properties.

```
First Name: <dsp:input bean="ProfileFormHandler.value.firstname" maxLength="30"
size="25" type="text"/>
Last Name: <dsp:input bean="ProfileFormHandler.value.lastname" maxLength="30"
size="25" type="text"/>
Email Address: <dsp:input bean="ProfileFormHandler.value.email" maxLength="30"
```





```

size="25" type="text"/>
<dsp:input bean="ProfileFormHandler.value.gender" type="radio"
  value="female"/>Female
<dsp:input bean="ProfileFormHandler.value.gender" type="radio" value="male"/>Male
<dsp:input bean="ProfileFormHandler.create" type="submit" value="Save"/>

```

Property/value pairs in the `value` Dictionary do not always correspond to profile properties; some might be required by the form handler in order to perform its operations. For example, a Change Password form should provide three input text fields, as shown here:

```

<dsp:input bean="ProfileFormHandler.value.olddpassword" maxsize="35" size="35"
  type="password"/>
<dsp:input bean="ProfileFormHandler.value.password" maxsize="35" size="35"
  type="password"/>
<dsp:input bean="ProfileFormHandler.value.confirmpassword" maxsize="35" size="35"
  type="password"/>
<dsp:input bean="ProfileFormHandler.changePassword" type="submit" value="Save New
  Password"/>

```

The `value` Dictionary stores the values from all fields; but only one—`value.password`—is written to the profile. The other two—`value.olddpassword` and `value.confirmpassword`—are required by the `ProfileFormHandler` to verify the user's identity, and to confirm that the user entered the new password correctly, respectively. If both conditions are true, the `ProfileFormHandler.changePassword` operation updates the profile password; otherwise, it returns an error.

## ProfileFormHandler Properties

`ProfileFormHandler` component properties are generally configured correctly for most situations. You can reset these properties in two ways:

- Reconfigure properties in the `ProfileFormHandler` component. The configured values are used to initialize each new instance of the component.
- Override a component's configured value by setting the property value from the current page. A value set this way applies only to the instance of the `ProfileFormHandler` that is associated with this page.

The following sections describe `ProfileFormHandler` properties that are

### ***badPasswordDelay***

Specifies in milliseconds the amount of time to wait before responding when a user submits an incorrect password. The delay serves to protect against a password-guessing hammering attack, where a large number of passwords are provided successively in an attempt to guess the correct one. The default value is 1000 (1 second).

***clearValuesOnCreate***

A Boolean, specifies whether to clear the value Dictionary on submission of a form to create a user profile. The default value is true.

***clearValuesOnUpdate***

A Boolean, specifies whether to clear the value Dictionary on submission of a form to update a user profile. The default value is true.

***confirmPassword***

A Boolean property, determines whether the user must confirm a password by reentering it. In general, this property's configured value should be false, so users only need to enter their passwords once—for example, on login. However, a new user registration page or reset password page should override the configured value and set `confirmPassword` to true, in order to prevent users from making mistakes when setting their passwords:

```
<dsp:input bean="ProfileFormHandler.confirmPassword" type="hidden"
value="true"/>
```

When this property is set to true, a form must have two text input fields for entering and reentering the password (see the [Create a User Profile](#) example).

***extractDefaultValuesFromProfile***

A Boolean property, determines whether the values in the ProfileFormHandler's value Dictionary are initially set from the current user's profile. If set to true, form fields are populated from the corresponding profile properties.

The property's default setting is true. If set to false, form fields are not initially populated from the profile, which might yield undesired results. For example, if `extractDefaultValuesFromProfile` is false, a form for updating profile information, such as contact data, is initially rendered without existing profile data. This can yield empty fields and require the user to reenter data. If any form fields are left empty, the corresponding entries in the ProfileFormHandler's value Dictionary are also empty, and on form submission are written back as null values to user's profile.

In general, you should set this property to false only for appropriate pages such as a form to register new users, as follows:

```
<dsp:setvalue bean="ProfileFormHandler.extractDefaultValuesFromProfile"
value="false"/>
```

***propertiesToCopyOnLogin***

An array property that lists profile properties to copy from the transient profile to the permanent profile on user login. These values overwrite any values previously stored in the user's permanent profile.

For example, the multi-valued `pagesViewed` property in a user's transient profile lists the names of pages that the user visited before logging in. If `propertiesToCopyOnLogin` includes `pagesViewed`, the names of the site pages a user visited before logging in are written to `pagesViewed` in the user's permanent profile.



If a member visits five pages before logging in, the value of the `pagesViewed` property in the transient profile is a list of the names of these five pages. When the member logs in, this value is copied to the permanent profile, and the previously existing list of pages in `pagesViewed` is overwritten, rather than appended to.

### ***trimProperties***

An array that lists names of profile properties, where leading or trailing spaces are automatically removed on form submission. By default, this property includes the properties `login` and `email`.

For example, if a user includes a space before or after an email address when filling out the registration form, and `trimProperties` includes the `email` property, the extra space is automatically removed on form submission.

### ***updateRepositoryId***

Used together with the `update` operation to ensure that the correct profile is updated. The default value is null; when a page containing the `update` operation is rendered, a hidden input tag can set this property to the repository ID of the current profile. For example:

```
<dsp:input bean="ProfileFormHandler.updateRepositoryId"
beanvalue="ProfileFormHandler.repositoryId" type="hidden"/>
```

On form submission, the `update` operation is called and the `ProfileFormHandler` compares the current user profile ID to `updateRepositoryId`. The current profile is updated only if the two IDs match. A mismatch typically occurs for two reasons:

- The original session expired
- The original user logged out or logged in via a different window.

### ***generateProfileUpdateEvents***

A Boolean property, specifies whether to send a JMS message when a profile is updated. The `ProfileUpdateTriggers` fires a separate JMS event for every updated profile. The default value is false.

### ***profileUpdateEvent***

Holds the property update information when generating `ProfileUpdateEvents`.

For more information, see *User Profiling Tools* in the *Working with User Profiles* chapter of the [ATG Personalization Programming Guide](#).

### ***profileUpdateTrigger***

Generates the `ProfileUpdateEvents` when requested.

## **Multi-Valued Properties**

In order to facilitate updates to multi-valued profile properties, the `ProfileFormHandler` provides the `addMulti` operation. This operation appends a new value to a multi-valued property in the `ProfileFormHandler`'s `value` Dictionary. `addMulti` can add multiple primitive values to types `array`, `set`

and `list`. The `addMulti` feature cannot handle multiple `Map` type values, or multiple items, as opposed to primitive data. To add multiple items, you must write your own form handler, as described in the *Working with Forms and Form Handlers* chapter of the [ATG Programming Guide](#).

Unlike the other `ProfileFormHandler` operations, `addMulti` does not update the user profile when its submit button is clicked. Instead, it appends the entered value to the `value` Dictionary. Thus, the form user can add multiple values to the property before submitting them. Therefore, a page that implements an `addMulti` submit button must also provide a submit button that creates or updates the user profile.

The following code shows how a form might contain an `addMulti` operation:

---

```
<dsp: select multiple="true" bean="ProfileFormHandler.value.interests"
name="interests" size="4">
  <dsp: droplet name="ForEach">
    <dsp: param bean="ProfileFormHandler.value.interests" name="array"/>
    <dsp: oparam name="output">
      <dsp: option paramvalue="element" selected="true"/><dsp: valueof
param="element">null</dsp: valueof>
    </dsp: oparam>
  </dsp: droplet>
</dsp: select>

<dsp: input name="Add Interests" type="hidden" value="AddMultiProperty_interests,
AddMultiValue_interests"/>
<dsp: input name="AddMultiProperty_interests" type="hidden"
value="interests"/>
<dsp: input name="AddMultiValue_interests" type="text"/>
<dsp: input bean="ProfileFormHandler.addMulti" name="addMultiProperty"
type="submit" value="Add Interests">
```

---

This code example contains the following steps:

1. The `dsp: select` tag specifies how many values for the `interests` field are available in the form.
2. The `ForEach` servlet bean iterates through the array of `Strings` specified in the bean. `ProfileFormHandler.value.interests` parameter. If an interest is selected, it outputs the value; otherwise, it outputs null as the interest element value.
3. A hidden input tag defines two text strings; these strings link the next two subsequent input tags:
  - `AddMultiProperty_interests` is referenced by a hidden input tag's name attribute; this input tag's `value` attribute identifies the target Dictionary property.
  - `AddMultiValue_interests` is referenced by the text input tag's name attribute. Form users use this field to enter new values for the target property.

The tag's name attribute is set to `Add Interests`; this links the Dictionary property and text input field with the `addMulti` submit button.



4. The tag that defines the addMulti submit button must set its name and value attributes as follows:

- name must be set to addMulti Property
- value must be set to Add Interests

**Note:** In order to enable addMulti operations for several multi-valued properties on the same page, each property must have its own pair of unique identifiers as described above, and a unique string for its submit button's value attribute.

## ProfileFormHandler Navigation Properties

After a profile form operation—registration, login, or logout—is complete, you might want to redirect the user to a new page. As in standard HTML, you can use the action attribute of the form tag to specify a page to display on form submission.

The action attribute can specify only a single page. You might want to specify several different pages, where the user's destination depends on the nature of the form operation and whether or not it succeeds.

The ProfileFormHandler has a set of properties you can use to control navigation after a form operation. These properties specify the URLs where the user is redirected on certain error and success conditions. Each operation except cancel has corresponding success and failure URL properties:

ProfileFormHandler operation	Success/failure URL properties
changePassword	changePasswordSuccessURL changePasswordErrorURL
create	createSuccessURL createErrorURL
delete	deleteSuccessURL deleteErrorURL
login	loginSuccessURL loginErrorURL
logout	logoutSuccessURL logoutErrorURL
update	updateSuccessURL updateErrorURL

**Note:** The cancel operation has only one navigation property, cancel URL.

The value of each property is a URL relative to the page specified by the form's action attribute or a local URL defined absolutely from the root. The default value of each property is null, so you must set these properties explicitly. You can specify the values of these URL properties in the form handler component,



or with hidden input tags in the JSP. If these property values are not set, the action-specified page is displayed on form submission.

The following example sets the value of `loginErrorURL`, redirecting the user to `login_failed.jsp`. If the lookup operation succeeds, and the value of `loginSuccessURL` is not set, the user is redirected to the action-specified page `index.jsp`:

---

```
<dsp:form action="index.jsp" method="post">
<dsp:input
  bean="ProfileFormHandler.loginErrorURL" type="hidden" value="login_failed.jsp"/>
...
</dsp:form>
```

---

## Single Profile Form Handler Error Messages

The ATG servlet bean `ProfileErrorMessageForEach` detects errors that occur when a user submits a form, and displays the error messages associated with these errors. `ProfileErrorMessageForEach` is an instance of the `ErrorMessageForEach` servlet bean, and has the following pathname:

```
/atg/userprofiling/ProfileErrorMessageForEach.
```

For an example of using `ProfileErrorMessageForEach` in a profile form page, see [Create a User Profile](#). For information about modifying the messages this component displays, see [Appendix B: ATG Servlet Beans](#).

## ProfileFormHandler Scope

A `ProfileFormHandler` component can be request- or session-scoped. The default scope is request: each time a user accesses a page with a profile form, the ATG platform creates an instance of the `ProfileFormHandler` that uses the configured values of the component's properties. You can override the values of these properties using hidden input tags, but the values set this way do not carry over if the user accesses another page with a profile form.

Property values can persist across multiple pages if the `ProfileFormHandler` is session-scoped. For example, a long registration form might split across two pages, with the second page containing the form's submit control. Navigation controls enable the user to move between the two pages before submitting the form. In order to retain values from both pages, the `ProfileFormHandler` must be session scoped; otherwise, values from one page are lost when the user navigates to the other.

If you set the `ProfileFormHandler` to be session-scoped, be careful how you use hidden input fields to override default property values. The new values remain in effect throughout the session unless explicitly changed. For example, a registration page sets `extractDefaultValuesFromProfile` to false. If during the same session, the user accesses another page to update profile information, the fields in this form are not filled in with values from the profile unless that page explicitly sets `extractDefaultValuesFromProfile` back to true.

After a user logs in or submits profile data, the `ProfileFormHandler` automatically calls the `clear` operation if `clearValuesOnCreate` or `clearValuesOnUpdate` is set to true—the default value for both.



The clear operation clears the temporary value Dictionary. This prevents leftover keys from corrupting another form if the user enters new data. This method is useful for session-scoped form: the form can safely retain information across multiple pages in its value Dictionary for a given session, and ensure that the data is cleared on form submission. To call the clear operation on a form, insert this line:

```
<dsp:setvalue bean="ProfileFormHandler.clear" value="" />
```

The clear operation executes when this line is rendered.

## Multi-Profile Form Handlers

Two form handler components are available that let administrators create and update multiple user profiles at the same time:

- MultiProfileAddFormHandler sets up multiple user profiles.
- MultiProfileUpdateFormHandler updates multiple user profiles.

Both components can differentiate profile properties whose values are shared, from those that must be set for each user. Both handlers have the same Nucleus path, located in `/atg/userprofiling/`.

Because login and logout properties are designed for single users who update their own profiles, these are omitted from the multi-profile form handlers. A user who has access to multiple forms is typically an administrator is already logged into the Web site, and is granted the necessary permissions to modify multiple profiles. For example, in the Motorprise Reference Application, a user logs in as a Web site administrator and is then granted security permission to edit multiple profiles. For more information, see the [ATG Business Commerce Reference Application Guide](#).

You can use multi-profile form handlers to perform the following tasks:

- Assign roles to a group of user profiles.
- Assign a group of user profiles to the same organization.
- Perform the same changes to a group of user profiles—for example, update contact information for relocated employees, or delete an entire group of user profiles.

For more information on working with roles and organizations see the *Working with the Dynamo User Directory* chapter in the [ATG Personalization Programming Guide](#).

### Multi-Profile Form Handler Submit Operations

The following table lists operations that are supported by MultiProfileAddFormHandler and MultiProfileUpdateFormHandler that can be specified for form submission:



Operation	Function
create	Valid only for <code>MultiProfileAddFormHandler</code> , creates a group of user profiles with the profile properties specified in the form.
update	Updates a group of user profiles with the properties specified in the form.
delete	Deletes a group of user profiles.
cancel	Cancels any unsubmitted changes to form values.
clear	Clears the <code>value Dictionary</code> .

For example:

```
<dsp:input bean="MultiProfileAddFormHandler.create" type="submit"
  value="Add New Users"/>
```

### ***Calling the clear operation automatically***

After a user submits profile data or deletes user profiles, the form handler automatically calls its `clear` operation, if the following properties are set:

- `MultiProfileAddFormHandler` has `clearValuesOnCreate` set to true.
- `MultiProfileUpdateFormHandler` has either `clearValuesOnDelete` or `clearValuesOnUpdate` set to true.

The `clear` operation clears the temporary `value Dictionary`. This prevents leftover keys from corrupting another form if the user enters new data. This method is useful for session-scoped form: the form can safely retain information across multiple pages in its `value Dictionary` for a given session, and ensure that the data is cleared on form submission. To call the `clear` operation on a form, insert this line immediately after the `dsp: form` tag:

```
<dsp:setvalue bean="MultiProfileAddFormHandler.clear" value="" />
```

## **Setting Values for Multiple Profiles**

Setting values for multiple profiles is similar to setting values for a single profile. Input fields on a registration page are used to set the form handler's `value Dictionary`, where data is stored pending a form submit operation such as `create` or `update`. On form submission, all values in the dictionary are written to the selected user profiles as a single transaction.

For detailed information about setting profile values, see [Setting Profile Values](#) earlier in this chapter.

### ***Setting values for individual profiles***

The `MultiProfileAddFormHandler` contains a `users` property, which is a list of all user profiles. This property provides access to the property values of individual profiles. The following example uses `ForEach` to iterate over all user profiles in `users`, and display text input fields for each one:






---

```
<dsp: dropl et name="/atg/dynamo/dropl et/ForEach">
  <dsp: param bean="Mul ti Profi l eAddFormHandl er. users" name="array" />
  <dsp: oparam name="output">
    <dsp: i nput bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. fi rstName"
      type="text" />
    <dsp: i nput bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. l astName"
      type="text" />
    <dsp: i nput bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. bi rthDate"
      type="text" />
    <dsp: i nput bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. phoneNumber"
      type="text" />
  </dsp: oparam>
</dsp: dropl et>
```

---

**Note:** The number of user profiles in users is set by the form handler's count and maxCount properties. The count property must be set in the handler before the form can be rendered. For more information, see the [count](#) property.

## MultiProfileAddFormHandler Properties

The default values for MultiProfileAddFormHandler properties are generally configured correctly for most situations. You can reset the properties of a MultiProfileAddFormHandler component in two ways:

- Reconfigure the properties in the MultiProfileAddFormHandler component. the configured values are used to initialize each new instance of the component.
- Override a component's configured value by setting the property value from the current page. A value set this way applies only to the instance of the MultiProfileAddFormHandler component that is associated with this page.

The effect of property settings depends on the scope of the MultiProfileAddFormHandler instance. For more information, see [Multi-Profile Form Handler Scope](#).

The Mul ti Profi l eAddFormHandl er also shares a number of properties with the ProfileFormHandler, which serve the same purpose. These include [tri mProperti es](#) and [confi rmPassword](#).

### ***clearValuesOnCreate***

This property uses the handl eCl ear method to clear the val ue Dictionary when a form creating users is submitted. This property is set to true by default.

### ***count***

An integer that represents the number of users to be processed by the form; serves as a dimension of the [users](#) list. The default value is 0.

This property must be set before the form is rendered on the page. For request-scoped form handler components, a hidden input tag must restore the state of the count property before the </dsp: form> close tag. For example:

---

```

<dsp: setValue bean="MultiProfileAddFormHandler.count" value="2" />
<body>
  <dsp: form action="MultiProfileAdd.jsp" method="post">
    ...
    ...
    <dsp: input type="hidden" bean="MultiProfileAddFormHandler.count" />
  </dsp: form>
</body>

```

---

### ***currentUser***

Contains the `UserListItem` object of the user currently being updated.

You can use this property in the `postCreateUser()` method when you need to extend the functionality of the `MultiProfileAddFormHandler`. See *Multiple ProfileFormHandlers* in the *Working with User Profiles* chapter of the [ATG Personalization Programming Guide](#) for more information.

### ***defaultPassword***

A `String` that contains the default password, set when the [generatePassword](#) property is set to true.

### ***extractDefaultValuesFromItem***

Identical to the `ProfileFormHandler` property [extractDefaultValuesFromProfile](#).

### ***generatePassword***

A `Boolean`, indicates whether to generate the password automatically when the form is submitted with the create operation. The default value is false. If set to true, the default password is set to the value of the [defaultPassword](#) property.

### ***maxCount***

The maximum value allowed for this form's [count](#) property. The default value is 0—that is, no maximum value. If set to a value greater than 0, `maxCount` always overrides the `count` property.

**Note:** `maxCount` must be set in the page before the `count` property.

### ***minCount***

The minimum value that the `count` property can be set to in this form. The default value for this property is 1. If `count` is less than `minCount`, and `minCount` is greater than 0, then `count` is set to `minCount`. You must set the `minCount` property in the page before you set the value of the `count` property. The `minCount` value must be greater than 0.

### ***profileTools***

This property contains a link to the `ProfileTools` component. The `MultiProfileAddFormHandler` uses this component to perform operations that are common to all `ProfileFormHandlers`. See *Using ProfileFormHandlers* in the *Working with Forms and Form Handlers* chapter of the [ATG Programming Guide](#).



### ***userListItemClass***

This String is the name of the class that the form uses to create the list of users to add. The default class is `UserListItem`. The `UserListItem` class is a subclass of `RepositoryFormHandler`. It provides a means to handle form exceptions for individual users. The `user` array instantiates this class so you can extend this class and pass added functionality to the objects in the user list.

### ***users***

A List of objects that contains the form update values for properties that apply to each user profile. For example, each user profile might have the properties `firstName`, `homeAddress`, and so on. For per-user property values, each object in the list contains a `value` Dictionary. This `value` Dictionary is used in the same way as the `value` Dictionary in the single `ProfileFormHandler`. It stores the values before the form is submitted.

## **Updating Values in Multiple Profiles**

The `MultiProfileUpdateFormHandler` updates or deletes sets of profiles. Because the profiles are items that already exist in the repository, each one is identified—for update or deletion—by its unique repository ID. These identifiers can be accessed in the handler's `repositoryIds` property.

Form data is stored in the `value` Dictionary, pending a form submit operation such as update or delete. On form submission, all values in the dictionary are written to the selected user profiles as a single transaction. To clear the `value` Dictionary after an update, invoke the handler's `clear` property at the beginning of a form:

```
<dsp: setvalue bean="MultipleProfileUpdateFormHandler.clear" value="" />
```

For more information about setting values in profiles, see [Setting Profile Values](#) earlier in this chapter.

## **MultiProfileUpdateFormHandler Properties**

The default values for `MultiProfileUpdateFormHandler` properties are generally configured correctly for most situations. You can reset the properties of a `MultiProfileUpdateFormHandler` component in two ways:

- Reconfigure the properties in the `MultiProfileUpdateFormHandler` component. the configured values are used to initialize each new instance of the component.
- Override a component's configured value by setting the property value from the current page. A value set this way applies only to the instance of the `MultiProfileUpdateFormHandler` component that is associated with this page.

The effect of property settings depends on the scope of the `MultiProfileUpdateFormHandler` instance. For more information, see [Multi-Profile Form Handler Scope](#).

The `MultiProfileUpdateFormHandler` also shares a number of properties with the `ProfileFormHandler`, which serve the same purpose. These include `trimProperties` and `confirmPassword`.

***clearValuesOnDelete***

A Boolean, clears the value Dictionary on submission of a form that deletes users. The default value is false.

***clearValuesOnUpdate***

A Boolean, clears the value Dictionary on submission of a form to update users. The default value is true.

***confirmOldPassword***

A Boolean, specifies whether the user is required to confirm an old password when changing the password property. The default value is false.

***extractDefaultValuesFromItem***

Identical to the ProfileFormHandler property [extractDefaultValuesFromProfile](#).

***generateProfileUpdateEvents***

A Boolean, specifies whether to send a JMS message when a profile is updated. The default value is false.

***profileTools***

Contains a link to the ProfileTools component. The MultiProfileUpdateFormHandler uses this component to perform operations that are common to all ProfileFormHandlers. See *Using ProfileFormHandlers* in the *Working With Forms and Form Handlers* chapter of the [ATG Programming Guide](#).

***profileUpdateEvent***

Holds the property update information when generating ProfileUpdateEvents.

***profileUpdateTrigger***

Generates the ProfileUpdateEvents when requested.

***repositoryIds***

A String array that represents the list of existing user profiles to update or delete. You must provide this list in order to update or delete existing repository profiles.

**Multi-Profile Form Handler Navigation Properties**

After a profile form operation—registration, login, or logout—is complete, you might want to redirect the user to a new page. As in standard HTML, you can use the `action` attribute of the form tag to specify a page to display on form submission.

The `action` attribute can specify only a single page. You might want to specify several different pages, where the user's destination depends on the nature of the form operation and whether or not it succeeds.

The MultiProfileUpdateFormHandler and MultiProfileAddFormHandler have a set of properties you can use to control navigation after a form operation. These properties specify the URLs where the user is



redirected on certain error and success conditions. Each operation except cancel has corresponding success and failure URL properties:

Handler operation	Success/failure URL properties
create	createSuccessURL createErrorURL
delete	deleteSuccessURL deleteErrorURL
update	updateSuccessURL updateErrorURL
cancel	cancel URL

The value of each property is a URL relative to the page specified by the form's `action` attribute or a local URL defined absolutely from the root. The default value of each property is null, so you must set these properties explicitly. You can specify the values of these URL properties in the form handler component, or with hidden input tags in the JSP. If these property values are not set, the action-specified page is displayed on form submission.

The following example sets the value of `createErrorURL`, redirecting the user to `create_failed.jsp` if the create operation fails. If the create operation succeeds and the value of `createSuccessURL` is not set, the user is redirected to the action-specified page `index.jsp`:

```
<dsp: form action="index.jsp" method="post">
  <dsp: input bean="MultiProfileAddFormHandler.createErrorURL"
    type="hidden" value="create_failed.jsp"/>
  ...
</dsp: form>
```

## Multi-Profile Form Handler Error Messages

Multi-profile form handlers use the ATG servlet bean `ProfileErrorMessageForEach` to display error messages. For more information about this servlet bean, see [Single Profile Form Handler Error Messages](#).

You can also access the `MultiProfileAddFormHandler.formExceptions` property to find a summary of exceptions, one for each type of exception that is generated during a submit operation. You can also obtain exceptions for each user profile through this syntax:

```
MultiProfileAddFormHandler.users[index].formExceptions
```

For example, the following code collects a summary of form errors:



---

```
<%-- To handle the summary list of errors - duplicate exceptions eliminated --%>
<dsp:droplet name="Switch">
  <dsp:param bean="MultiProfileAddFormHandler.formError" name="value"/>
  <dsp:oparam name="true">
    <dsp:droplet name="ProfileErrorMessageForEach">
      <dsp:param bean="MultiProfileAddFormHandler.formExceptions"
        name="exceptions"/>
      <dsp:oparam name="output">
        <dsp:valueof param="message"/>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

---

The following code collects errors generated by specific user profiles:

---

```
<%-- To see the errors generated for each user --%>
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
  <dsp:param bean="MultiProfileAddFormHandler.users" name="array"/>
  <dsp:droplet name="Switch">
    <dsp:param bean="MultiProfileAddFormHandler.users[param:index].formError"
      name="value"/>
    <dsp:oparam name="true">
      <dsp:droplet name="ProfileErrorMessageForEach">
        <dsp:param
          bean="MultiProfileAddFormHandler.users[param:index].formExceptions"
          name="exceptions"/>
        <dsp:oparam name="output">
          <dsp:valueof param="message"/>
        </dsp:oparam>
      </dsp:droplet>
    </dsp:oparam>
  </dsp:droplet>
</dsp:droplet>
```

---

## Multi-Profile Form Handler Scope

A multi-profile form handler component can be request- or session-scoped. The default scope is request: each time a user accesses a page with a profile form, the ATG platform creates an instance of the form handler that uses the configured values of the component's properties. You can override the values of these properties using hidden input tags, but the values set this way do not carry over if the user accesses another page with a profile form.

Property values can persist across multiple pages if the form handler is session-scoped. For example, a long registration form might split across two pages, with the second page containing the form's submit control. Navigation controls enable the user to move between the two pages before submitting the form.



In order to retain values from both pages, the form handler must be session scoped; otherwise, values from one page are lost when the user navigates to the other.

If you set a form handler to be session-scoped, be careful about using hidden input fields to override default property values. The new values remain in effect throughout the session unless explicitly changed. For example, a registration page sets `extractDefaultValuesFromItem` to false. If, during the session, the user subsequently accesses a page to update profile information, the fields in this form are not filled in with values from the profile unless that page explicitly sets `extractDefaultValuesFromItem` back to true.

## Profile Form Examples

The following sections provide three examples of profile forms, which perform these tasks:

- [Create a user profile.](#)
- [Create multiple user profiles](#)
- [Update multiple user profiles.](#)

### Create a User Profile

The following example defines a page that contains a registration form. Embedded comments explain how the form works.

---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>

<dsp: page>

<HTML>
<HEAD>
<TITLE>Registration Form</TITLE>
</HEAD>

<dsp:importbean bean="/atg/userprofiling/ProfileFormHandler"/>
<dsp:importbean bean="/atg/userprofiling/ProfileErrorMessageForEach"/>
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>

<!-- The next line sets the Profile Form Handler's extractDefaultValuesFromProfile
      property to false, to ensure that the fields in the form are blank when the
      user accesses the page. --%>
<dsp:setvalue bean="ProfileFormHandler.extractDefaultValuesFromProfile"
value="false"/>

<BODY>
```



```
<%-- The Switch bean checks whether the current profile is a transient profile.
      If it isn't, this means that the user is logged in and therefore cannot
      register, so a warning message is displayed. If the current profile is
      transient, the registration form is displayed. --%>
<dsp: droplet name="Switch">
  <dsp: param bean="ProfileFormHandler.profile.transient" name="value"/>
  <dsp: oparam name="false">
    You are currently logged in. If you wish to register as a new user
    please logout first.
  </dsp: oparam>

  <dsp: oparam name="default">

<%-- The next two lines set the action page for the form and the createSuccessURL
      page. The createErrorURL is not defined, so if there is an error, the user
      is directed to the action page. In this example, the action page is the
      same as the current page, so the page is redisplayed with the appropriate
      error messages. --%>
  <dsp: form action="register.jsp" method="post">
    <dsp: input bean="ProfileFormHandler.createSuccessURL" type="hidden"
      value="create_success.jsp"/>

<%-- This line sets the confirmPassword property to true, so the user is
      required to confirm the password entered. --%>
    <dsp: input bean="ProfileFormHandler.confirmPassword" type="hidden"
      value="true"/>

<h3>Register as a user:</h3>

<%-- The Switch bean checks the value of the formExceptions property. If the
      value of this property is true, there were errors when the form was
      submitted. The ProfileErrorMessageForEach bean cycles through the
      formExceptions array, and displays the message associated with
      each error. --%>
<dsp: droplet name="Switch">
  <dsp: param bean="ProfileFormHandler.formError" name="value"/>
  <dsp: oparam name="true">
    <UL>
      <dsp: droplet name="ProfileErrorMessageForEach">
        <dsp: param bean="ProfileFormHandler.formExceptions" name="exceptions"/>
        <dsp: oparam name="output">
          <LI> <dsp: valueof param="message"/>
        </dsp: oparam>
      </dsp: droplet>
    </UL>
  </dsp: oparam>
</dsp: droplet>

<%-- This hidden input tag sets the member property of the value Dictionary
      to true. When the form is submitted, the member property of the permanent
```





```

        profile is therefore set to true, indicating this user is a member rather
        than a guest. --%>
<dsp:input bean="ProfileFormHandler.value.member" type="hidden"
value="true"/>

<table width=456 border=0>

<%-- The next 30 or so lines create the actual fields the user fills in, and
      uses these fields to set properties in the Profile Form Handler's value
      Dictionary. When the form is submitted, the values in the value Dictionary
      are then copied to properties in the user's permanent profile. --%>
    <tr>
      <td valign=middle align=right>User Name: </td>
      <td><dsp:input bean="ProfileFormHandler.value.login" maxsize="20" size="20"
type="text"/></td>
    </tr>

    <tr>
      <td valign=middle align=right>Password: </td>
      <td><dsp:input bean="ProfileFormHandler.value.password" maxsize="35" size="35"
type="password"/></td>
    </tr>

    <%-- When the user submits the form, the Profile Form Handler checks
          whether the value in the confirmpassword field matches the value in
          the password field. If the values don't match, the Profile Form Handler
          produces an error, and does not create the new profile. --%>

    <tr>
      <td valign=middle align=right>Password Confirmation: </td>
      <td><dsp:input bean="ProfileFormHandler.value.confirmpassword" maxsize="35"
size="35" type="password"/></td>
    </tr>

    <tr>
      <td valign=middle align=right>Email Address: </td>
      <td><dsp:input bean="ProfileFormHandler.value.email" maxsize="30" size="30"
type="text"/></td>
    </tr>

    <tr>
      <td valign=middle align=right>Gender: </td>
      <td>
        <dsp:input bean="ProfileFormHandler.value.gender" type="radio"
value="male"/>Male
        <dsp:input bean="ProfileFormHandler.value.gender" type="radio"
value="female"/>Female
      </td>
    </tr>

```

```
<%-- This line invokes the Profile Form Handler's create operation, which creates
      a new permanent profile using the values in the value Dictionary. --%>
      <tr>
        <td valign=middle align=right></td>
        <td><dsp:input bean="ProfileFormHandler.create" type="submit"
value="Register"/></td>
      </tr>
    </table>

</dsp:form>
</dsp:oparam>
</dsp:droplet>

</BODY>
</HTML>
</dsp:page>
```

---

## Create Multiple User Profiles

The following example shows how to create multiple user profiles. In a real-life application, a preceding page contains the user count and that value is passed as a parameter to this page and set according by the <dsp:setvalue> function.

---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>

<dsp:page>
<HTML>
<HEAD>
<TITLE>Create Multiple Profile Forms</TITLE>
</HEAD>

<dsp:importbean bean="/atg/userprofiling/MultiProfileAddFormHandler"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>

<%-- The next line sets the form handler's count property to the number of users
to be added by this form. For demonstration purposes, the count value has been
hard-coded here. In reality, a preceding page would specify the user count and
pass it as a parameter to this page to be set accordingly using <dsp:setvalue
value="null"/>. --%>

<dsp:setvalue bean="MultiProfileAddFormHandler.count" value="2"/>

<BODY>

<%-- The next line sets the action page for this form to be the same
as the current page --%>
<dsp:form action="MultiProfileAdd.jsp" method="post">
```



```

<%-- The following hidden input tag makes sure that the count value is set and
carried over for a request scoped (the default) form handler. Placing it at the
beginning of the form ensures it will be set before the get is done on the users
array property which is where the allocation of the array is done --%>

<dsp:input bean="MultiProfileAddFormHandler.count" type="hidden"/>

<%-- The next two lines set the form redirection locations for success or failure.
--%>
<dsp:input bean="MultiProfileAddFormHandler.createSuccessURL" type="hidden"
value="MultiProfileUpdate.jsp"/>
<dsp:input bean="MultiProfileAddFormHandler.createErrorURL" type="hidden"
value="myAdderror.jsp"/>

<%-- The next two lines initialize the setting of a default password to be used
for each user created by this form. --%>
<dsp:input bean="MultiProfileAddFormHandler.generatePassword" type="hidden"
value="true"/>
<dsp:input bean="MultiProfileAddFormHandler.defaultPassword" type="hidden"
value="test"/>

<h3>Update per-user contact information:</h3>

<%-- Using a ForEach droplet, display the input tags for each per-user property to
be set by this form --%>
<dsp:droplet name="/atg/dynamo/droplet/ForEach">
<%-- Set the array parameter to the list of users to be added so that the
properties set will end up in the value Dictionary for the associated user. --%>
  <dsp:param bean="MultiProfileAddFormHandler.users" name="array"/>
  <dsp:oparam name="output">
    <UL>
      User: <dsp:valueof param="count"/>
    </UL>
    <table width=456 border=0>
      <tr>
        <%-- The following lines create the actual fields displayed on the form for
each profile being added. They are placed in a table here and repeated for the
number of users requested by this form. --%>
        <td valign=middle align=right>Login Name:</td>
        <td><dsp:input
bean="MultiProfileAddFormHandler.users[param:index].value.login" maxsize="30"
size="30" type="text"/></td>
      </tr>
      <tr>
        <td valign=middle align=right>First Name:</td>
        <td><dsp:input
bean="MultiProfileAddFormHandler.users[param:index].value.firstName" maxsize="30"
size="30" type="text"/></td>
      </tr>
      <tr>

```



```
<td val ign=mi ddle al ign=ri ght>Last Name: </td>
<td><dsp: input
bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. val ue. l astName" maxsi ze="30"
si ze="30" type="text"/></td>
</tr>
<tr>
<td val ign=mi ddle al ign=ri ght>Email : </td>
<td><dsp: input
bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. val ue. emai l " maxsi ze="30"
si ze="30" type="text"/></td>
</tr>
<tr>
<td val ign=mi ddle al ign=ri ght>Aggresi ve I ndex: </td>
<td><dsp: input
bean="Mul ti Profi l eAddFormHandl er. users[param: i ndex]. val ue. aggresi veI ndex"
maxsi ze="30" si ze="30" type="text"/></td>
</tr>
</table>
</dsp: oparam>
<dsp: oparam name="empty">
array is empty!
</dsp: oparam>
</dsp: dropl et>
```

<%-- The next section of lines contain the input tags for the property values that are shared by all the users being added by this form. Notice they are assigned to the "value" Dictionary in the same way as in using the ProfileFormHandler. --%>

<h3>Update common properties: </h3>

```
<table width=456 border=0>
<tr>
<td val ign=mi ddle al ign=ri ght>Local e: </td>
<td><dsp: input bean="Mul ti Profi l eAddFormHandl er. val ue. l ocal e" maxsi ze="30"
si ze="30" type="text"/></td>
</tr>
<tr>
<td val ign=mi ddle al ign=ri ght>Member (true/fal se): </td>
<td><dsp: input bean="Mul ti Profi l eAddFormHandl er. val ue. member" maxsi ze="30"
si ze="30" type="text"/></td>
</tr>
<tr>
<td val ign=mi ddle al ign=ri ght>User Type (i nvestor/broker/guest): </td>
<td><dsp: input bean="Mul ti Profi l eAddFormHandl er. val ue. userType" maxsi ze="30"
si ze="30" type="text"/></td>
</tr>
</table>
```

<%-- Display two different submit options - create and cancel which invoke methods in the Form Handler --%>

```
<dsp: input bean="Mul ti Profi l eAddFormHandl er. create" type="submi t" val ue="save"/>
<dsp: input bean="Mul ti Profi l eAddFormHandl er. cancel " type="submi t" val ue="cancel "/>
```



```

</dsp: form>
</BODY>
</HTML>
</dsp: page>

```

---

## Update Multiple User Profiles

The following example shows how to update multiple user profiles.

```

<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>

<%@ page import="atg.servlet.*"%>

<dsp: page>
<HTML>
<HEAD>
<TITLE>Update Contact Information Checklist</TITLE>
</HEAD>

<dsp:importbean bean="/atg/userprofiling/MultiProfileUpdateFormHandler"/>
<dsp:importbean bean="/atg/dynamo/droplet/RQLQueryForEach"/>

<BODY>
<!-- This page displays a list of users beginning with the letter 's', using the
RQLQueryForEach droplet - It allows you to check off the users you'd like to
update and offers a few property <dsp:input> tags for you to update the selected
user
information --%>

<!-- The next line sets the action page for this form to be the same as the
current page --%>
<dsp: form action="<%=ServletUtil.getDynamoRequest(request).getRequestURL()%"
method="post">

<!-- The next two lines set the form redirection locations for success or failure.
--%>
<dsp:input bean="MultiProfileUpdateFormHandler.updateSuccessURL" type="hidden"
value="MultiProfileDelete.jsp"/>
<dsp:input bean="MultiProfileUpdateFormHandler.updateErrorURL" type="hidden"
value="myUpdateError.jsp"/>

<h3>Select which users to update:</h3>

<!-- For testing purposes, this page finds users with firstname beginning with 's'
--%>

<dsp: param name="beginChar" value="s"/>
<!-- The following droplet displays the list of users with a checkbox for each

```



one. When boxes are checked, the repositoryId for that user is placed in the repositoryIds array property. --%>

```
<dsp: droplet name="/atg/dynamo/droplet/RQLQueryForEach">
  <dsp: param name="repository"
    value="/atg/userprofiling/ProfileAdapterRepository"/>
  <dsp: param name="itemDescriptor" value="user"/>
  <dsp: param bean="/atg/dynamo/transaction/TransactionManager"
    name="transactionManager"/>
  <dsp: param name="queryRQL" value="firstName STARTS WITH IGNORECASE
    : beginChar"/>
  <dsp: oparam name="outputStart">
    <table width=100% cellpadding=0 cellspacing=0 border=0>
  </dsp: oparam>
  <dsp: oparam name="output">
    <tr><td>
      <dsp: input bean="MultiProfileUpdateFormHandler.repositoryIds"
        paramvalue="element.repositoryId" type="checkbox"/>
      <dsp: valueof param="element.firstName"/>&nbsp;
      <dsp: valueof param="element.lastName"/>
    </td></tr>
  </dsp: oparam>

  <dsp: oparam name="outputEnd">
    </table>
  </dsp: oparam>

  <dsp: oparam name="error">
    *** Error occurred *** <p><p>
    <dsp: valueof param="rqlException"/>
  </dsp: oparam>
  <p>*** No results returned ***

</dsp: droplet>

<%-- This table displays the list of properties to be set for each of the checked
users --%>
<table border=0>
  <tr>
    <td valign=middle align=right>Aggressive Index: </td>
    <td><dsp: input bean="MultiProfileUpdateFormHandler.value.aggressivelindex"
      maxsize="30" size="30" type="text"/></td>
    </tr>
    <tr>
    <td valign=middle align=right>Goals: (short-term, retirement-focus)</td>
    <td><dsp: input bean="MultiProfileUpdateFormHandler.value.goals" maxsize="30"
      size="30" type="text"/></td>
    </tr>
    <tr>
    <td valign=middle align=right>Strategy: (moderate, aggressive)</td>
    <td><dsp: input bean="MultiProfileUpdateFormHandler.value.strategy">
```



```
maxsize="30" size="30" type="text"/></td>
</tr>
</table>

<dsp:input bean="MultiProfileUpdateFormHandler.update" type="submit"
value="update" />
<dsp:input bean="MultiProfileUpdateFormHandler.cancel" type="submit"
value="cancel" />
</dsp:form>
</BODY>
</HTML>
</dsp:page>
```

---







# 12 Displaying Repository Data in HTML Tables

Applications often need to display repository data in an HTML table. For example, you might want to display a table of user names and addresses, a table of authors and article titles, or a table of product information. `atg.service.util.TableInfo` is a class that you can use to create components that manage display of information in a table. A `TableInfo` component helps you create HTML tables whose rows correspond to JavaBeans, and whose columns display properties of each bean. You can also configure a `TableInfo` component to make tables sortable, so users can click on a column heading to sort the table on the column values.

A `TableInfo` component does not maintain or display the items rendered in the table. The list of items must be maintained in a separately managed component. Instead, `TableInfo` maintains information about table columns, such as column headings, the property to display in each column, and the sort criteria for each column. To render the table, you use the column information in the `TableInfo` component together with servlet beans that iterate over the list of items, such as `ForEach` and `TableForEach`.

This chapter includes the following sections:

- [TableInfo Basics](#)
- [Localizing Column Headings](#)
- [Managing Sort Criteria](#)
- [Simple Table Example](#)
- [User-Sorted Tables](#)
- [Adding Graphic Sort Indicators](#)
- [TableInfo Example](#)

## TableInfo Basics

In a simple configuration of a `TableInfo` component, the `columns` property specifies the name and corresponding property of each HTML table column. For example, the following `TableInfo.properties` file configures a `TableInfo` component to display user profile information.



---

```
$class=atg.service.util.TableInfo
$scope=session
```

```
columns=\
  Last Name=lastName, \
  First Name=firstName, \
  City=homeAddress.city, \
  Email Address=email
```

---

A TableInfo component assumes the existence of a separately managed list of JavaBeans: each columns expression names a bean property. For example, the previous example of a TableInfo configuration assumes the existence of a separately managed list of user profile objects with the following properties: lastName, firstName, homeAddress.city, and email. The TableInfo component, configured as described, yields an HTML table that looks like this:

Last Name	First Name	City	Email Address
Appleby	Mark	San Diego	mappleby@anywhere.net
Smith	Paula	San Francisco	paulas@myemail.com
Jones	Stephanie	San Francisco	founder@jonesnet.com
Smith	Albert	Los Angeles	albert.smith@mycompany.com

In general, each HTML table requires one session-scoped instance of TableInfo.

In most cases, column data is sorted on the property values that are displayed in column cells. Some exceptions can occur—for example, each cell in a table column might contain an HTML anchor tag that links to a news article. In this case, you should sort the column on the article title rather than the table cell contents of each.

When a column's display property and sort property are different, you can list both on the right side of the column name or specifier, separating the two property names by a semi-colon (;). In the example of the table of news articles, if the display property for a column is named anchorTag, but you want to sort the column on the title property, you specify the following:

---

```
columns=\
  Title=anchorTag ; title, \
  Author=author.lastname, \
  ...
```

---



## Localizing Column Headings

If your application supports multiple locales, you can localize table column headings with the `TableInfo.columnHeadings` property.

`TableInfo.columnHeadings` is a dictionary whose keys are locale-specific variants of a column name and whose values are the column headings to use for the corresponding locales. The following example shows how to use the `columnHeadings` property in a `TableInfo` configuration. The `TableInfo` component is configured to display a two-column table, where column headings are localized for English and French:

---

```
$class=atg.service.util.TableInfo
$scope=session

# Define the basic table layout using symbolic names for each column.

columnNames=\
    name=book.displayName, \
    price=book.listPrice

# Now create localized column headings for each column.

columnHeadings=\
    name=Title, \
    name_fr=Titre, \
    price=List Price, \
    price_fr=Prix Courant
```

---

To support additional locales, simply add their column headings to the `columnHeadings` list.

**Note:** This example ignores the country and variant parts of the French locale. You can include country- and variant-specific column headings by using entries such as `name_fr_FR=Prix`.

A `TableInfo` component computes the headings for each column by checking whether `TableInfo.columnHeadings` is set. If it is not set, then the column headings used in the table are the column names, as defined in `TableInfo.columnNames`.

If `TableInfo.columnHeadings` is set, the `TableInfo` determines the current locale in the following order of precedence:

1. Uses the locale specified in `TableInfo.userLocale`, if it is set.

Setting `TableInfo.userLocale` enables you to override the default or request locale, if desired. For example, you might want to copy the value of the user's preferred locale into `TableInfo` as follows:

```
<dsp:setvalue bean="/app/TableInfo.userLocale"
beanvalue="Profile.preferredLocale"/>
```

Or, you might want to force the page to display the table for the French locale with:

```
<dsp: setvalue bean="/app/TableInfo.userLocale" value="fr_FR"/>
```

2. Otherwise, uses `ServletUtil.getCurrentRequest` (class `atg.servlet.ServletUtil`) to identify the ATG servlet request being processed in the current thread. Then use the request's locale for the current locale.
3. If no current request exists or the request locale is not set, uses the module's default locale for the current locale.

When the current locale is determined, `TableInfo` examines the keys in the `TableInfo.columnHeadings` dictionary, looking for a locale-specific variant of the column name. It applies the same logic that the `java.util.ResourceBundle` class uses when looking for resource bundles. For example, if the column name is `price`, and the current locale is `fr_FR`, then `TableInfo` searches first for a key named `price_fr_FR`, next for a key named `price_fr`, and finally for a key named `price`. The value corresponding to the first matching key becomes the column's heading. If no matching key is found, then the column heading used in the table is the column name, as defined in `TableInfo.columns`.

By using a single `TableInfo` component that defines locale-specific column headings in `TableInfo.columnHeadings`, the ATG platform can automatically search for the best available match to the user's preferred language and locale. However, you can also localize table column headings by defining and configuring an instance of `TableInfo` for each desired locale. Your JSP can then refer to the appropriate instance depending on the language of the page or the user's locale. For example, an English-language page might include the following code to import a version of a given table with English column headings:

```
<dsp: importbean bean="/myapp/tables/en/ItemTable"/>
```

In turn, a German-language page might include the following code to import a version of the given table with German column headings:

```
<dsp: importbean bean="/myapp/tables/de/ItemTable"/>
```

Both pages can then refer to the `TableInfo` component as `bean="ItemTable"` and use its own locale-specific version of the component.

## Managing Sort Criteria

A `TableInfo` component supports sorting table data on one or more columns, in ascending or descending order. To support this functionality, the `TableInfo` maintains information about sort columns and their respective sort directions. A `TableInfo` also makes sorting instructions available to a form, where it can be used by standard ATG servlet beans such as `ForEach` and `TableForEach`.

A `TableInfo` constructs an array of `Column` objects from two properties:

- The column name and property information specified in `TableInfo.columns`
- The column heading information specified in `TableInfo.columnHeading`

The `TableInfo` stores this array of `Column` objects in `TableInfo.tableColumns`. Each `Column` object maintains information about itself in the following properties:



Column object property	Description
heading	Column display heading; read-only.  <b>Note:</b> See <a href="#">Localizing Column Headings</a> for information about how the column heading is determined.
name	Column name; read-only.
property	The JavaBean property to display in the column; read-only.
sortIndex	The zero-based index of the column within the list of columns that are being sorted. 0 if the column is the primary sort column. -1 if the column is not sorted.
sortPosition	The 1-based index of the column within the list of columns to sort. 1 if the column is the primary sort column. 0 if the column is not sorted.
sortDirection	Column sort direction, set to ascending, descending, or none.

As shown by the code examples in later sections, you use `TableInfo.tableColumns` to iterate over the array of `Column` objects in order to obtain each column's heading, name, and JavaBean property. You can also use JSP code to change the sort direction of a column or the sort order of columns in the table. When you do so, `TableInfo` automatically updates the sort properties for the affected `Column` objects. You can then render the table using only the primary sort column or all sort columns:

- To render the table with only the primary sort column, use the `TableInfo.primarySortString` property, which stores a string containing a single property associated with the table's primary sort column. A possible value might be `+name`, which sorts the items in the table on ascending values of `name`.
- To render the table using the complete set of sort columns, use the `TableInfo.sortString` property, which stores a string containing multiple properties associated with all of the table's sort columns. A possible value might be `+name, -price`, which first sorts the items in the table on ascending values of `name` and then on descending values of `price`.

**Note:** `TableInfo` automatically updates `primarySortString` and `sortString` when you use JSP code to change a column's sort direction or the columns' sort order. You never explicitly set either of these `TableInfo` properties. Instead, you use them to obtain the current sorting instructions for the table.

If no explicit sorting is specified for table columns, you can use the `defaultSortString` property to specify a default sort order. If no explicit sort directions are set for a table, a call to `getSortString` returns `defaultSortString`, and a call to `getPrimarySortString` returns the first comma-separated token from `defaultSortString`.

For example, if `defaultSortString` is set to the string `+name`, then calling either `getSortString` or `getPrimarySortString` returns `+name`. If `defaultSortString` is set to the string `+name, -price`, then calling `getSortString` returns `+name, -price`, but calling `getPrimarySortString` returns only `+name`.

The value of `defaultSortString` is used only when the `sortString` property or `primarySortString` property otherwise returns an empty string. Setting `defaultSortString` does not modify the sort direction of any columns. Instead, it specifies the default sorting instructions to use if all of the columns are currently unsorted.

## Simple Table Example

The following JSP code shows how to use a `TableInfo` component and nested `ForEach` servlet beans to render a simple table of user profile information. User profile information is stored in a parameter named `userProfiles`, which can be populated in several ways. For example, you might use a `TargetingArray` servlet bean to retrieve from the Profile Repository all user profiles that meet the targeting rules you specify—for example, all users who live in Boston and are registered members of your site. You can then set the value of `userProfiles` to the array of profiles that is the result of the targeting operation, and pass the `userProfiles` parameter into an included page that renders the profiles in a table using `TableInfo`:

---

```
<dsp:include page="build_table.jsp">
  <dsp:param name="userProfiles" param="element"/>
</dsp:include>
```

---

The following code example shows how to use a `TableInfo` component to render a simple table. Later sections show how to build more tables, where the user can change the columns on which the table is sorted.

---

```
1 <!-- Create table displaying profiles stored in param: userProfiles --%>
2 <!-- using table display information from a bean named TableInfo --%>
3
4 <dsp:droplet name="ForEach">
5   <dsp:param name="array" param="userProfiles"/>
6   <dsp:param bean="TableInfo.sortString" name="sortProperties"/>
7   <dsp:oparam name="empty">
8     There are no items in your list.
9   </dsp:oparam>
10
11 <!-- Generate table start and column headings in outputStart --%>
12
13   <dsp:oparam name="outputStart">
14     <TABLE border=1 cellspacing=2 cellpadding=2>
15       <TR>
16         <dsp:droplet name="ForEach">
17           <dsp:param bean="TableInfo.tableColumns" name="array"/>
```

---



```

18         <dsp: param name="sortProperties" value="" />
19         <dsp: oparam name="output">
20             <dsp: setvalue paramvalue="element.heading" 21 param="columnHeading" />
22             <TD align="center">
23                 <dsp: valueof param="columnHeading" />
24             </TD>
25         </dsp: oparam>
26     </dsp: droplet>
27 </TR>
28 </dsp: oparam>
29
30 <!-- Generate a row per item in output --%>
31
32     <dsp: oparam name="output">
33         <dsp: setvalue paramvalue="element" param="currentProfile" />
34         <TR>
35
36 <!--
37 Here you are going to iterate over all of the property names you
38 want to display in this row, using the BeanProperty servlet bean to
39 display each value.
40 --%>
41
42         <dsp: droplet name="ForEach">
43             <dsp: param bean="TableInfo.tableColumns" name="array" />
44             <dsp: param name="sortProperties" value="" />
45
46             <dsp: oparam name="output">
47                 <TD>
48
49                     <dsp: droplet name="BeanProperty">
50
51                         <dsp: param name="bean" param="currentProfile" />
52                         <dsp: param name="propertyName" param="element.property" />
53
54                         <dsp: oparam name="output">
55                             <dsp: valueof param="propertyValue" />
56                         </dsp: oparam>
57                     </dsp: droplet>
58
59                 </TD>
60             </dsp: oparam>
61         </dsp: droplet>
62     </TR>
63 </dsp: oparam>
64
65 <!-- Close the table in outputEnd --%>
66
67     <dsp: oparam name="outputEnd"></TABLE>
68 </dsp: oparam>

```



```
69
70 </dsp: droplet>
```

---

**Line 4**

The outer `ForEach` servlet bean on line 4 iterates over an array of profiles in `userProfiles`, passed into the servlet bean in line 5, and renders its output `oparam` once for each profile in the array.

**Line 6**

The current sorting instructions for the table are passed into the outer `ForEach` servlet bean using the `sortProperties` input parameter. In example, `sortProperties` is empty by default, so no columns in the table are sorted. Consequently, the profiles in `userProfiles` are rendered in the table in the order they are listed in the `userProfiles` array.

In many instances, this behavior is sufficient because array items are already sorted. For example, if a `TargetingArray` servlet bean is used to populate the `userProfiles` array, you can specify the sort order for the array set as the result of the targeting operation.

The use of `sortString` enables the table to be sorted on multiple columns. To sort a table on only a single column, render the table with `TableInfo.primarySortString`.

**Lines 16-26**

An inner `ForEach` servlet bean is embedded in the output `start oparam` of the outer `ForEach`, and it builds the column headings for the table. The array of `ColumnInfo` objects, maintained in `TableInfo.tableColumns`, is passed into the bean in the array input parameter. During each iteration over this array, the `columnName` parameter is set to the value of the current column's heading (lines 20-21), then rendered in a table cell in the heading table row (line 22-24).

Explicitly setting a `columnName` parameter in lines 20-21 is unnecessary; you can produce the same result with `<dsp: valueof param="element.heading" />`. However, copying the value in this explicitly-named parameter makes the JSP code clearer, particularly in the more complex examples shown later.

**Lines 42-57**

A second inner `ForEach` servlet bean is used in the output `oparam` of the outer `ForEach` servlet bean. It generates a table row for each item in `param: userProfiles`. Each row must display a single item's values for each property associated with a table column.

Each row is built as follows:

- The array of `ColumnInfo` objects (maintained in `TableInfo.tableColumns`) is passed into the `ForEach` servlet bean in its array input parameter (line 42).
- During each iteration through this array, the property expression for the current column is retrieved using the `BeanProperty` servlet bean nested in the output `oparam`. First, the current profile is passed into `BeanProperty` in the bean input parameter. Next, the current property for the current `ColumnInfo` object is passed into `BeanProperty` in the `propertyName` input parameter.





- BeanProperty sets its propertyValue output parameter equal to the value of the current column property for the current profile.
- The value is rendered in a table cell.

Each table cell in the current table row is constructed in this way as the application iterates over the tableColumns array and renders the output (a table cell) once for each Column object in the array. Likewise, each table row in the table is constructed as the application iterates over the param: userProfilees array and renders the output (that is, a table row) once for each profile in the array.

In lines 18 and 44, sortProperties is set to an empty string in both inner ForEach servlet beans, because they are nested inside the outer ForEach servlet bean; consequently, the sortProperties input parameter of the outer bean is visible to both inner beans. Because the sorting instructions should be applied only to the output rendered in the outer ForEach servlet bean, sortProperties must be overridden within the scope of each inner servlet bean.

## User-Sorted Tables

A table might be sorted by a single column (the primary sort column) or multiple columns. If a table is sorted by multiple columns, then the column by which the items are first sorted is the primary sort column.

You can enable users to change the primary sort column of a table by adding clickable column headings that redisplay the page, set the clicked column as the primary sort column, and set the column's sortDirection to a direction type you specify.

When you specify a column's sortDirection, you can specify either a literal value (ascending, descending, or none) or a semantic value, such as toggle. The following table describes the sort direction types you can use to specify a column's sort direction.

**Note:** When you specify a semantic value, TableInfo examines the column's current sort direction and computes a new literal value based on the sort direction type you specify.

sortDirection setting	Description
ascending	Sorts the column by ascending values.
descending	Sorts the column by descending values.
none	The column is not sorted.
toggle	Toggles the column's sort direction between ascending and descending.



sortDirection setting	Description
toggle_if_primary	If the column is the primary sort column for the table, toggles the column's sort direction between ascending and descending. Otherwise, makes the column the primary sort column and sets its sort direction to ascending.
cycle_up	Cycles the column's sort direction from none to ascending to descending and back to none again.
cycle_up_if_primary	If the column is the primary sort column for the table, cycles the column's sort direction from none to ascending to descending and back to none again. Otherwise, makes the column the primary sort column and sets its sort direction to ascending.
cycle_down	Cycles the column's sort direction from none to descending to ascending and back to none again.
cycle_down_if_primary	If the column is the primary sort column for the table, cycles the column's sort direction from none to descending to ascending and back to none again. Otherwise, makes the column the primary sort column and sets its sort direction to descending.

The earlier example uses `TableInfo` to render a simple table. You can modify this code, so users can click on any column heading and sort the table by the property associated with that column, thereby setting the primary sort column. The following sections show how to implement these changes in two steps:

1. [Modify the `outputStart` oparam.](#)
2. [Get the `sortColumn` parameter value.](#)

### **Modify the `outputStart` oparam**

Modify the `outputStart` oparam so each column heading links back to the current page and passes back to the page its column index as the value of a parameter named `sortColumn`.

```
<dsp:oparam name="outputStart">
  <TABLE border=1 cellspacing=2 cellpadding=2>
  <TR>
  <dsp:droplet name="ForEach">
    <dsp:param bean="TableInfo.tableColumns" name="array"/>
    <dsp:param name="sortProperties" value=""/>
    <dsp:oparam name="output">
      <dsp:setvalue paramvalue="element.heading" param="columnHeading"/>
      <TD align="center">
        <dsp:a href="table.jsp">
          <dsp:param name="sortColumn" param="index"/>
          <dsp:valueof param="columnHeading"/>
        </dsp:a>
      </TD>
    </dsp:oparam>
  </dsp:droplet>
  </TR>
  </TABLE>
</dsp:oparam>
```



```

        </dsp: a>
    </TD>
</dsp: oparam>
</dsp: droplet>
</TR>
</dsp: oparam>

```

When a user clicks on a column heading, the column's index is passed back to the page in the `sortColumn` parameter. The code you add in step 2 checks for a value for this parameter.

### ***Get the sortColumn parameter value***

Add code to the beginning of the page that receives the `sortColumn` parameter, checks the parameter for a value, and sets the primary sort column accordingly if a value exists.

The code should set the `sortDirection` for the column whose index is passed in to the page, to a direction type that makes the column the primary sort column (if it is not already). Direction types that set a column as the primary sort column are `toggle_primary`, `cycle_up_primary`, and `cycle_down_primary`. If you specify a direction type other than one of these, the sort direction of the clicked column changes accordingly, but the column does not become the primary column on which table items are sorted. (Refer to the table previously provided in this section for more information on direction types.)

```

<dsp: droplet name="IsNull">
    <dsp: param name="value" param="sortColumn" />

    <dsp: oparam name="false">
        <dsp: setvalue
bean="TableInfo. tableColumns[param: sortColumn]. sortDirection"
value="toggle_primary" />
    </dsp: oparam>

    <dsp: oparam name="true">
        <dsp: setvalue bean="TableInfo. reset" value="{null}" />
    </oparam/>

</dsp: droplet>

```

When the page is called, the application checks for a value for `sortColumn`. One of two processes occurs, depending on whether `sortColumn` has a value:

**sortColumn has no value:** The sorting instructions in `TableInfo` are reset. The `sortDirection` property for each column is set to none, and both `TableInfo.sortString` and `TableInfo.primarySortString` are cleared, which causes the table to be rendered unsorted. Consequently, the items in the table are then rendered in the order they are listed in the object that maintains them.

**sortColumn has a value:** The value is the index of the column by which to sort the items in the table, and the sort direction for the respective column is set to the specified direction type. In this example, the specified direction type is `toggle_primary`, which either toggles the column's sort direction if the

column is already the primary sort column, or makes the column the primary sort column (if it is not already) and sets the sort direction to the default sort direction (normally ascending) for the `TableInfo` component. The table is then rendered according to the current sorting instructions.

For example, a table displaying the model, price, and color of a list of automobiles in that order. The user clicks on the column heading of the first column, and the page sets the value of `TableInfo.tableColumns[0].sortDirection` to `toggle_primary`. The `TableInfo` component automatically sets both `sortString` and `primarySortString` to `+model`, indicating that the table should be sorted on ascending values of model.

**Note:** `primarySortString` is always equal to the first term in `sortString`.

Later, the user clicks on the column heading of the second column, and the page sets the value of `TableInfo.tableColumns[1].sortDirection` to `cycle_down_if_primary`. The `TableInfo` component automatically prepends the second column's sort property to `sortString`, and sets it as the new value of `primarySortString`. Now, `sortString` is `-price, +model`, and `primarySortString` is `-price`.

Note that if the page sets the value of `TableInfo.tableColumns[1].sortDirection` to `descending`, the sort direction of the column changes *in place*. That is, the column does not become the primary sort column. `sortString` is `+model, -price`, and `primarySortString` remains `+model`.

By using `setValue` and hidden input fields to change the `sortDirection` property of a column, you can manipulate both when the column is sorted against the other columns in the table and the sort direction of the column.

## Adding Graphic Sort Indicators

To enhance the usability of a sortable table, you can provide visual cues that indicate which column is the current primary sort column for the table.

The following code example identifies and displays the column heading of the primary sort column followed by `[+]`, `[-]`, or no indicator, depending on the current sort direction for the column (ascending, descending, or none, respectively). To add this kind of graphic indicator, make the following modifications to the `outputStartOpParam`:

1. As you render the column heading for the current column, retrieve the sort position and sort direction for the column and set `sortPosition` and `sortDirection`, respectively, equal to their values.
2. Using the information obtained in step 1, check if the `sortPosition` of the current column is 1. A `sortPosition` of 1 indicates the column is the primary sort column in the table. Display only the column heading if it is not the primary sort column.
3. If the current column is the primary sort column, use the `Switch` servlet bean to display the column heading followed by the graphic indicator appropriate for the column's current `sortDirection`.

The following JSP code shows to add graphic sort indicators to the column headings:




---

```

<dsp:oparam name="outputStart">
  <TABLE border=1 cellspacing=2 cellpadding=2>
  <TR>
    <dsp:droplet name="ForEach">
      <dsp:param bean="TableInfo.tableColumns" name="array"/>
      <dsp:param name="sortProperties" value="" />

      <dsp:oparam name="output">
        <dsp:setvalue paramvalue="element.heading" param="columnHeading" />
        <dsp:setvalue paramvalue="element.sortPosition" param="sortPosition" />
        <dsp:setvalue paramvalue="element.sortDirection" param="sortDirection" />
        <TD align="center">
          <dsp:a href="table.jsp">
            <dsp:param name="sortColumn" param="index" />

          <dsp:droplet name="Switch">
            <dsp:param name="value" param="sortPosition" />

            <dsp:oparam name="1">
              <dsp:droplet name="Switch">
                <dsp:param name="value" param="sortDirection" />

                <dsp:oparam name="ascending">
                  <dsp:valueof param="columnHeading" />&nbsp;  [+]
                </dsp:oparam>

                <dsp:oparam name="descending">
                  <dsp:valueof param="columnHeading" />&nbsp;  [-]
                </dsp:oparam>

                <dsp:oparam name="none">
                  <dsp:valueof param="columnHeading" />
                </dsp:oparam>
              </dsp:droplet>
            </dsp:oparam>

            <dsp:oparam name="default">
              <dsp:valueof param="columnHeading" />
            </dsp:oparam>
          </dsp:droplet>
        </dsp:a>
      </TD>
    </dsp:oparam>
  </dsp:droplet>
</TR>
</dsp:oparam>

```

---



## TableInfo Example

The JSP code fragment provided in this section can be used to build a sortable table from any collection of objects. This code example includes all of the functionality described in each of the previous sections about TableInfo.

---

```
<%-- Include this fragment to generate a sortable table.      --%>
<%-- tableItems is the array of collection of items to display.  --%>
<%-- tableInfo is the TableInfo object that describes the columns. --%>
<%-- tablePage is the jsp that contains the table.            --%>

<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<dsp:page>
<HTML>
<BODY>

<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
<dsp:importbean bean="/atg/dynamo/droplet/Format"/>
<dsp:importbean bean="/atg/dynamo/droplet/IsNull"/>
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>

<%--
    Handle updating sort directives if you were passed a sortColumn parameter
    and an optional sort direction in a sortDir parameter.
--%>

<dsp:droplet name="IsNull">
    <dsp:param name="value" param="sortColumn"/>

    <%-- If you don't have a sortColumn parameter, reset the sort info --%>

    <dsp:oparam name="true">
        <dsp:setvalue param="tableInfo.reset" value="{null}"/>
    </dsp:oparam>

    <%-- If you have a sortColumn parameter, it will be the index of a --%>
    <%-- column in the TableInfo object. Set its sort direction now. --%>

    <dsp:oparam name="false">

    <%-- Look to see if you got the optional sortDir parameter too. --%>
    <%-- If so, use it, else default to cycle_up_if_primary. --%>

    <dsp:droplet name="IsNull">
        <dsp:param name="value" param="sortDir"/>
```



```

        <dsp:oparam name="true">
            <dsp:setvalue
                param="tableInfo.tableColumns[param:sortColumn].sortDirection"
                value="cycle_up_if_primary"/>
        </dsp:oparam>

        <dsp:oparam name="false">
            <dsp:setvalue
                param="tableInfo.tableColumns[param:sortColumn].sortDirection"
                paramvalue="sortDir"/>
        </dsp:oparam>
    </dsp:droplet>

</dsp:oparam>
</dsp:droplet>

<%--
    Now you've dealt with the case where you got to this page by clicking on a
    column heading to re-sort and redisplay the table, so you can go ahead and build
    the table display using the updated sort parameters.
--%>

<dsp:droplet name="ForEach">
    <dsp:param param="tableItems" name="array"/>
    <dsp:param name="sortProperties" param="tableInfo.sortString"/>

    <dsp:oparam name="empty">
        There are no items in your list.
    </dsp:oparam>

    <%-- Generate column headings. Each heading is a link. Clicking
    <%-- on the link makes that column the primary sort column, cycling
    <%-- through sort directions if it was already the primary sort
    <%--column.
--%>

    <dsp:oparam name="outputStart">
        <TABLE border=1 cellspacing=2 cellpadding=2>
        <TR>
        <dsp:droplet name="ForEach">
            <dsp:param name="array" param="tableInfo.tableColumns"/>
            <dsp:param name="sortProperties" value=""/>

            <dsp:oparam name="output">
                <%-- Extract useful properties from the current Column descriptor. --%>
                <dsp:setvalue paramvalue="element.heading" param="columnHeading"/>
                <dsp:setvalue paramvalue="element.sortPosition" param="sortPosition"/>
                <dsp:setvalue paramvalue="element.sortDirection" param="sortDirection"/>
                <TD align="center"><B>
                    <dsp:getvalueof var="a92" param="tablePage"
                    vartype="java.lang.String">

```



```
<dsp: a href="{a92}">
  <dsp: param name="sortColumn" param="index"/>

<!-- If the current column is the primary sort column, --%>
<!-- you want to display + or - as a directional indicator to show --%>
<!-- the sort direction for that column. --%>

<dsp: droplet name="Switch">
  <dsp: param name="value" param="sortPosition"/>

  <!-- Primary sort column gets special treatment --%>
  <dsp: oparam name="1">
    <dsp: droplet name="Switch">
      <dsp: param name="value" param="sortDirection"/>
      <dsp: oparam name="ascending">
        <dsp: valueof param="columnHeading"/>&nbsp; [+]
      </dsp: oparam>
      <dsp: oparam name="descending">
        <dsp: valueof param="columnHeading"/>&nbsp; [-]
      </dsp: oparam>
      <dsp: oparam name="null">
        <dsp: valueof param="columnHeading"/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>

  <!-- Others just display the column name --%>
  <dsp: oparam name="default">
    <dsp: valueof param="columnHeading"/>
  </dsp: oparam>
</dsp: droplet>
</dsp: a>
</dsp: getvalueof>
</TD>
</dsp: oparam>
</dsp: droplet>
</TR>
</dsp: oparam>

<!-- Generate a row per item --%>

<dsp: oparam name="output">
  <dsp: setvalue paramvalue="element" param="currentItem"/>
  <TR>
    <!-- Here you are going to iterate over all of the property names
         you want to display in this row, using the bean Property
         servlet bean to display each value.
    --%>

    <dsp: droplet name="ForEach">
```





```

<dsp: param name="array" param="tableInfo.tableColumns"/>
<dsp: param name="sortProperties" value=""/>
<dsp: oparam name="output">
  <TD>
    <dsp: droplet name="BeanProperty">
      <dsp: param name="bean" param="currentItem"/>
      <dsp: param name="propertyName" param="element.property"/>
      <dsp: oparam name="output">
        <dsp: valueof param="propertyValue"/>
      </dsp: oparam>
    </dsp: droplet>
  </TD>
</dsp: oparam>
</dsp: droplet>

</TR>
</dsp: oparam>

<!-- Close the table -->

<dsp: oparam name="outputEnd">
  </TABLE>
  <p>
    <dsp: getvalueof var="a197" param="tablePage" vartype="java.lang.String">
      <dsp: a href="{a197}">Reset sort properties</dsp: a>
    </dsp: getvalueof>
  </dsp: oparam>
</dsp: droplet>
</BODY>
</HTML>
</dsp: page>

```





# Appendix A: DSP Tag Libraries

The DSP tag library lets you work with Nucleus components and other dynamic elements in your JSPs. DSP library tags support both runtime expressions, such as references to scripting variables, and the JSTL Expression Language (EL) elements, which also evaluated at runtime.

The table below lists alphabetically the tags that are implemented in the DSP tag libraries. This appendix combines these tag libraries into one discussion by describing all relevant attributes and properties for each tag.

This section is followed by a complete discussion of each tag, its attributes and, when applicable, its variables and properties.

- Attributes accept values used to set properties on a tag. For example, the `var` attribute names an EL variable that references the object of this tag. Other tags can use EL to reference the object's properties.
- Variables provide a way to set dynamic values with runtime expressions.
- Properties refer to properties of the object referenced by the EL variable. Other tags can use properties to access content in that object.

Finally, an example and corresponding explanation are provided for each tag.

DSP tag names and attributes that have equivalent names in HTML, JSP, or WML typically replicate the functionality of the corresponding tag while providing extra functionality. One exception applies: the `iclass` attribute is used in place of the `class` attribute to avoid using a standard method name in Java code. The discussions that follow omit explanations for non-required attributes that provide expected functionality, such as `onmouseover` and other custom event attributes. All such attributes are specified in the TLD with lowercase characters.

See the DSP tag library `tld` files for a listing of each tag and its attributes:

```
<ATG10dir>/DAS/taglib/dspjspTaglib/1.0/tld.
```

## Summary of DSP tags

DSP tag	Summary description
<code>dsp:a</code>	Supports passing values to component properties and page parameters on click-through. Also handles URL rewriting.
<code>dsp:beginTransaction</code>	Starts a transaction.



DSP tag	Summary description
<a href="#">dsp: commitTransaction</a>	Completes a transaction.
<a href="#">dsp: contains</a>	Determines whether a Collection contains a specific single valued item.
<a href="#">dsp: containsEJB</a>	Determines whether a Collection contains an EJB.
<a href="#">dsp: demarcateTransaction</a>	Manages a transaction by starting the transaction, checking for errors, rolling back the transaction when errors are found and committing it when they are not.
<a href="#">dsp: droplet</a>	Invokes an ATG Servlet Bean.
<a href="#">dsp: equalEJB</a>	Determines whether two EJBs have the same primary keys.
<a href="#">dsp: form</a>	Encloses a form that can send DSP form events.
<a href="#">dsp: frame</a>	Embeds a page by encoding the frame src URL.
<a href="#">dsp: getvalueof</a>	Creates an EL variable that references the specified component property or page parameter.
<a href="#">dsp: go</a>	Encloses a form that can send WML form events.
<a href="#">dsp: iframe</a>	Embeds a dynamic page, by encoding the frame src URL.
<a href="#">dsp: img</a>	Inserts an image.
<a href="#">dsp: importbean</a>	Imports a Nucleus component into a page so it can be referred to without using its entire pathname. Also, creates a reference to a Nucleus component in an attribute visible to EL expressions.
<a href="#">dsp: include</a>	Embeds a page into another page.
<a href="#">dsp: input</a>	Passes values to a component property on submission.
<a href="#">dsp: link</a>	References a page, such as a stylesheet, by encoding the link src URLs.
<a href="#">dsp: oparam</a>	Specifies content to be rendered by an enclosing <a href="#">dsp: droplet</a> tag.
<a href="#">dsp: option</a>	Specifies content to be rendered by an enclosing <a href="#">dsp: select</a> tag.
<a href="#">dsp: orderBy</a>	Provides the sorting pattern to parent tag <a href="#">dsp: sort</a> .
<a href="#">dsp: page</a>	Enables ATG page processing functionality.
<a href="#">dsp: param</a>	Stores a value in a parameter.
<a href="#">dsp: postfield</a>	Passes values to a component property on submission. (WML)



DSP tag	Summary description
<a href="#">dsp: property</a>	Sets a component property from <a href="#">dsp: a</a> tag.
<a href="#">dsp: rol l backTransacti on</a>	Causes any actions in the current transaction to be returned to their pretransaction state.
<a href="#">dsp: sel ect</a>	Passes values to a component property on submission.
<a href="#">dsp: setTransacti onRol l backOnl y</a>	Specifies that, when a transaction is prompted to end, it ends in a rollback action.
<a href="#">dsp: setval ue</a>	Sets the value of a component property or a page parameter to a specified value.
<a href="#">dsp: setxml</a>	Sets attribute display format to XML or HTML.
<a href="#">dsp: sort</a>	Organizes the contents of a Container or array based on a sorting pattern.
<a href="#">dsp: test</a>	Makes an object's descriptive information available so other tags can find out its size, data type, and so on.
<a href="#">dsp: textarea</a>	Passes values to a component property on submission.
<a href="#">dsp: tomap</a>	Introduces a page parameter, standard JavaBean or Dynamic Bean component, or constant value as an element in a JSP that other tags can render using EL.
<a href="#">dsp: transacti onStatus</a>	Reads the current transaction's status.
<a href="#">dsp: val ueof</a>	Retrieves and displays the value of a page parameter, component property, or constant value.

## dsp:a

Defines a link to a target destination.

```

<dsp: a link-destination link-text
    [param-list]
    [property-settings]
    [anchori d="i d"]
    [requi resSessi onConfi rmati on="{ true| fal se}"]
</dsp: a>

```



## Attributes

### *link-destination*

Specifies the link destination with one of the following attributes:

Attribute	Description
href	Set to path and file name. Resolves relative paths using the current page as a starting point. Resolves absolute paths using the Web server doc root as a starting point.
page	Set to path and file name. Resolves relative paths using the current page as a starting point. Resolves absolute paths using the Web application root as a starting point by prepending the request object's context root to the specified value.

### *param-list*

One or more embedded [dsp: param](#) tags that specify the parameters to pass to the link destination. For example:

```
<dsp: a href="homepage.jsp">Go to your home page
  <dsp: param name="city" beanvalue="Profile.homeAddress.city"/>
</dsp: a>
```

### *property-settings*

Specifies settings for one or more bean properties. To set a single property, use the following syntax:

```
bean="property-spec" source-spec
```

In order to set multiple properties within the same `dsp: a` tag, embed multiple `dsp: property` tags as follows:

```
<dsp: property bean="property-spec" source-spec />
<dsp: property bean="property-spec" source-spec />
...
```

In both cases, set *property-spec* and *source-spec* as follows:

- property-spec* specifies the property to update: its Nucleus path, component name, and property name. The property specification can also include a tag converter, such as [date](#) or [null](#). For more information, see [Tag Converters](#) in Chapter 2.
- source-spec* specifies the source value in one of the following ways:



- `beanval ue="property-spec"`

Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:

```
bean="Student_01. firstClass" beanval ue="FreshmanScience.name"
```

- `paramval ue="param-name"`

Specifies the value from the page parameter *param-name*. For example:

```
bean="Student_01.name" paramval ue="name"
```

- `val ue="value"`

Specifies a static value. For example:

```
bean="Student_01.enrolled" val ue="true"
```

**Note:** The same `dsp:a` tag cannot use both methods to set properties.

### ***anchorid***

Assigns a page-unique identifier to the anchor. Property-setting anchors require unique IDs when a page specifies multiple anchors and does not consistently display all of them. By providing an anchor ID, you can reliably reference this anchor.

### ***requiresSessionConfirmation***

Set to true in order to prevent cross-site attacks. If set to true, activation of this `dsp:a` tag supplies the request parameter `_dynSessConf`, which contains a randomly generated session long number that is associated with the current session. On receiving this request, the request-handling pipeline validates `_dynSessConf` against the current session's confirmation number. If it detects a mismatch or missing number, it blocks further request processing.

## **Usage Notes**

Like an HTML anchor tag, `dsp:a` defines a link with a target destination. `dsp:a` can also update a servlet bean property value and pass a page parameter to the destination page.

**Note:** `dsp:a` uses the `iclass` attribute in place of the cascading stylesheet class attribute to avoid using a Java reserved word in Java code.

## **Example**

---

```
<dsp:a href="index.jsp" bean="/samples/Student_01.name"
paramval ue="userName">Click here to set the name</dsp:a>
```

---

In this example, the text `Click here to set the name` is a link. When clicked, the ATG platform sets the name property of the `Student_01` component to the value held by the `userName` page parameter and opens `index.jsp`.



## dsp:beginTransaction

Starts a transaction and tracks whether the transaction is created successfully.

```
<dsp:beginTransaction var-spec>
...
</dsp:beginTransaction>
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>session</code> , or <code>application</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
id	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has two properties:

- `success`: A Boolean property, specifies the status of the start transaction operation. A call to `isSuccess` obtains the property's setting.
- `exception`: Identifies the `Throwable` exception object produced when a transaction does not start properly. A call to `getException` returns the exception object.

### Example

```
<dsp:beginTransaction var="beginXA"/>
<c:choose>
  <c:when test="{beginXA.success}">
    Transaction started <br/>
  </c:when>
  <c:otherwise>
    Transaction failed due to this exception:
    <c:out value="{beginXA.exception}"/>
  </c:otherwise>
</c:choose>
```





In this example, the tag causes a transaction to begin. If the call is successful, the user sees the message `Transaction started`. If the call fails, the user sees the message `Transaction failed due to this exception` and the exception string.

## dsp:commitTransaction

Triggers execution of all tasks encapsulated by the current transaction.

---

```
<dsp:commitTransaction var-spec>
  ...
</dsp:commitTransaction>
```

---

### Attributes

#### *var-spec*

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
<code>var</code>	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>session</code> , or <code>application</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
<code>id</code>	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has two properties:

- `success`: A Boolean property, specifies whether the commit operation is successful. A call to `isSuccess` obtains the property's setting.
- `exception`: Identifies the `Throwable` exception object produced when the commit operation fails. A call to `getException` returns the exception object.

### Example

---

```
<dsp:commitTransaction var="CommitXA"/>
  <c:choose>
    <c:when test="{CommitXA.success}">
      The transaction committed successfully<br/>
    </c:when>
    <c:otherwise>
```



```
Transaction failed for the following reason: <br/>
  <c: out value="{Commi tXA. excepti on}"/>
</c: otherwi se>
</c: choose>
```

---

The `dsp: commi tTransacti on` tag causes the transaction to commit. If the commit operation succeeds, the user sees the message `The transacti on commi tted successful ly`. If the transaction fails to commit, the user sees the message `Transacti on fai led for the fol lowi ng reason: and the excepti on string`.

## dsp:contains

Checks whether a Collection contains the specified value.

---

```
<dsp: contain s var-spec
  val ues=" col l ecti on"
  type=" search-val ue"
  [compareEl ement="{ true | fal se}"]
  [comparator=" comparator-obj" ]
/>
```

---

### Attributes

#### *var-spec*

An EL variable or scripting variable that captures the Boolean result of the operation, defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>sessi on</code> , or <code>appl i cati on</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
i d	Names a scripting variable that scriptlets and expressions can access at runtime.

#### *values*

Specifies the Collection to search. This attribute can specify items of the following data types: primitive array, Object array, Enumeration, Iterator, Collection, and Map.

**type**

Identifies the data type of the search value, one of the following:

boolean  
byte  
char  
double  
float  
int  
long  
object  
short

Set this attribute to the search value.

**compareElement**

If the `values` attribute is set to a Map, `compareElement` specifies how to evaluate the Map contents:

- `false` (default): Search all Map keys.
- `true`: Search all Map values.

**comparator**

Set to an instance of `java.util.Comparator` that you define. Use a comparator to compare specific portions of one element to another. If no comparator is specified, the entire first item is compared to a portion of the second.

**Usage Notes**

`dsp:contains` checks whether a given value is contained within the specified Collection. When you use this tag, you set the single value to an attribute named for that value's data type. You then specify the Collection that you want to search in the `values` attribute.

Alternatively, you can construct a complex operator that compares some portion of the two items. To do this, define a subclass of `java.util.Comparator` that identifies the characteristics to compare.

To find an EJB in a Collection of EJBs, use [dsp:containsEJB](#).

**Example**


---

```
<dsp:tomap bean="Math101Students" var="students"/>
<dsp:tomap bean="Profile" var="profile"/>

<dsp:contains var="enrolled" values="{students}" object="{profile.getId}"/>
  <c:choose>
    <c:when test="{enrolled}">
      Welcome, <c:out value="{profile.firstName}"/>. You're
```



```
        enrolled in Math 101!  
    </c: when>  
</c: choose>
```

In this example, if the specified user profile ID matches one of the IDs in `students`, the user's first name is retrieved and the specified message displays.

## dsp:containsEJB

Checks whether a `Collection` contains the specified EJB.

```
<dsp:containsEJB var-spec  
    values="collection"  
    ejb="search-ejb"  
>
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that captures the Boolean result of the operation, defined with one of these attributes:

Attribute	Description
<code>var</code>	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>session</code> , or <code>application</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
<code>id</code>	Names a scripting variable that scriptlets and expressions can access at runtime.

#### *values*

Specifies the `Collection` to search for the specified EJB. This attribute can specify items of the following data types: `array`, `Enumeration`, `Iterator`, `Collection`, and `Map`. If `values` is set to `Map`, the `Map`'s key is compared.

#### *ejb*

Identifies the EJB to find in the `Collection`.



## Usage Notes

EJB comparison operations require their own tag because two EJB instances might appear identical but have different addresses, and so fail a comparison that should otherwise pass. `dsp:containsEJB` only compares object primary keys and ignores object addresses.

## Example

```
<dsp:getvalueof bean="Student_01" var="student">

<dsp:containsEJB var="seniorReminder" values="{student.class}" ejb="senior"/>
  <c:choose>
    <c:when test="{seniorReminder}">
      Don't forget to order your cap and gown!
    </c:when>
  </c:choose>
</dsp:getvalueof>
```

In this example, the `dsp:containsEJB` tag determines whether `senior` is an EJB in `Student_01.class`. If so, the specified message displays.

# dsp:demarcateTransaction

Encapsulates a transaction.

```
<dsp:demarcateTransaction var-spec>
  ...
</dsp:demarcateTransaction>
```

## Attributes

### *var-spec*

Defines an EL variable or scripting variable with one of these attributes:

Attribute	Description
<code>var</code>	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>session</code> , or <code>application</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
<code>id</code>	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has three properties:

- **success:** A Boolean property, specifies the status of transaction. A call to `isSuccess` obtains the property's setting:
  - **true:** The start and commit operations succeeded.
  - **false:** The start or commit operation failed.
- **exception:** Identifies the `Throwable` exception object produced when a transaction fails to start properly. A call to `getException` returns the exception object.
- **commitException:** Identifies the `Throwable` exception object produced when a transaction fails to commit. A call to `getCommitException` returns the exception object.

## Usage Notes

The `dsp:demarcateTransaction` tag manages a transaction by starting it, checking it for errors, committing the transaction if no errors exist, and rolling back the transaction if errors occur. This tag creates a variable whose properties—`success`, `exception`, and `commitException`—indicate the status of the transaction.

**Note:** An error might occur if `dsp:demarcateTransaction` starts execution when another transaction is in progress.

## Example

The following example assumes the following conditions are true:

- `currentStudent` is a page parameter that is set while a servlet bean iterates through all students in a class.
- A mechanism exists to consecutively save each student's grade to the `currentGrade` page parameter.

---

```
<dsp:importbean bean="\atg\samples\Student_01" var="student"/>
<dsp:importbean bean="\atg\samples\Algebra" />

<dsp:demarcateTransaction var="demarcateXA">
  <dsp:setvalue bean="Algebra.class.student" value="{student.name}"/>
  <dsp:setvalue bean="Algebra.class.grade" value="{student.grade}"/>

  <c:if test="{!demarcateXA.success}">
    The grade could not be processed. Here's why:
    <c:choose>
      <c:when value="{!empty demarcateXA.exception}"/>
        <c:out value="{demarcateXA.exception}"/>

      <c:when value="{!empty demarcateXA.commitException}"/>
```



```

        <c: out value="${demarcateXA.commitException}"/>
      </c: choose>
    </c: if>
  </dsp: demarcateTransaction>

```

In this example, the transaction encapsulates the tasks required to save the grades provided to students in a class. The transaction ensures that student names and grades are added to Algebra. Student\_01 at the same time. If the transaction fails, no name or grade is saved; instead, the appropriate message displays for the exception or commitException exceptions.

## dsp:droplet

Invokes a servlet bean.

```

<dsp: droplet [var-spec] name=" servlet-bean" >
  ...
</dsp: droplet>

```

### Attributes

#### ***name***

Sets the Nucleus path and servlet bean to invoke.

#### ***var-spec***

Defines an EL variable or scripting variable with one of these attributes:

Attribute	Description
var	Names an EL-accessible Map variable, which holds a servlet bean's parameters and parameter values. When you use var, you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
id	Names a scripting variable that scriptlets and expressions can access at runtime.

### Usage Notes

The dsp: droplet tag lets you invoke a servlet bean from a JSP page. It encapsulates programming logic in a server-side JavaBean and makes that logic accessible to the JSP page that calls it. Tag output is held by its parameters, which can be referenced by nested tags.



## Example

```
<dsp: dropl et var="fe" name="/atg/dynamo/dropl et/ForEach">
  <dsp: param bean="/sampl es/Student_01. subj ects" name="array"/>
  <dsp: oparam name="outputStart">
    <p>The student is registered for these courses:
  </dsp: oparam>

  <dsp: oparam name="output">
    <li><c: out value="{fe. el ement}"/>
  </dsp: oparam>
</dsp: dropl et>
```

In this example, the `dsp: dropl et` tag calls the `ForEach` servlet bean and creates a `Map` variable called `fe` that saves each parameter and parameter value used on this page. The `ForEach` servlet bean iterates over the `Student_01. subj ects` array. Beneath the specified message text, the `c: out` tag displays each array element.

## dsp:equalEJB

Checks whether two EJBs have the same primary keys.

```
<dsp: equal EJB var-spec ej b1="ej b-spec" ej b2="ej b-spec" />
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that captures the Boolean result, defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>sessi on</code> , or <code>appl i cati on</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
i d	Names a scripting variable that scriptlets and expressions can access at runtime.

#### *ejb1/ejb2*

Specify the two EJBs to compare.





## Example

---

```
<dsp: equal EJB var="frisbeeAdvertisement" ejb1="${winter}"
  ejb2="${currentSeason}"/>
  <c: choose>
    <c: when test="${frisbeeAdvertisement}">
      Try out for Ultimate Frisbee in April!
    </c: when>
  </c: choose>
```

---

When the current season is winter, the two EJBs have the same primary keys so this code sample advertises try outs for the ultimate Frisbee team.

## dsp:form

Sets up a form for user input.

---

```
<dsp: form
  method="{get|post}"
  [action="jsp-page"]
  [synchronized="component-spec"]
  [formid="id"]
  [requiresSessionConfiguration="{true|false}"]
/>
```

---

### Attributes

#### **method**

Specifies the action the ATG platform should perform with the data associated with the dsp: input tags:

- **get** retrieves the value in the component property indicated in the dsp: input bean attribute and displays it in the input field.
- **post** performs the get operation and saves updated data in the input field to the same bean properties.

#### **action**

Designates the JSP to display on form submission. If no value is provided, the page holding the dsp: form tag is displayed upon form submission.

**Note:** If a form's contents are not on the same page as the dsp: form tag, the dsp: form tag must include the **action** attribute.

***synchronized***

Specifies the Nucleus path and name of a component that is blocked from accepting other form submissions until this form is submitted. If a form uses a form handler that is global or session scoped, new data is liable to overwrite its data before it is saved to the database. See [Synchronizing Form Submissions](#) for more information.

***formid***

Assigns a unique identifier to the form. Forms require unique identifiers when a page specifies multiple forms and conditionally displays only one of them. By providing a form ID, the page compiler can distinguish one form from another.

**Note:** You can also use hidden input fields to differentiate various forms. This mechanism is especially useful when you generate multiple forms dynamically; in this case, you can test the value of each form's hidden input field to determine which form to render.

***requiresSessionConfirmation***

If set to true, prevents cross-site attacks. The HTTP request that is generated on form submission supplies the request parameter `_dynSessConf`, which contains a randomly generated session long number that is associated with the current session. On receiving this request, the request-handling pipeline validates `_dynSessConf` against the current session's confirmation number. If it detects a mismatch or missing number, it blocks further request processing.

**Usage Notes**

`dsp: form` lets you set up a form that accepts user inputs, which are saved as component properties. The methods associated with form operations process JSP tags into forms, validate user-entered data, and manage errors. For more information, see [Forms](#).

**Note:** The `dsp: form` tag uses the `class` attribute in place of the cascading stylesheet `class` attribute to avoid using a Java-reserved word in Java code.

**Example**

```
<dsp: form action="FacilitySchedule.jsp" method="post"
synchronized="/atg/droplet/FacilityFormHandler">

Room you'd like to reserve: <dsp: select bean="Classroom.roomNumbers">
  <p><dsp: option value="10">10</dsp: option>
  <p><dsp: option value="12">12</dsp: option>
  <p><dsp: option value="14">14</dsp: option>
  <p><dsp: option value="16">16</dsp: option>
  <p><dsp: option value="20">20</dsp: option>
</dsp: select>
<p>Course Title: <dsp: input type="text"
  bean="FacilityFormHandler.course"/>
<p>Instructor:
```



```

<p><dsp:input type="radio" bean="FacilityFormHandler.instructor"
  value="pear">Ms. Pear</dsp:input></p>
<p><dsp:input type="radio" bean="FacilityFormHandler.instructor"
  value="apricot"/>Mr. apricot</dsp:input>
<p><dsp:input type="radio" bean="FacilityFormHandler.instructor"
  value="canteen"/>Ms. Canteen</dsp:input>
<p><dsp:input type="radio" bean="FacilityFormHandler.instructor"
  value="pineapple">Mr. Pineapple</dsp:input>
<dsp:input type="submit" value="Reserve Room"
  bean="FacilityFormHandler.create"/>
</dsp:form>

```

This example demonstrates a form for reserving a classroom. Using this form, a user can specify:

- desired room (drop-down list)
- course title (text box)
- teacher's name (group of radio buttons)

When a user clicks Reserve Room, the entered data is saved to the `FacilityFormHandler` and authorized, and when data is approved as valid, saved to the database.

Because this example uses the post method, the form is pre-populated with the data last-saved to `FacilityFormHandler`—for example, 16, Geometry 101, Ms. Pear. There are no specific failure or success instructions; after form submission, `FacilityScheduler.jsp` is displayed as specified by the form's `action` attribute.

## dsp:frame

Generates an HTML frame.

```
<dsp:frame {page="page-spec" | src="page-spec" [otherContext="context"]} />
```

### Attributes

#### *page*

Sets the page's path and file name in two ways:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's `contextRoot` to the specified value.

**src**

Sets the page's path and file name. The `src` attribute can access pages in another Web application if the current application's `web.xml` `atg.dynamo.contextPaths` parameter is configured appropriately, or if the `otherContext` attribute specifies the correct context path. For more information, see [Invoking Pages in Other Web Applications](#).

**otherContext**

identifies a context path that is not specified in the application's `web.xml`; valid with the `src` attribute.

**Usage Notes**

`dsp: frame` acts just like the HTML `frame` tag; you can use it to embed other pages or screen elements into a parent page. There is no corresponding `frameset` tag because you can use the standard HTML `frameset` tag. The DSP tag libraries tags let you access data and represent that data in your pages, but they do not handle page layout or presentation. The primary difference between the HTML `frame` tag and `dsp: frame` is that you pass page parameters and bean property values from a parent page to the child page.

**Note:** The `dsp: frame` tag uses the `iclass` attribute in place of the cascading stylesheet `class` attribute to avoid using a Java reserved word in Java code.

**Example**

```
<dsp: frame page="en/brokerhome.jsp" />
```

In this example, `brokerhome.jsp` is inserted into the parent page.

## dsp:getvalueof

Creates an EL variable that references the specified component property or page parameter.

---

```
<dsp: getvalueof  
    source-value var-spec [data-type]  
</getvalueof>
```

---

**Attributes****source-value**

Specifies the value sources as a [JavaBean component](#), [page parameter](#), or [static value](#), as follows:



- `bean="nucl eus-path/component-name"`
- `param="param-name"`
- `value="static-value"`

### ***var-spec***

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
<code>var</code>	Names an EL variable. When you use <code>var</code> , you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
<code>id</code>	Names a scripting variable that scriptlets and expressions can access at runtime.

### ***data-type***

By default, the variable that is created by `dsp: getvalue` is of type `java. lang. Object`. You can set the variable to a different type through the `var type` and `id type` attributes, which qualify the `var` and `id` attributes, respectively. For example, if you set the EL variable to a page parameter that holds a string, set its `var type` attribute to `java. lang. String`. This enables the returned value to execute string-specific methods without explicit typecasting.

### **Example**

```
<dsp: getvalueof var="backgroundcolor" bean="Profile.preferredColor"
  var type="java. lang. String">
  <body bgcolor="{ ${backgroundcolor} }" >
</dsp: getvalueof>
```

In this example, the tag sets the `backgroundcolor` attribute to the value held by the `preferredColor` or property of the `Profile` component. The `preferredColor` value is kept as a `String`. The EL variable renders the `backgroundcolor` value so the current page background color is set to it.

## **dsp:go**

Sets up a form in a WML page.

```
<dsp: go
  method="{get|post}"
```



```
href="wml -page"  
[synchroni zed=" component -spec" ]  
...  
/dsp: go>
```

---

## Attributes

### *href*

Identifies the WML page to display on form submission, if others are not supplied in URLSuccess and URLFailure properties. Often, this attribute points to the form page that supplies it.

### *method*

Specifies the action the ATG platform should perform when sending data associated with the dsp: postfield tags to the server:

- get retrieves the value in the component property indicated in the dsp: postfield bean attribute and displays it in the input field.
- post performs the get operation and saves updated data in the input field to the same bean properties.

### *synchronized*

Specifies the Nucleus path and name of a component that is blocked from accepting other form submissions until this form is submitted. If a form uses a form handler that is global or session scoped, new data is liable to overwrite its data before it is saved to the database. See [Synchronizing Form Submissions](#) for more information.

## Usage Notes

dsp: go allows you set up a form in a WML page that accepts user inputs, which are saved as component properties. The methods associated with form operations process JSP tags into forms, validate user-entered data, and manage errors. This tag provides the functionality accomplished by dsp: form to the wireless context.

## Example

---

```
<dsp: go href='<%= request.getContextPath() +"login.jsp"%>' method="post">  
  <dsp: postfield bean="ProfileFormHandler.value.login" value="$login" />  
  <dsp: postfield bean="ProfileFormHandler.value.password"  
    value="$password" />  
  <dsp: postfield bean="ProfileFormHandler.login" value=" " />  
  <dsp: postfield bean="ProfileFormHandler.loginSuccessURL"  
    value="FormSuccess.jsp"/>  
</dsp: go>
```

---



This example describes a page that handles login information. The first line specifies the default page to display on form submission. The form page, called `login.jsp`, is located in the same context as the current page. To do this, the context path provided in the request object is prepended to the path preceding the page name.

The remaining tags handle login, validation, and success actions. The next two tags save login and password values held in WML variables to the `ProfileFormHandler` so when the login handler method is invoked, (specified in the next line of code) the ATG platform verifies the login/password pair against those already in existence. Successful login diverts users to the page provided by the `ProfileFormHandler.loginSuccessURL` property, which is set to `FormSuccess.jsp`.

## dsp:iframe

Embeds an iframe in the current JSP.

---

```
<dsp:iframe {page="page-spec" | src="page-spec" [otherContext="context" ]}/>
```

---

### Attributes

#### *page*

Sets the page's path and file name in two ways:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's `context root` to the specified value.

#### *src*

Sets the page's path and file name. The `src` attribute can access pages in another Web application if the current application's `web.xml` `atg.dynamo.contextPaths` parameter is configured appropriately, or if the `otherContext` attribute specifies the correct context path. For more information, see [Invoking Pages in Other Web Applications](#).

#### *otherContext*

identifies a context path that is not specified in the application's `web.xml`; valid with the `src` attribute.

### Usage Notes

`dsp:iframe` lets you use an `iframe` in a page in the same way you use an `iframe` tag in an HTML page. Furthermore, you can set an `iframe` page to accept the page parameters from its parent page.

**Note:** `dsp:iframe` uses the `iclass` attribute in place of the cascading stylesheet `class` attribute to avoid using a Java-reserved word in Java code.



## Example

---

```
<dsp:iframe src="GraduationAnnouncements.jsp"/>
```

---

The GraduationAnnouncements.jsp page is embedded in a parent page as an iframe.

## dsp:img

Inserts an image into a JSP.

---

```
<dsp:img {page="page-spec" | src="page-spec" [otherContext="context"]} />
```

---

### Attributes

#### *page*

Sets the page's path and file name in two ways:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's context root to the specified value.

#### *src*

Sets the page's path and file name. The src attribute can access pages in another Web application if the current application's web.xml atg.dynamo.contextPaths parameter is configured appropriately, or if the otherContext attribute specifies the correct context path. For more information, see [Invoking Pages in Other Web Applications](#).

#### *otherContext*

identifies a context path that is not specified in the application's web.xml; valid with the src attribute.

### Usage Notes

dsp:img lets you insert an image file into a JSP, like an HTML img tag. The ATG platform processes the image source URL by encoding it.

**Note:** dsp:img uses the class attribute in place of the cascading stylesheet class attribute to avoid using a Java reserved word in Java code.

## Example

```
<dsp:img page="en/images/IdPhoto.gif">
```





In this example, the `IDPhoto.gif` image, located in `en/images`, is imported into the parent page.

## dsp:importbean

Imports a servlet bean or JavaBean component into a JSP.

---

```
<dsp:importbean bean="Nucleus-path" [var="attr-name" [scope="scope-spec" ] ]
```

---

`dsp:importbean` can import a servlet bean or a JavaBean component into a JSP. After a component is imported, references to it can omit its full Nucleus path. You can also use `dsp:importbean` to map the imported component to a scoped attribute that is EL-accessible. `dsp:importbean` tags are typically placed at the top of the JSP, in order to facilitate references to them.

### Attributes

#### *bean*

Set to the fully-qualified Nucleus path and name of the servlet bean to import.

#### *var*

Names an EL variable that enables access to other page tags. When you use `var`, you can set its scope attribute to `page`, `request`, `session`, or `application`. The default scope is `page`.

If `var` is omitted, the shorthand name for the component is used.

#### *scope*

By default, the named variable is set to `page` scope, which makes it accessible to any EL-enabled resources on the same page. The `scope` attribute lets you define a larger scope for variable, by setting it to `request`, `session`, or `application`. For more information, see [EL Variable Scopes](#) in Chapter 2.

### Example

---

```
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>
<dsp:importbean bean="/atg/dynamo/droplet/Redirect"/>
<dsp:importbean bean="/atg/dynamo/servlet/RequestLocale" var="requestLocale"/>

<dsp:setvalue bean="${requestLocale.refresh}" value=""/>
<dsp:droplet name="Switch">
  <dsp:param bean="${requestLocale.locale.language}" name="value"/>
  <dsp:oparam name="fr">
    <dsp:droplet name="Redirect">
      <dsp:param name="url" value="fr/index.jsp"/>
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```



```
</dsp:oparam>
<dsp:oparam name="de">
  <dsp:droplet name="Redirect">
    <dsp:param name="url" value="de/index.jsp"/>
  </dsp:droplet>
</dsp:oparam>
<dsp:oparam name="default">
  <dsp:droplet name="Redirect">
    <dsp:param name="url" value="en/index.jsp"/>
  </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

---

In this example, `Switch`, `RequestLocal`, and `Redirect` are imported to the current JSP. When these components are used elsewhere in the page, they use the component and property names only. The `RequestLocal` component is imported to a `var` attribute so the `dsp:setvalue` and `dsp:param` tags can access `RequestLocal` with EL.

## dsp:include

Embeds a page fragment in a JSP.

```
<dsp:include {page="page-spec" | src="page-spec" [otherContext="context"]}
  staticCharset="charset"
/>
```

---

### Attributes

#### *page*

Sets the page's path and file name in two ways:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's `context root` to the specified value.

#### *src*

Sets the page's path and file name. The `src` attribute can specify a page in another Web application if one of the following is true:

- `src` is set as follows:

```
src="webAppName: /path"
```

where the application's `WebAppRegistry` maps *webAppName* to a context root. The *path*-specified page is searched within that context root.



- The current application's `web.xml` `atg.dynamo.contextPaths` parameter is configured appropriately,
- The `otherContext` attribute specifies the correct context path.

For more information, see [Invoking Pages in Other Web Applications](#).

### ***otherContext***

identifies a context path that is not specified in the application's `web.xml`; valid with the `src` attribute.

### ***staticcharset***

Specifies the character set to use when bytes are converted to characters. The default setting character set is ISO-8859-1.

## **Usage Notes**

`dsp:include` lets you embed a page fragment in a JSP just as you can with the JSP `include` tag. It is also possible to pass page parameters from the parent page to the included page using this tag.

For information on other methods for embedding documents, see [Included Pages](#).

## **Example**

---

```
<dsp:include otherContext="/myOtherWebApp"
page="/SophomoreRegistration.jsp"/>
```

---

The `SophomoreRegistration.jsp` page is embedded in the parent page.

For more information about using `dsp:include`, see [Included Pages](#).

## **dsp:input**

Creates a form control that accepts user input.

---

```
<dsp:input [type="input-control"] [name="input-name"]
  bean="property-spec" ["source-spec"]
  [checked="{true|false}"]
  [default="value"]
  [priority=integer-value]
/>

<dsp:input type="submit"
  [name="input-name"]
```



```
[bean="property-spec" [submit value="value"] value="value"]
/>

dsp:input type="image" {src|image}=path
    [name="input-name"]
    [bean="property-spec" submit value="value"]
/>
```

---

## Attributes

### **type**

Defines the HTML input control—for example, as a checkbox, radio button, or text box, among others. If this attribute is omitted, the default type is `text`. To learn about input fields, see [Setting Property Values in Forms](#).

You can also define a submit control by setting `type` to one of these values:

- `submit` specifies to display a submit button.
- `image` defines a submit control that uses the image specified by the `src` or `page` attribute.

### **name**

Assigns a name to this input field, which enables access to it during form processing. If no name is provided, the ATG platform automatically supplies one. For all input types other than `radio` and `checkbox`, the assigned name must be unique.

**Note:** if two or more submit input tags are associated with the same property, each submit input tag must have a unique name attribute.

### **bean**

Specifies the property that this input field updates on form submission: its Nucleus path, component name, and property name. The property specification can also include a tag converter, such as `date` or `null`. For more information, see [Tag Converters](#) in Chapter 2.

The bean attribute for a submit control can also specify a form handler operation such as `cancel` or `update`.

### **source-spec**

Specifies to prepopulate the input field from one of the following sources:

- `beanvalue="property-spec"`  
Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:  
`bean="Student_01.firstClass" beanvalue="FreshmanScience.name"`
- `paramvalue="param-name"`



Specifies the value from the page parameter *param-name*. For example:

```
bean="Student_01. name" paramvalue="name"
```

- `value="value"`

Specifies a static value. For example:

```
bean="Student_01. enrolled" value="true"
```

If *source-value* is omitted, the input field contains the target property's current value.

### **checked**

If set to true, displays a checkbox or radio button as selected.

### **default**

Specifies a default value to save to the component property if the user neglects to supply a value and none is otherwise provided. Options include true.

### **priority**

Specifies the priority of this input field during form processing, relative to other input fields. This attribute can be especially helpful when making changes to an item's repository ID and the properties it contains. If you first update the supplemental properties and then the repository ID, the new property values can be flushed when the repository ID change occurs.

For more information, see [Order of Tag Processing](#).

### **value**

Valid only if the *type* attribute is set to *submit*, this attribute sets two values:

- The value of the bean-specified property if the *submitvalue* attribute is omitted.
- The submit button's label.

### **submitvalue**

Sets the target property on activation of the submit control. The value must be a constant. For example:

```
<dsp:input type="submit" bean="/TestData/Student_01. age" submitvalue="34"
value="Click here" />
```

### **src**

Specifies the path and file name of the image to use if *submit-control* is set to *image*. You can set *src* to a relative or absolute path, which is resolved as follows:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web server doc root as a starting point.

**page**

Specifies the path and file name of the image to use if *submit-control* is set to image. You can set page to a relative or absolute path, which is resolved as follows:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's context root to the specified value.

**Usage Notes**

dsp:input creates a form control that accepts user input, much like the HTML input tag. dsp:input can create two kinds of controls:

- A form field that contains user-entered data that is passed to a component.
- A submit button or image.

All dsp:input tags must be embedded between <dsp:form> . . </dsp:form> tags, which are either in the same JSP or a parent JSP (see [Embedding Pages in Forms](#)), so it is subject to the form handler's submit method.

dsp:input defines the input field—for example, as a text box, radio button group, drop-down list—and the value to save to the specified component property. Initially, the input field can display a value derived either from the target property, another source, or the specified default.

**Note:** dsp:input uses the class attribute in place of the cascading stylesheet class attribute to avoid using a Java reserved word in Java code.

**Embedding custom attributes**

A dsp:input tag can embed one or more custom attributes through the [dsp:tagAttribute](#) tag, as follows:

---

```
<dsp:input . . . >
  <dsp:tagAttribute name="attr-name" value="value" />
</dsp:input/>
```

---

You can use [dsp:tagAttribute](#) tags in order to add attributes to the generated HTML beyond the fixed set of attributes supported by the DSP tag library.

**Example**

---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<%@ page import="atg.servlet.*"%>
<dsp:page>
```



```

<dsp:importbean bean="/atg/TestData/StudentFormHandler"/>

<dsp:form action="<%=ServletUtil.getDynamoRequest(request).getRequestURI()%"
method="POST">
  <dsp:input bean="StudentFormHandler.updateSuccessURL" type="hidden"
    value="index.jsp"/>
  <dsp:input type="hidden" bean="StudentFormHandler.updateRepositoryId"
    beanvalue="StudentFormHandler.repositoryId" />

  <table border=0 cellpadding=2>
    <tr>
      <td align=right>First Name </td>
      <td><dsp:input type="text" bean="StudentFormHandler.value.firstName"
        maxlength="30" size="30" required="true"/></td>
    </tr>

    <tr>
      <td align=right>Last Name </td>
      <td><dsp:input type="text" bean="StudentFormHandler.value.lastName"
        maxlength="30" size="30" required="true"/></td>
    </tr>

    <tr>
      <td align=right>Address </td>
      <td><dsp:input type="text"
        bean="StudentFormHandler.value.primaryResidence.address"
        maxlength="30" size="30" required="true"/></td>
    </tr>

    <tr>
      <td align=right>City </td>
      <td><dsp:input type="text"
        bean="StudentFormHandler.value.primaryResidence.city"
        maxlength="30" size="30" required="true"/></td>
    </tr>

    <tr valign=top>
      <td align=right>State/Province </td>
      <td><dsp:input type="text"
        bean="StudentFormHandler.value.primaryResidence.state"
        maxlength="30" size="30" required="true"/></td>
    </tr>

    <tr>
      <td align=right>ZIP/Postal Code </td>
      <td><dsp:input type="text"
        bean="StudentFormHandler.value.primaryResidence.postalCode"
        maxlength="10" size="10" required="true"/></td>
    </tr>
  </table>

```



```
<tr>
  <td align=right>Phone Number </td>
  <td><dsp: input type="text"
    bean="StudentFormHandler. value. primaryResidence. phoneNumber"
    maxLength="20" size="30" required="true" /></td>
</tr>

<tr>
  <td align=right>Birthday (MM/DD/YYYY)</td>
  <td><dsp: input bean="StudentFormHandler. value. dateOfBirth"
    date="M/dd/yyyy" maxLength="10" size="10" type="text" /></td>
</tr>

<tr>
  <td align=right>Gender </td>
  <td>
    <dsp: input type="radio" bean="StudentFormHandler. value. gender"
      value="male">Male</dsp: input>
    <dsp: input type="radio" bean="StudentFormHandler. value. gender"
      value="female">Female</dsp: input>
  </td>
</tr>

<tr>
  <td align=right>Graduation Date </td>
  <td>
    <dsp: select bean="StudentFormHandler. value. gradDate" />
    <dsp: option value="6/2003">2003</dsp: option>
    <dsp: option value="6/2004">2004</dsp: option>
    <dsp: option value="6/2005">2005</dsp: option>
    <dsp: option value="6/2006" selected="true">2006</dsp: option>
  </td>
</tr>

<tr>
  <td align=right>Interests </td>
  <td>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests"
      value="drama" checked="true">Drama</dsp: input>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests"
      value="art" checked="true">Art</dsp: input>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests"
      value="studentGovernment" checked="true">Student
      Government</dsp: input>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests"
      value="sports" checked="true">Sports</dsp: input>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests"
      value="band" checked="true">Band</dsp: input>
    <dsp: input type="checkbox" bean="StudentFormHandler. value. interests">
```





```

        value="academicTeam" checked="true">Academic Team</dsp: input>
<dsp: input type="checkbox" bean="StudentFormHandler.value.interests"
        value="singing" checked="true">Singing</dsp: input>
    </td>
</tr>

<tr>
    <td align="right">Comments</td>
    <td>
        <dsp: textarea bean="StudentFormHandler.value.comments" />
    </td>
</tr>

<dsp: input type="submit" bean="StudentFormHandler.create" value="Register" />
</dsp: form>
</dsp: page>

```

The form in this example might be part of a new student registration page. The page begins with the directive statements that import the DSP tag library and the servlet class. This is followed by an `import` statement that makes the `StudentFormHandler` available to the page like other imported items.

The form elements define the default result page as the active page by retrieving the page path and name dynamically from the request objects `RequestURI` property. The page uses a hidden component to set the `StudentFormHandler` success property to the `index.jsp`. Because a failure URL is not provided, the default (current page) is used. A second hidden bean ensures that the repository ID that represents the form remains the same by updating it with the current value.

The `dsp: input` tags that follow insert text boxes, radio buttons, checkboxes, and extended text fields, that are populated with data when current values exist (implemented by the `post` action). The first seven text boxes present the student's first name, last name, street address, city, state, postal code and phone number, and indicate that they must hold valid data in order for the form to process successfully. Each text box saves data to the `primaryResidence` array of the value Dictionary. The value in the `birthday` text box is converted so it appears in a format consistent with 1/15/1936.

The form includes a set of radio buttons for documenting the student's gender (neither are default selected) and a set for specifying the student's graduation date. In both cases only one item in each group can be selected. Because most new students are freshmen, the last occurring date is the default. The `interests` checkbox group saves all selected items to the `interests` array property. All items are selected by default to encourage students to involve themselves in activities. The extended text box provides a place for free form comments about a student.

The final `dsp: input` tag inserts a Register button that, when clicked, invokes the submit handler method to process the form.

## dsp:layerBundle

Loads a resource bundle whose settings are combined with existing resource bundle settings.



---

```
<dsp:layerBundle basename="bundle-name" [var-spec] />
```

---

## Attributes

### ***basename***

Specifies the base name of the resource bundle *bundle-name*, where *bundle-name* excludes localization suffixes and filename extensions.

### ***var-spec***

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use var, you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
id	Names a scripting variable that scriptlets and expressions can access at runtime.

## Usage Notes

dsp:layerBundle loads an i18n localization context and stores it in a scoped variable or scripting variable. Unlike the JSTL tag `fmt:setBundle`, dsp:layerBundle combines the settings of the specified resource bundle with existing resource bundle settings. These settings remain in effect only for the duration of the specified scope.

## dsp:link

References files such as stylesheets and formatting pages.

---

```
<dsp:link [rel="file-type"]  
  type="mime-type"  
  {href="file-path" | page="file-path"}  
  [title="title"] />
```

---



## Attributes

### *rel*

Describes the item that is referenced; for example:

```
rel="stylesheet"
```

### *type*

Specifies the MIME type that represents the referenced page. The MIME type must be a recognized MIME type, supported by the `MI METyper` and `MI METypeDispatcher` components.

### *href*

Sets the page's path and file name in two ways:

- Resolves relative paths using the current page as a starting point.
- Resolves absolute paths using the Web application root as a starting point by prepending the request object's `contextRoot` to the specified value.

### *page*

Path and file name. Resolves relative paths using the current page as a starting point. Resolves absolute paths using the Web application root as a starting point by prepending the request object's `contextRoot` to the specified value.

### *title*

Contains the title as it appears on the referenced page.

## Usage Notes

`dsp:Link` lets you reference files such as stylesheets and formatting pages and makes them available to the parent page in much the same way as the HTML `Link` tag.

**Note:** `dsp:Link` uses the `iclass` attribute in place of the cascading stylesheet `class` attribute to avoid using a Java reserved word in Java code.

## Example

```
<dsp:Link rel="stylesheet"
  type="text/css"
  href="Style.css"
  title="Jefferson High School StyleSheet"/>
```

This example creates a link from the current page to a stylesheet called `style.css` that is located in the same directory. The linked page uses the standard MIME TYPE for stylesheets (`text/css`) and is titled `Jefferson High School StyleSheet`.



## dsp:oparam

Supplies a parameter to the current servlet bean.

---

```
<dsp:oparam name="param-name">
  ...
</dsp:oparam>
```

---

### Attributes

#### ***name***

Identifies the parameter to set for the current servlet bean.

#### ***value***

The value to assign this parameter.

### Usage Notes

dsp:oparam (open parameter) takes the value supplied to it in a JSP and passes it to the current dsp:droplet tag. Each servlet bean maintains a set of open parameters which the servlet bean can set, service, and deliver as output to a JSP. For information about open parameters that are available for specific servlet beans, see [Appendix B: ATG Servlet Beans](#).

See the *How JSPs Handle Servlet Beans* section in the *Creating and Using ATG Servlet Beans* chapter of the [ATG Programming Guide](#).

### Example

---

```
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>

<dsp:valueof param="person.name"/>: &nbsp;

<dsp:droplet name="Switch">
  <dsp:param name="value" param="person.hasCar"/>
  <dsp:oparam name="false">
    No vehicle
  </dsp:oparam>
  <dsp:oparam name="true">
    Vehicle with capacity of <dsp:valueof param="person.carCapacity"/>
  </dsp:oparam>
</dsp:droplet>
```

---

The Switch servlet bean evaluates the input parameter value and tests the Boolean result against the open parameters true and false. The servlet bean processes the matching open parameter.



## dsp:option

Used with [dsp:select](#) to display an option in a drop-down list.

---

```
<dsp:option value-spec label  
    selected="{true|false}"  
</dsp:option>
```

---

### Attributes

#### *value-spec*

Specifies this option's value in one of the following ways:

- `beanvalue="property-spec"`  
Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:  
`beanvalue="FreshmanScience.name"`
- `paramvalue="param-name"`  
Specifies the value from the page parameter *param-name*. For example:  
`paramvalue="name"`
- `value="value"`  
Specifies a static value. For example:  
`value="Bartleby Scribe"`

The value specification can include a tag converter, such as `date` or `null`. For more information, see [Tag Converters](#) in Chapter 2.

#### *label*

This option's displayed text.

#### *selected*

Set to `true` or `false` in order to override the value obtained from the target property specified by `dsp:select`:

- `true`: The option displays as selected.
- `false`: The option displays as unselected.

**Note:** You can specify to select multiple options if the parent `dsp:select` tag sets its `multiple` attribute to `true`.

## Usage Notes

`dsp:option` is used in the context of `dsp:select` to display a drop-down list option. This tag is only valid when enclosed in a `dsp:select` tag.

**Note:** `dsp:option` uses the `iclass` attribute in place of the cascading style sheet `class` attribute to avoid using a Java reserved word in Java code.

## Example

---

```
<dsp:select bean="Student_01.summerHobbies" multiple="true">
  <dsp:option value="hiking">hiking</dsp:option>
  <dsp:option value="biking">biking</dsp:option>
  <dsp:option value="swimming">swimming</dsp:option>
</dsp:select>
```

---

This drop-down list provides a list of hobbies that include hiking, biking, and swimming. The `summerHobbies` property is a `Collection`, so it can hold as many options as the user selects. No options use the `selected` attribute, so none display as selected unless the `summerHobbies` property is already set to one or more of those values.

## dsp:orderBy

Defines the sort order for the enclosing `dsp:sort` tag.

---

```
<dsp:orderBy property="sort-property" reverse="{true|false}" />
```

---

## Attributes

### *property*

The property on which to sort. Two constraints apply:

- The property cannot be a Dynamic Bean.
- The property must belong to the object specified by the `values` attribute of the parent `dsp:sort`.

### *reverse*

After the item is sorted, determines whether to use the initial order (false) or reverse it (true). If you omit this attribute, the default is false.



## Usage Notes

`dsp: orderBy` specifies the properties used to sort items in the `dsp: sort`-specified object. Multiple `dsp: orderBy` tags within the same `dsp: sort` tag specify multiple sort levels. For example, given a Collection of user profiles, you might sort first on the ages, and within each age, sort on last names, then first names.

## Example

---

```
<dsp: importbeantomap bean="AllStudents" var="students"/>
<dsp: sort var="sorter" values="{students}">
  <dsp: orderBy property="age" reverse="true"/>
  <dsp: orderBy property="lastName"/>

  <c: forEach var="forEach" begin="0" items="{sorter.sortedArray}">
    <li>Name: <c: out value="{students.lastName}"/>
  </c: forEach>
</dsp: sort>
```

---

In this example, the students are ordered first in reverse by age, and then as a secondary sort, they are ordered by last name. The eldest students are returned at the top of the list and those who have the same age are organized alphabetically by last name.

## dsp:page

---

```
<dsp: page
  [xml="{true|false}"]
  [useXmlParamDelimiter="{true|false}"]
>
  ...
</dsp: page>
```

---

## Attributes

### *xml*

Specifies the page output as HTML (false) or XML (true). If you omit this attribute, the page output is HTML, unless a true setting is inherited from a parent page. To designate different output types within the same page, use [dsp: setxml](#).

### *useXmlParamDelimiter*

Specifies the delimiter used by `DynamoHttpServletRequest.addQueryParam` to separate query parameters:



- true: Use the XML delimiter &amp;
- false: Use HTML delimiter &

A value provided by a `dsp: page` or `dsp: setxml` tag to a parent page is automatically inherited by an included page, unless the included page itself declares a delimiter. If no value is detected, the ATG platform uses the delimiter associated with the page's MIME type.

## Usage Notes

`dsp: page` encloses a JSP: the first tag follows the page directive and the last tag ends the page.  
`dsp: page` invokes the JSP handler, which calls the servlet pipeline and generates `HttpServletRequest`. The servlet pipeline manages generic page-related tasks, such as retrieving the servlet session components, instructing the page to render its contents, and clearing cached information.

## Example

```
<%@ taglib uri=http://www.atg.com/taglibs/daf/dspjspTaglib1_0 prefix="dsp" %>
<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c' %>

<dsp: page>

<dsp: droplet name="/atg/dynamo/droplet/Switch">
  <dsp: param bean="/atg/dynamo/servlet/RequestLocale.Locale.Language"
name="value"/>
  <dsp: oparam name="fr"><%response.sendRedirect("fr"); %></dsp: oparam>
  <dsp: oparam name="de"><%response.sendRedirect("de"); %></dsp: oparam>
  <dsp: oparam name="ja"><%response.sendRedirect("ja"); %></dsp: oparam>
  <dsp: oparam name="en"><%response.sendRedirect("en"); %></dsp: oparam>
  <dsp: oparam name="default"><%response.sendRedirect("en/default.jsp"); %>
  </dsp: oparam>
</dsp: droplet>

</dsp: page>
```

This page points to a page in the user's preferred language. The language preference is obtained from the `RequestLocale.Locale.Language` property. The `dsp: page` tags surround the body of the JSP; only the page directive that defines the tag library precedes the `dsp: page` tags.

## dsp:param

Identifies a servlet bean input parameter; or defines a page parameter.





```
<dsp: param name="sbparam-name" sbparam-value />

<dsp: param name="pgparam-name" pgparam-value />
```

Attributes

name (servlet bean)

Identifies an input parameter that is defined for the current servlet bean. For information on input parameters for specific servlet beans, see [Appendix B: ATG Servlet Beans](#).

name (page parameter)

Defines a page parameter that is accessible to the current JSP and embedded child pages. For more information, see [Page Parameters](#).

sbparam-value

Specifies the input parameter’s value in one of the following ways:

bean="prop-spec"	Sets the parameter to the specified property, where <i>prop-spec</i> includes the Nucleus path, component name, and property name. For example:  <dsp: param name="repository" bean="/atg/dynamo/droplet/PossibleValues.repository"/>
param="pName"	Sets the parameter to the value of the page parameter <i>pName</i> . For example:  <dsp: param name="propertyName" param="element.repositoryId"/>
value="value"	Sets the parameter to a static value. For example:  <dsp: param name="sortProperties" value="-date"/>

The parameter value specification can also include a tag converter such as [date](#) or [nullable](#). For more information, see [Tag Converters](#) in Chapter 2.

pgparam-value

Specifies the page parameter’s value in one of the following ways:

beanvalue="prop-spec"	Sets the parameter to a property value, where <i>prop-spec</i> includes a Nucleus path, component name, and property name. For example:  beanvalue="FreshmanScience.name"
-----------------------	---



<code>paramvalue="pName"</code>	Sets the parameter from the page parameter <i>pName</i> . For example: <code>paramvalue="name"</code>
<code>value="value"</code>	Sets the parameter to a static value. For example: <code>value="Bartleby Scribe"</code>

## dsp:postfield

```
<dsp:postfield [name="input-name"]
  bean="property-spec" ["source-spec"]
  [checked="{true|false}"]
  [default="value"]
  [priority=integer-value]
  [submitvalue="value"]
/>
```

### Usage Notes

`dsp:postfield` creates an object that accepts user input much like the WML tag `postfield`. You can use `dsp:postfield` to pass data to a component property or page parameter. Sometimes the input field displays with a value, which can be the current value in the property specified for update or another value provided as default.

You can also use `dsp:postfield` to process a form by making a call to a handler method. When you use this tag, make sure it is enclosed in a `dsp:go` tag. `dsp:postfield` must be used in the context of a `dsp:go` tag so it is executed by a submit handler method.

#### **name**

Assigns a name to this input field, which enables access to it during form processing. If no name is provided, the ATG platform automatically supplies one.

#### **bean**

Specifies the property that this input field updates on form submission: its Nucleus path, component name, and property name. The property specification can also include a tag converter, such as date or null. For more information, see [Tag Converters](#) in Chapter 2. For example:

```
<dsp:postfield bean="/atg/userprofile/ProfileFormHandler.name" required="true"/>
```

The bean attribute for a submit control can also specify a form handler operation such as cancel or update. For example:



```
<dsp: postfield type="submit" bean=/atg/userprofiling/ProfileFormHandler.cancel"/>
```

### **source-spec**

Specifies to prepopulate the input field from one of the following sources:

- `beanvalue="property-spec"`  
Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:  
`bean="Student_01.firstClass" beanvalue="FreshmanScience.name"`
- `paramvalue="param-name"`  
Specifies the value from the page parameter *param-name*. For example:  
`bean="Student_01.name" paramvalue="name"`
- `value="value"`  
Specifies a static value. For example:  
`bean="Student_01.enrolled" value="true"`

If *source-value* is omitted, the input field contains the target property's current value.

### **priority**

Specifies the priority of this input field during form processing, relative to other input fields. This attribute can be especially helpful when making changes to an item's repository ID and the properties it contains. If you first update the supplemental properties and then the repository ID, the new property values can be flushed when the repository ID change occurs.

For more information, see [Order of Tag Processing](#).

### **submitvalue**

Sets the bean-specified property on form submission. The value must be a constant. For example:

```
<dsp: postfield bean="/TestData/Student_01.age" submitvalue="34"
value="Click here"/>
```

## **Example**

```
<p>Login: <input type="text" name="login" title="login"/></p>
<p>Password: <input type="password" name="password" title="password"/></p>

<do type="accept" label="<input type="text" name="login" title="login"/>">
  <dsp: go href='<%= request.getContextPath() + "/wml/login.jsp"%>'
  method="post">
    <dsp: postfield bean="ProfileFormHandler.value.login" value="$login"
    />
    <dsp: postfield bean="ProfileFormHandler.value.password">
```



```
        value="$password" />
<dsp: postfield bean="ProfileFormHandler.login" value="Login" />
<dsp: postfield bean="ProfileFormHandler.loginSuccessURL" value="/Form
    Success.jsp" />
</dsp: go>
</do>
```

---

This example shows an excerpt from a login form. Notice that login and password entered by the user in the input fields are saved to WML parameters `$login` and `$password` respectively. This makes the login and password available to other tags on the page, so the `dsp: postfield` tag can save them to the `ProfileFormHandler` properties. By setting the user inputs to the property values via WML variables, you are able to workaround the `dsp: postfield` constraint that prevents you from saving the user input to a component property directly.

The `dsp: postfield` tag calls the login handler method so when a submit control is clicked, the ATG platform validates the login/password pair in the `ProfileFormHandler`. After the ATG platform accepts the login/password pair, the user is directed to the page identified in `loginSuccessURL`, which is set to `FormSuccess.jsp`.

## dsp:property

Sets a component property from [dsp: a](#) tag.

---

```
<dsp: property bean="property-spec" source-spec
    [name="input-name" ]/>
```

---

### Attributes

#### **bean**

Specifies the property to update on activation of the enclosing [dsp:a](#) anchor tag: its Nucleus path, component name, and property name. The property specification can also include a tag converter, such as date or null. For more information, see [Tag Converters](#) in Chapter 2.

#### **source-spec**

Specifies the source value in one of the following ways:

- `beanvalue="property-spec"`  
Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:  
`bean="Student_01.firstClass" beanvalue="FreshmanScience.name"`
- `paramvalue="param-name"`  
Specifies the value from the page parameter *param-name*. For example:



```
bean="Student_01. name" paramvalue="name"
```

- `value="value"`

Specifies a static value. For example:

```
bean="Student_01. enrolled" value="true"
```

### ***name***

Assigns a name to the target property, which is used to construct the corresponding query parameter. The names of all `dsp:property` tags within a given `dsp:a` tag must be unique, and cannot conflict with existing query parameters for the page.

## **Usage Notes**

You can embed multiple `dsp:property` tags within a `dsp:a` tag, in order to update multiple properties with a single `dsp:a` invocation.

This tag cannot be used together with the `dsp:a`'s `bean` attribute.

### ***Internet Explorer constraints***

Internet Explorer has a maximum URL length of 2083. The query parameter name that is generated for each property uses its complete Nucleus path and name, and is repeated twice. You can easily exceed this URL maximum length with multiple properties. To shorten the URL, use the `name` attribute to assign a short name to each property. For example:

```
<dsp:a href="test.jsp">my page
  <dsp:property bean="/atg/dynamo/test/MyForm.strProp" value="fpp" name="a" />
  <dsp:property bean="/atg/dynamo/test/MyForm.intProp" value="3" name="b" />
</dsp:a>
```

## **dsp:rollbackTransaction**

Rolls back the current transaction.

---

```
<dsp:rollbackTransaction var-spec>
...
</dsp:rollbackTransaction>
```

---

### **Attributes**

#### ***var-spec***

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use var, you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
id	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has two properties:

- **success:** A Boolean property, specifies the status of the rollback transaction operation. A call to `isSuccess` obtains the property's setting.
- **exception:** Identifies the `Throwable` exception object produced when a transaction does not rollback properly. A call to `getException` returns the exception object.

## Usage Notes

`<dsp:rollbackTransaction>` causes the tasks associated with the current transaction to halt processing and return to their pre-transaction state. This tag tracks whether the rollback action is successful and when it is not, provides a `Throwable` exception object. This tag creates an EL variable that can be referenced by later tags, and provides access to the referenced object's `success` and `exception` properties.

## Example

```
<dsp:rollbackTransaction var="rollbackXA"/>
<c:choose>
  <c:when test="{rollbackXA.success}">
    Your band uniform order was canceled.
  </c:when>
  <c:otherwise>
    Your band uniform order could not be canceled. Here's why:
    <c:out value="{rollbackXA.exception}"/>
  </c:otherwise>
</c:choose>
```

In this example, if a particular transaction is successfully rolled back, the user sees the message `Your band uniform order was canceled`. If the transaction is not successfully rolled back, the user sees the second message string `Your band uniform order was canceled. Here's why:` and the exception.



## dsp:select

Defines a selection control—either a drop-down list or selection box..

---

```
<dsp: select bean="property-spec" [name=input-name]
  [nodefault="{true|false}"]
  [priority="integer-value"]
  [multiple="{true|false}"]
>

  dsp: option tags

</dsp: select>
```

---

### Attributes

#### **bean**

Specifies the property to update from the user's selection: its Nucleus path, component name, and property name. The property specification can also include a tag converter, such as date or null. For more information, see [Tag Converters](#) in Chapter 2.

#### **name**

Assigns a name to this drop-down list, which enables access to it during form processing. If no name is provided, the ATG platform automatically supplies one.

The name attribute provides a name to a drop-down list that is used during form processing. It is not necessary to include a name in dsp: select tags; the ATG platform automatically supplies one if none is specified.

#### **nodefault**

Unless you specify otherwise, the control displays the bean's current value as the initial selection.. To override this behavior, set the dsp: select tag's nodefault attribute together with the dsp: option tag's selected attribute.

When the bean property in the dsp: select tag holds a value that matches one or more options provided by the subordinate dsp: option tags, those values display as default selections. You can block this behavior by setting the nodefault attribute to true as a runtime expression, indicating that default values are not displayed.

#### **priority**

Specifies the priority of this control during form processing relative to other input controls. This attribute can be especially helpful when making changes to an item's repository ID and the properties it contains. If you first update the supplemental properties and then the repository ID, the new property values can be flushed when the repository ID change occurs.

For more information, see [Order of Tag Processing](#).

### **multiple**

If set to true, enables selection of multiple options. The default behavior enables selection of only one option. Set this attribute to true only if the target property is an array,

## **Usage Notes**

dsp: select lets you define drop-down lists or select boxes in JSPs much like the HTML tag select. List options are defined by multiple dsp: option tags.

This tag must be enclosed in a dsp: form, which can be provided in the same page or a parent page. See [Embedding Pages in Forms](#).

**Note:** dsp: select uses the lclass attribute in place of the cascading stylesheet class attribute to avoid using a Java reserved word in Java code.

### **Embedding custom attributes**

A dsp: select tag can embed one or more custom attributes through the [dsp: tagAttribute](#) tag, as follows:

---

```
<dsp: select ... >
  dsp: tagAttribute name="attr-name" value="value" />
</dsp: select/>
```

---

You can use [dsp: tagAttribute](#) tags in order to add attributes to the generated HTML beyond the fixed set of attributes supported by the DSP tag library.

## **Example**

```
<dsp: select bean="Student_01. Summerhobbies" multiple="true"
  nodefault="true" priority="10">
  <dsp: option value="hiking">hiking</dsp: option>
  <dsp: option value="biking">biking</dsp: option>
  <dsp: option value="swimming">swimming</dsp: option>
</dsp: select>
```

This example creates a select box of summer hobbies from which a user can select options: hiking, biking, or swimming. The user's selection, which is saved to the summerHobbies property of the Student\_01 component, can include multiple options (hiking and biking, for example) so the summerHobbies property must be an array. If the summerHobbies property already holds the values biking and swimming, for example, those values are not selected as defaults in the form because the nodefault attribute blocks that functionality. By indicating a high priority number, you ensure that this tag set is processed before others in the form.





## dsp:setTransactionRollbackOnly

Requires roll back of the current transaction.

```
<dsp: setTransacti onRol l backOnl y var-spec>
  ...
</dsp: setTransacti onRol l backOnl y>
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
var	Names an EL variable. When you use <code>var</code> , you can set its scope attribute to page, request, sessi on, or appl i cati on. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
i d	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has two properties:

- `success`: A Boolean property, specifies the status of the `setTransacti onRol l backOnl y` operation. A call to `isSuccess` obtains the property's setting.
- `excepti on`: Identifies the `Throwabl e` exception object produced when a transaction does not rollback properly. A call to `getExcepti on` returns the exception object.

### Usage Notes

`dsp: setTransacti onRol l backOnl y` specifies that when the current transaction ends, it must fail and roll back. This tag does not cause a transaction to roll back now, but rather instructs the transaction to end in a rollback action when it is prompted to end.

The tag's ability to set the current transaction to `setTransacti onRol l backOnl y` is saved to the named EL variable. You can see tag's success or failure through the `success` property and exceptions through the `excepti on` property.



## Example

```
<dsp:beginTransaction var="beginTransactionAdvancedAlgebra"/>
<dsp:toMap bean="/atg/samples/Student_01" var="student"/>

<c:choose>
  <c:when test="{!student.passedBeginningAlgebra}"/>
    You can't take advanced because you did not pass Beginning Algebra.
    <dsp:setTransactionRollbackOnly var="rollbackOnlyXA"/>
    <c:if test="{rollbackOnlyXA.exception != !empty}">
      The system is having trouble processing this information
      for the following reason:
      <c:out value="{rollbackOnlyXA.exception}"/>
    </c:if>
  </c:when>
  <c:otherwise>
    You may enroll in Advanced Algebra.
  </c:otherwise>
</c:choose>
<dsp:commitTransaction var="commitAdvancedAlgebra"/>
```

In this example, a transaction is used to manage the enrollment process for Advanced Algebra. After the transaction is created, the student's record is checked for successful completion of the beginning level Algebra class. The beginning level class is a prerequisite for enrolling in the advanced class, so the transactions are set to roll back. If an error occurs and `dsp:setTransactionRollbackOnly` does not set the transaction to end in a future rollback action, the exception is displayed. Otherwise, the `dsp:commitTransaction` tag tries to end the transaction.

## dsp:setvalue

Sets a bean property or page parameter.

```
<dsp:setvalue target-spec [source-spec] />
```

### Attributes

#### *target-spec*

Specifies the bean property or page parameter to set with one of the following attributes:

Attribute	Set to...
bean	Nucleus path, component name, and property name



param	Page parameter
-------	----------------

The property specification can also include a tag converter, such as date or null. For more information, see [Tag Converters](#) in Chapter 2.

**source-spec**

Specifies the value source as a [JavaBean component](#), [page parameter](#), or [static value](#), as follows:

beanval ue=" <i>property-spec</i> "	Specifies a property value, where <i>property-spec</i> includes a Nucleus path, component name, and property name. For example:  param="name" beanval ue="Student_01.name"
param=" <i>param-name</i> "	Specifies the value from the page parameter <i>param-name</i> . For example:  bean="Student_01.enrollmentDate" paramval ue="today"
val ue=" <i>static-value</i> "	Specifies a static value. For example:  bean="Student_01.juniorHighSchool " val ue="Roosevelt"  bean="Student_01.juniorHighSchool " val ue="true"

**Usage Notes**

dsp: setval ue lets you set a bean property or page parameter with a value copied from another bean property, page parameter, or constant. If no value is specified, the variable is set to null.

**Example**

```
<dsp:setval ue bean="Student_01.name" paramval ue="currentName"/>
<dsp:setval ue param="graduationDate" val ue="June 12, 2002"
date="M/dd/yyyy/">
```

This example assumes that the currentName page parameter has been previously set to a value. That value is copied to the name property of the Student\_01 component by the first dsp: setval ue tag. The second tag sets the constant June 12, 2002 to page parameter graduationDate and formats that date so it is saved as 6/12/2002.



## dsp:setxml

---

```
<dsp:setxml value="{true|false}" [useXmlParamDelimiter="{true|false}"] />
```

---

### Attributes

#### *value*

Determines how to format the remaining page:

- `true`: XML-style tags
- `false` (default): HTML-style tags

#### *useXmlParamDelimiter*

Specifies the delimiter used by `DynamoHttpServletRequest.addQueryParam` to separate query parameters that exist in the page after this tag:

- `true`: XML delimiter `&amp;`
- `false`: HTML delimiter `&`

If no delimiter value is specified by a `dsp:setxml` or `dsp:page` tag in this page or one of its parents, the ATG platform uses the delimiter associated with the page's MIME type.

### Usage Notes

`dsp:setxml` lets you specify whether to generate XML or HTML from tag attributes on a given page. By default, tags use the XML formatting:

```
<sometag selected="selected">
```

If generated as HTML, the attribute value is omitted and assumes `true` unless otherwise specified:

```
<sometag selected>
```

HTML tags do not require closing tags; XML tags require one of the following formats:

- `<sometag></sometag>`
- `<sometag />`

Based on tag placement, you might designate parts of a page to use HTML formatting and other parts to use XML. You can also set the output format for the entire page with the `xml` attribute of [dsp:page](#).

### Example

```
<dsp:tomap bean="BrowserAttributes" var="browser"/>
<c:if value="{browser.Mosaic}">
```



```
<dsp:setxml value="false" />
</c:if>
```

This example checks to see if the browser that made the request is a Mosaic browser. Because Mosaic is an old browser, it might have trouble handling XML-styled attributes so when a Mosaic browser makes a request, the `dsp:setxml` tag sets the attribute style to HTML.

## dsp:sort

Sorts the contents of a Collection or an array.

```
<dsp:sort var-spec values="sort-item"
  [comparator="comparator-obj"]
  [reverse="{true|false}"]
>
  ...
</dsp:sort>
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that stores the sorted output, defined with one of these attributes:

Attribute	Description
var	<p>Names an EL variable. When you use <code>var</code>, you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.</p> <p>By default, the values that are sorted into the <i>var</i>-specified object have the same data type as the source values. For example, sorting an array of strings produces another array of Strings.</p> <p>The EL variable has a number of properties that enable you to access the sorted output as a different data type. For more information, see Usage Notes below.</p>
id	Names a scripting variable that scriptlets and expressions can access at runtime.

#### *values*

Identifies the item to sort. This attribute can be set to items of the following data types:



- array of primitives
- array of objects
- Enumeration
- Iterator
- Collection
- Map

This attribute cannot be set to a Dynamic Bean or any items with Dynamic Bean properties to be used for sorting.

***comparator***

Set to an instance of `java.util.Comparator` that you define. Use a comparator to define how to sort elements in the first item in the `values` attribute.

***reverse***

After the item is sorted, determines whether to use the initial order (false) or reverse it (true). If you omit this attribute, the default is false.

**Usage Notes**

`dsp: sort` sorts the contents of a Collection or an array in the specified order. You can sort items of the following data types:

- Array of objects
- Array of primitives
- Collection
- Iterator
- Enumeration
- Map

The page processor sorts the contents of the specified item as follows, in descending order of precedence:

1. If the `comparator` attribute is set to a `java.util.Comparator` object, `dsp: sort` uses the sort instructions defined in the `Comparator` object.
2. `dsp: sort` uses the nested `dsp: orderBy` tags.
3. `dsp: sort` interprets the items in their natural order, when all items are of type `java.lang.Comparable`.

By default, the values that are sorted into the *var*-specified object have the same data type as the source values. For example, sorting an array of strings produces another array of Strings.



The EL variable has a number of properties that provide access to the sorted output as a different data type, shown in the table below. For example, the `sortedMap` property lets you render a `Collection` as a `Map`. The resultant `Map` holds the `Collection` items as keys.

**Note:** Rendering an item as a different data type slows site performance.

### ***Sort variable properties***

Sort property	Description
<code>sortedArray</code>	Renders the sorted item as an array.
<code>sortedBooleanArray</code>	Renders the sorted item as an array of Boolean objects.
<code>sortedByteArray</code>	Renders the sorted item as an array of Byte objects.
<code>sortedCharArray</code>	Renders the sorted item as an array of Char objects.
<code>sortedCollection</code>	Renders the sorted item as a Collection.
<code>sortedDoubleArray</code>	Renders the sorted item as an array of Double objects.
<code>sortedEnumeration</code>	Renders the sorted item as an Enumeration.
<code>sortedFloatArray</code>	Renders the sorted item as an array of Float objects.
<code>sortedIntArray</code>	Renders the sorted item as an array of Int objects.
<code>sortedIterator</code>	Renders the sorted item as an Iterator.
<code>sortedLongArray</code>	Renders the sorted item as an Array of Long objects.
<code>sortedMap</code>	Renders the sorted item as a Map.
<code>sortedShortArray</code>	Renders the sorted item as an array of Short objects.

The following table describes what happens when you convert items from one data type to another:

Original data type	Accessed as this data type	Example
Array of primitive values	Collection Iterator Enumeration Map array of objects	array of Strings > Collection: Collection of String objects
Collection array	Map	Enumeration of Longs > Map: Map with key entries that are Long objects



Original data type	Accessed as this data type	Example
Map	Collection Iterator Enumeration Map array of objects	Map > Collection:  Collection composed of Map key values
Collection Iterator Enumeration Map array of objects	Array of primitives	Iterator > Array of primitives:  Null value
Array of primitives	Array of primitives of a different data type	array of ints > array of shorts:  Array of short objects that are truncated versions of the original int values

### Example

```
<dsp:importbean bean="/atg/samples/AdvancedAlgebra" var="advAlgebra"/>
<dsp:sort var="sorter" values="{advAlgebra.students}">
  <dsp:orderBy property="lastName"/>

  <c:forEach var="student" begin="0" items="{sorter.sortedMapArray}">
    <dsp:tomap value="{student}" var="studentMap"/>
    <li>Name: <c:out value="{studentMap.lastName}"/>,
    <c:out value="{studentMap.firstName}"/>
  </c:forEach>
</dsp:sort>
```

This example organizes the student roster for Advanced Algebra. The AdvancedAlgebra component has a Students property that is a Map of student last names and first names. In this example, the students are organized alphabetically by last name. Then, each student's first name and last name are displayed.

## dsp:tagAttribute

Sets a custom attribute within a [dsp:input](#) or [dsp:select](#) tag.

```
<dsp:tagAttribute name="attr-name" value="value"/>
```





Attributes

*name*

The attribute name.

*value*

The value associated with the named attribute.

Usage Notes

`dsp:tagAttribute` can be embedded within `dsp:input` and `dsp:select` tags; it lets you add attributes to the generated HTML beyond the fixed set of attributes supported by the DSP tag library.

dsp:test

Sets a variable to reference a Collection, array, or Map in order to test its properties.

```
<dsp: test  var-spec  value=value-spec  />
```

Attributes

*var-spec*

An EL variable or scripting variable that is set to the object to test, defined with one of these attributes:

Attribute	Description
var	<p>Names an EL variable. When you use <code>var</code>, you can set its <code>scope</code> attribute to <code>page</code>, <code>request</code>, <code>session</code>, or <code>application</code>. The default scope is <code>page</code>. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.</p> <p>The EL variable has a number of properties that you can query—for example, the referenced object’s size or number of items. For information on available properties, see Usage Notes below.</p>
id	<p>Names a scripting variable that scriptlets and expressions can access at runtime.</p>

*value*

Sets the specified variable to a Collection, array, or Map.



## Usage Notes

dsp: test takes an object input and produces an object named by the specified variable. Other tags can query the object to find out information about the original input object, such as its size (empty, null, or number of items).

The following properties and runtime methods are available:

Property/Method	Description
isNull isNull	Returns true if the tested object has a value of 0.
isArray isArray	Returns true if the tested object is an array.
collection isCollection	Returns true if the tested object is a Collection.
arrayOrCollection isArrayOrCollection	Returns true if the tested object is an array or Collection.
size ssize	Tests the size of the object, returns one of the following: <ul style="list-style-type: none"><li>- Array, Collection, Map, or Enumeration objects: the object size.</li><li>- String objects: the String length.</li><li>- All other objects: -1</li></ul>
emptyValue isEmpty	Tests whether the object has a value, returns one of the following: <ul style="list-style-type: none"><li>- Array, Collection, Iterator, Map, or Enumeration objects: true if the object has zero or no objects.</li><li>- String objects: true if the String is empty or the value is null.</li></ul>
valueClass	Identifies the class of the tested object, returns null if the value attribute is set to null.
hashCode isHashCode	Returns the tested object's hash code. returns null if no hash code exists or the hash code is 0.

## Example

```
<dsp: tomap bean="Articles.current" var="currentArticles"/>
<dsp: test var="objectSize" value="{currentArticles}"/>
  Total article count for this week should be 10. We have
  <c: out value="{objectSize.size}"/> articles.
```



In this example, the current property of the `Articles` component is passed through the `dsp:test` tag. The object referenced by the EL variable `objectSize` holds information about `Articles.current` and so a call to its `size` property returns the size of the current array.

## dsp:textarea

---

```
<dsp:textarea bean=" property-spec"
  [name=" input-name" ]
  [default=" value" ]
  [priority=integer-value]

/>
```

---

- *property-spec*

Specifies a property value, where *property-spec* includes a Nucleus path, component name, and property name. For example:

```
param="name" beanvalue="Student_01.name"
```

### Attributes

#### **bean**

Specifies the Nucleus path, component name, and name of the property to update from the user-entered data. If this property has data, it displays in the multi-line text box as the default value unless the default attribute specifies another value.

The property specification can also include a tag converter, such as `date` or `null`. For more information, see [Tag Converters](#) in Chapter 2.

#### **name**

Assigns a name to this text box, which enables access to it during form processing. If no name is provided, the ATG platform automatically supplies one. For all input types other than `radio` and `checkbox`, the assigned name must be unique.

#### **default**

Specifies a default value to display in the text box. The default attribute can be set to a bean property, page parameter, or constant value.

You can omit the default attribute and instead embed default text (dynamic or static) between the start and end `dsp:textarea` tags. If you use both methods, the text in the default attribute has precedence.

**priority**

Specifies the priority of this text box field during form processing, relative to other input fields. This attribute can be especially helpful when making changes to an item's repository ID and the properties it contains. If you first update the supplemental properties and then the repository ID, the new property values can be flushed when the repository ID change occurs.

For more information, see [Order of Tag Processing](#).

**Usage Notes**

dsp: textarea lets you create an extended text box that for multi-line input. If you use start and end tag, they should be closely spaced together; any text between the two tags (including white space and line returns) is rendered as the text box's default value.

This tag must be enclosed in a dsp: form tag.

**Note:** The dsp: textarea tag uses the i class attribute instead of the cascading stylesheet class attribute to avoid using a Java reserved word in Java code.

**Example**

---

```
<dsp: textarea bean="Student_01. disciplinaryAction" default="None" />
```

---

This tag inserts a multi-line text box on a form page. The text box displays the default value None despite the data in the disciplinaryAction property. The default value or any changes made to it are saved to the disciplinaryAction property of the Student\_01 component.

## dsp:tomap

Saves a dynamic value to a variable.

---

```
<dsp: tomap var-spec source-spec recursive="{true|false}" />
```

---

**Attributes****var-spec**

An EL variable or scripting variable that is defined with one of these attributes:



Attribute	Description
var	Names an EL variable. When you use var, you can set its scope attribute to page, request, session, or application. The default scope is page. For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
id	Names a scripting variable that scriptlets and expressions can access at runtime.

### ***source-spec***

Specifies the variable to RepositoryItem bean, [page parameter](#), or [static value](#), as follows:

- `bean="property-spec"`  
Specifies a RepositoryItem property value, where *property-spec* includes a Nucleus path, component name, and (optionally) property name. For example:  
`bean="Student_01.name"`
- `param="param-name"`  
Specifies a page parameter.
- `value="static-value"`  
Specifies a static value.

### ***recursive***

If set to true, all underlying Dynamic Beans are wrapped as Maps and can be accessed through the EL variable. This can help simplify code that accesses multi-dimensional data. For example, if the RepositoryItem Profile has a property fundList which is itself a RepositoryItem, you can provide direct access to the fundList property as follows:

---

```
<dsp: tomap bean="atg/userprofiling/Profile" var="profile" recursive="true" />
I own these mutual funds: <c: out value="{profile.fundList}" />
```

---

If desired, you can access the underlying Dynamic Bean rather than its Map wrapper through the DynamicBeanMap property `_realObject`. Thus, given the previous example, you might access the Dynamic Bean fundList as follows:

```
{profile.fundList._realObject}
```

## **Usage Notes**

`dsp: tomap` provides access to Dynamic Bean components, standard JavaBean components, page parameters, and values by other tags that use EL or scripting variables. When you pass a parameter, component, or other value to a `dsp: tomap` tag, the object that is referenced by the EL or scripting variable treats the input as a Map. For example, when you pass in a component, the component appears as a Map in which the properties are keys and the property values are values.



You must use `dsp: tomap` in order to access a repository item through an EL expression.

### Example

```
<dsp: tomap var="currentProfile" bean="Profile" recursive="true"/>
  Hello <c: out value="{currentProfile.firstName}"/>
  Do you still live in <c: out value="{currentProfile.address.city}"/>
```

In this example, the `Profile` component is saved to the Map specified by `currentProfile`, so other tags can use EL to render this component. Because the `recursive` attribute is set to `true`, the same EL variable provides access to underlying components such as the Dynamic Bean address and its properties.

## dsp:transactionStatus

```
<dsp: beginTransaction var-spec>
  ...
</dsp: beginTransaction>
```

### Attributes

#### *var-spec*

An EL variable or scripting variable that is defined with one of these attributes:

Attribute	Description
<code>var</code>	Names an EL variable. When you use <code>var</code> , you can set its <code>scope</code> attribute to <code>page</code> , <code>request</code> , <code>session</code> , or <code>application</code> . The default scope is <code>page</code> . For more information, see <a href="#">EL Variable Scopes</a> in Chapter 2.
<code>id</code>	Names a scripting variable that scriptlets and expressions can access at runtime.

The variable has one of the following properties:

- `success`: A Boolean property, specifies whether the tag retrieved the transaction's status. A call to `isSuccess` obtains the property's setting.
- `exception`: Identifies the `Throwable` exception object produced when the transaction status is not accessed. A call to `getException` returns the exception object.
- One of the following Boolean status properties:



active  
committed  
committing  
markedRollback  
noTransaction  
prepared  
preparing  
rolledBack  
rollingBack  
unknown

See Usage Notes for more information on these status codes.

## Usage Notes

`dsp: transactionStatus` retrieves the status of the active transaction and saves that status the specified variable's `statusString` property. Other tags can query the variable for transaction status information. If the tag is unable to access status information, the `success` property is set to false and the `exception` property holds a `Throwable` exception.

The variable can be tested for one of the following Boolean properties:

### ***active***

Indicates whether the transaction is in its early stages of operation and has not received any instructions about how to end.

### ***committed***

Indicates whether tasks associated with the transaction executed, any changes made to a database(s) as a result were saved, and the transaction ended.

### ***committing***

Indicates whether tasks associated with the database are in the process of committing. It is most likely that the transaction involves multiple databases, of which some have committed data and others do not.

### ***markedRollback***

Indicates whether the transaction has been instructed that, when it is prompted to end, it must revert the database(s) to its pre-transaction state (rollback).

### ***noTransaction***

Indicates whether any transaction is associated with current request thread.

### ***prepared***

Indicates whether each database involved in the transaction has verified that it can commit the transaction without any errors.

### ***preparing***

Indicates whether each database involved in the transaction is investigating whether it can commit the tasks associated with the transaction.

### ***rolledback***

Indicates whether the tasks associated with the transaction did not complete and so no changes were made by the transaction. The transaction ended.

### ***rollingback***

Indicates whether that the transaction is in the process of undoing any tasks that it performed, so no changes are saved.

### ***unknown***

Indicates whether the transaction manager can recognize the status of any other status properties.

## **Example**

---

```
<dsp: transactionStatus var="statusXA"/>
  <c: choose>
    <c: when test="{statusXA.committed}">
      Congratulations! You will graduate this June!
    </c: when>

    <c: when test="{statusXA.rolledBack}">
      You will not graduate this year. Be sure to enroll in classes for
      next semester.
    </c: when>

    <c: otherwise>
      It takes 10 days to gather your records and process them. Check
      again in a few days.
    </c: otherwise>
  </c: choose>
```

---

In this example, the status of the transaction determines the message a user sees. The transaction status is set in the EL variable `statusXA` that is defined by `dsp: transactionStatus`. When the status is `rolledback` users see one message, when it is `committed` they see another. All other transaction statuses provide the same in process notification to their recipients.





## dsp:valueof

---

```
<dsp: valueof value-source>
  [default-value]
</dsp: valueof>
```

---

### Usage Notes

`dsp: valueof` renders a value in a bean property or page parameter. If desired, you can specify a default value between the open and close `dsp: valueof` tags, which displays only if the specified source is empty.

**Note:** If you do not intend to use a default, make sure no text, white space, or carriage return separates the open and close tags.

### Attributes

#### *value-source*

Specifies the source of the value to display as a [JavaBean component](#), [page parameter](#), or [static value](#), as follows:

- `bean="property-spec"`  
Displays the contents of the specified property, where *property-spec* includes the Nucleus path, component name, and property name.
- `param="param-name"`  
Displays the contents of the specified parameter *param-name*.
- `value="static-value"`  
Displays the specified static value

The value source specification can include a tag converter, such as date or null, so you can display the value in a specific format. For more information, see [Tag Converters](#) in Chapter 2.

### Example

```
<dsp: valueof bean="Student_01. age" number="##" />
<dsp: valueof param="currentName">Happy User</dsp: valueof>
```

In this example, the age property in the Student\_01 component is retrieved and converted into a format that renders a number like this: 35. The second tag gets the value of the currentName page parameter. If no value exists, it displays Happy User.





# Appendix B: ATG Servlet Beans

This appendix provides detailed reference entries for the servlet beans that ATG provides. It begins with a list of servlet beans grouped by functional area, and continues with the reference entries in alphabetical order.

Additional servlet beans are included with other ATG solutions. Your application can also use custom servlet beans written by programmers at your site.

## Servlet Beans by Category

The following sections summarize ATG servlet beans according to the following categories:

- [Standard servlet beans](#) can be used in any ATG application.
- [Database and repository access servlet beans](#) can be used in ATG applications that store data in a database or a repository.
- Multisite servlet beans let you establish site context and generate links to other registered sites.
- [XML servlet beans](#) can be used with ATG applications that include XML documents.
- [Transaction servlet beans](#) mark the bounds of a transaction within a JSP.
- [Personalization servlet beans](#) supply content to users based on information in their profiles.
- Business process tracking servlet beans

### Standard Servlet Beans

The following servlet beans can be used in any ATG application.

<b>Class Name</b>	atg.droplet. <a href="#">ArrayIncludesValue</a>
<b>Component</b>	/atg/dynamo/droplet/ArrayIncludesValue
<b>Function</b>	Verifies whether an array-type property includes a value.



<b>Class Name</b>	atg.droplet. <a href="#">Cache</a>
<b>Component</b>	/atg/dynamo/droplet/Cache
<b>Function</b>	Caches content that changes infrequently

<b>Class Name</b>	atg.droplet. <a href="#">Compare</a>
<b>Component</b>	/atg/dynamo/droplet/Compare
<b>Function</b>	Displays one of a set of possible outputs, depending on the relative value of its two input parameters

<b>Class Name</b>	atg.droplet. <a href="#">ComponentExists</a>
<b>Component</b>	/atg/dynamo/droplet/ComponentExists
<b>Function</b>	Tests whether a Nucleus path refers to a non-null object.

<b>Class Name</b>	atg.droplet. <a href="#">CurrencyConversionFormatter</a>
<b>Component</b>	/atg/dynamo/droplet/CurrencyConversionFormatter
<b>Function</b>	Displays a numeric value as a currency amount, and converts a value from one currency to another, formatting it based on the locale

<b>Class Name</b>	atg.droplet. <a href="#">CurrencyFormatter</a>
<b>Component</b>	/atg/dynamo/droplet/CurrencyFormatter
<b>Function</b>	Displays a numeric value as a currency amount, formatting it based on the locale

<b>Class Name</b>	atg.droplet. <a href="#">ErrorMessageForEach</a>
<b>Component</b>	/atg/dynamo/droplet/ErrorMessageForEach /atg/userprofiling/ProfileErrorMessageForEach /atg/demo/QuincyFunds/FormHandlers/RepositoryErrorMessageForEach
<b>Function</b>	Displays error messages that occur during a form submission



<b>Class Name</b>	atg.droplet. <a href="#">For</a>
<b>Component</b>	/atg/dynamo/droplet/For
<b>Function</b>	Displays a single output the number of times specified
<b>Class Name</b>	atg.droplet. <a href="#">ForEach</a>
<b>Component</b>	/atg/dynamo/droplet/ForEach
<b>Function</b>	Displays each element of an array
<b>Class Name</b>	atg.droplet. <a href="#">Format</a>
<b>Component</b>	/atg/dynamo/droplet/Format
<b>Function</b>	Displays one or more text values, formatting them based on locale
<b>Class Name</b>	atg.droplet. <a href="#">IsEmpty</a>
<b>Component</b>	/atg/dynamo/droplet/IsEmpty
<b>Function</b>	Displays one of two possible outputs, depending on whether its input parameter is empty
<b>Class Name</b>	atg.droplet. <a href="#">IsNull</a>
<b>Component</b>	/atg/dynamo/droplet/IsNull
<b>Function</b>	Displays one of two possible outputs, depending on whether its input parameter is null
<b>Class Name</b>	atg.droplet. <a href="#">Protocol Change</a>
<b>Component</b>	/atg/dynamo/droplet/Protocol Change
<b>Function</b>	Enables switching between HTTP and HTTPS protocols
<b>Class Name</b>	atg.droplet. <a href="#">Range</a>
<b>Component</b>	/atg/dynamo/droplet/Range
<b>Function</b>	Displays a subset of the elements of an array



<b>Class Name</b>	atg.droplet. <a href="#">Redirect</a>
<b>Component</b>	/atg/dynamo/droplet/Redirect
<b>Function</b>	Redirects the user to the specified page

<b>Class Name</b>	atg.droplet. <a href="#">Switch</a>
<b>Component</b>	/atg/dynamo/droplet/Switch
<b>Function</b>	Displays one of a set of possible outputs, depending on the value of its input parameter

<b>Class Name</b>	atg.droplet. <a href="#">TableForEach</a>
<b>Component</b>	/atg/dynamo/droplet/TableForEach
<b>Function</b>	Displays each element of an array, arranging the output in a two-dimensional format

<b>Class Name</b>	atg.droplet. <a href="#">TableRange</a>
<b>Component</b>	/atg/dynamo/droplet/TableRange
<b>Function</b>	Displays a subset of the elements of an array, arranging the output in a two-dimensional format

## Database and Repository Access Servlet Beans

The following servlet beans enable access to a database or repository.

<b>Class Name</b>	atg.repository.seo. <a href="#">CanonicalItemLink</a>
<b>Component</b>	Not provided with the ATG platform
<b>Function</b>	Takes a repository item as input and generates the canonical URL for the page associated with that item.

<b>Class Name</b>	atg.droplet. <a href="#">ContentDroplet</a>
<b>Component</b>	Not provided
<b>Function</b>	



<b>Class Name</b>	atg.droplet. <a href="#">ContentFolderView</a>
<b>Component</b>	Not provided
<b>Function</b>	
<hr/>	
<b>Class Name</b>	atg.repository.seo. <a href="#">ItemLink</a>
<b>Component</b>	Not provided with the ATG platform
<b>Function</b>	Takes a repository item as input and generates either a static or dynamic URL, depending on whether the page it is on is being viewed by a human visitor or a Web spider.
<hr/>	
<b>Class Name</b>	atg.repository.servlet. <a href="#">ItemLookupDroplet</a>
<b>Component</b>	/atg/dynamo/droplet/ItemLookupDroplet
<b>Function</b>	Looks up an item in one or more repositories, based on the item's ID, and renders the item on the page
<hr/>	
<b>Class Name</b>	atg.repository.servlet. <a href="#">NamedQueryForEach</a>
<b>Component</b>	Not provided with the ATG platform
<b>Function</b>	Constructs an RQL query and renders its output parameter once for each element returned by the query
<hr/>	
<b>Class Name</b>	atg.repository.servlet. <a href="#">NavHistoryCollector</a>
<b>Component</b>	Not provided with the ATG platform
<b>Function</b>	Tracks each page that a user visits and makes it easy for that user to backtrack and return to pages he or she has already visited
<hr/>	
<b>Class Name</b>	atg.service.pipeline.servlet. <a href="#">PipelineChainInvocation</a>
<b>Component</b>	/atg/dynamo/droplet/PipelineChainInvocation
<b>Function</b>	Initiates a pipeline thread



<b>Class Name</b>	atg.repository.servlet. <a href="#">PossibleValues</a>
<b>Component</b>	/atg/dynamo/droplet/PossibleValues
<b>Function</b>	Returns a list of repository items of the specified item type

<b>Class Name</b>	atg.repository.servlet. <a href="#">RQLQueryForEach</a>
<b>Component</b>	/atg/dynamo/droplet/RQLQueryForEach
<b>Function</b>	Constructs an RQL query and renders its output parameter once for each element returned by the query

<b>Class Name</b>	atg.repository.servlet. <a href="#">RQLQueryRange</a>
<b>Component</b>	/atg/dynamo/droplet/RQLQueryRange
<b>Function</b>	Constructs an RQL query of an SQL database and renders its output parameter for a selected subset of the elements returned by the query

<b>Class Name</b>	atg.droplet.sql. <a href="#">SQLQueryForEach</a>
<b>Component</b>	/atg/dynamo/droplet/SQLQueryForEach
<b>Function</b>	Constructs a query of an SQL database and renders its output parameter once for each row returned by the database query

<b>Class Name</b>	atg.droplet.sql. <a href="#">SQLQueryRange</a>
<b>Component</b>	/atg/dynamo/droplet/SQLQueryRange
<b>Function</b>	Constructs a query of an SQL database and renders its output parameter once for each of a specified range of the rows returned by the database query

## Multisite Servlet Beans

<b>Class Name</b>	atg.droplet.multisite. <a href="#">GetSiteDroplet</a>
<b>Component</b>	/atg/dynamo/droplet/multisite/GetSiteDroplet
<b>Function</b>	Gets the site associated with a site ID.





<b>Class Name</b>	atg.droplet.multiplesite. <a href="#">SiteContextDroplet</a>
<b>Component</b>	/atg/dynamo/droplet/multiplesite/SiteContextDroplet
<b>Function</b>	Establishes a site context for this servlet bean's output.

<b>Class Name</b>	atg.droplet.multiplesite. <a href="#">SiteLinkDroplet</a>
<b>Component</b>	/atg/dynamo/droplet/multiplesite/SiteLinkDroplet
<b>Function</b>	Generates a link to a registered site.

<b>Class Name</b>	atg.droplet.multiplesite. <a href="#">SitesShareableDroplet</a>
<b>Component</b>	/atg/dynamo/droplet/multiplesite/SitesShareableDroplet
<b>Function</b>	Tests whether a given shareable type is shared between the current site and another site.

## XML Servlet Beans

The following servlet beans are used with XML documents.

<b>Class Name</b>	atg.droplet.xml. <a href="#">NodeForEach</a>
<b>Component</b>	/atg/dynamo/droplet/xml/NodeForEach
<b>Function</b>	Given a DOM node, selects all nodes that match a specified pattern and iterates over each selected node

<b>Class Name</b>	atg.droplet.xml. <a href="#">NodeMatch</a>
<b>Component</b>	/atg/dynamo/droplet/xml/NodeMatch
<b>Function</b>	Given a DOM node, selects the next node that matches a specified pattern

<b>Class Name</b>	atg.droplet.xml.DOM
<b>Component</b>	/atg/dynamo/droplet/xml/XMLToDOM
<b>Function</b>	Parses an XML document and transforms it into a DOM document



<b>Class Name</b>	atg.droplet.xml.XMLTransform
<b>Component</b>	/atg/dynamo/droplet/xml/XMLTransform
<b>Function</b>	Given an XML document and an XSLT or JSP template, transforms and outputs the XML document

## Transaction Servlet Beans

The following servlet beans are used for transaction demarcation.

<b>Class Name</b>	atg.dtm.EndTransacti onDropl et
<b>Component</b>	/atg/dynamo/transacti on/dropl et/EndTransacti on
<b>Function</b>	Commits or rolls back the current transaction

<b>Class Name</b>	atg.dtm.Transacti onDropl et
<b>Component</b>	/atg/dynamo/transacti on/dropl et/Transacti on
<b>Function</b>	Marks the bounds of a transaction within a JSP

## Personalization Servlet Beans

The following servlet beans are for use with user profiles.

<b>Class Name</b>	atg.markers.userprofil ling.dropl et.AddMarkerToProfi le
<b>Component</b>	/atg/markers/userprofil ling/dropl et/AddMarkerToProfi leDropl et
<b>Function</b>	Add a profile marker to a profile

<b>Class Name</b>	atg.markers.userprofil ling.dropl et.Profi leHasLastMarker
<b>Component</b>	/atg/markers/userprofil ling/dropl et/Profi leHasLastMarkerDropl et
<b>Function</b>	Locates the last profile marker added to a profile



<b>Class Name</b>	atg.markers.userprofil ling.droplet. <a href="#">ProfileHasLastMarkerWithKey</a>
<b>Component</b>	/atg/markers/userprofil ling/droplet/ProfileHasLastMarkerWithKeyDroplet
<b>Function</b>	Locates the last profile marker with a particular key that was added to a profile

<b>Class Name</b>	atg.markers.userprofil ling.droplet. <a href="#">ProfileHasMarker</a>
<b>Component</b>	/atg/markers/userprofil ling/droplet/ProfileHasMarkerDroplet
<b>Function</b>	Determines whether a profile has a profile marker

<b>Class Name</b>	atg.markers.userprofil ling.droplet. <a href="#">RemoveAllMarkersFromProfile</a>
<b>Component</b>	/atg/markers/userprofil ling/droplet/RemoveAllMarkersFromProfileDroplet
<b>Function</b>	Removes all profile markers from a profile

<b>Class Name</b>	atg.markers.userprofil ling.droplet. <a href="#">RemoveMarkersFromProfile</a>
<b>Component</b>	/atg/markers/userprofil ling/droplet/RemoveMarkersFromProfileDroplet
<b>Function</b>	Removes a profile marker from a profile

<b>Class Name</b>	atg.servi ce.col lecti ons.fi lter.droplet. <a href="#">CollectionFilter</a>
<b>Component</b>	/atg/col lecti ons/fi lter/droplet/StartEndDateFilterDroplet
<b>Function</b>	Filters objects in a collection

<b>Class Name</b>	atg.userprofil ling. <a href="#">GetDirectoryPrincipal</a>
<b>Component</b>	/atg/userprofil ling/GetDirectoryPrincipal
<b>Function</b>	Returns the DirectoryPrincipal for a specified type and ID

<b>Class Name</b>	atg.userprofil ling. <a href="#">HasEffectivePrincipal</a>
<b>Component</b>	/atg/userprofil ling/HasEffectivePrincipal
<b>Function</b>	Fires a content event if a user has the appropriate Principal (identity)



<b>Class Name</b>	atg.userprofil ing. <a href="#">HasFunction</a>
<b>Component</b>	/atg/userdi rectory/dropl et/HasFunction
<b>Function</b>	Tests whether a given user has the specified role.
<b>Class Name</b>	atg.userprofil ing. <a href="#">PageEventTri ggerDropl et</a>
<b>Component</b>	/atg/userprofil ing/SendPageEvent
<b>Function</b>	Sends page viewed events for the current page being viewed
<b>Class Name</b>	atg.targeti ng. <a href="#">Rul eBasedReposi toryI temGroupFi lter</a>
<b>Component</b>	/atg/targeti ng/Rul eBasedReposi toryI temGroupFi lter
<b>Function</b>	Renders a collection of Reposi toryI tems that belong to the specified Rul eBasedReposi toryI temGroup
<b>Class Name</b>	atg.targeti ng. <a href="#">Reposi toryLookup</a>
<b>Component</b>	/atg/targeti ng/Reposi toryLookup
<b>Function</b>	Looks up an item in a specific repository, based on the item's ID, and renders the item on the page
<b>Class Name</b>	atg.targeti ng. <a href="#">Targeti ngArray</a>
<b>Component</b>	/atg/targeti ng/Targeti ngArray
<b>Function</b>	Performs a targeting operation and passes the results to another servlet bean for display
<b>Class Name</b>	atg.targeti ng. <a href="#">Targeti ngFi rst</a>
<b>Component</b>	/atg/targeti ng/Targeti ngFi rst
<b>Function</b>	Performs a targeting operation and displays the first <i>n</i> items returned by the targeter, where <i>n</i> is a number you specify



<b>Class Name</b>	atg.targeting.TargetingForEach
<b>Component</b>	/atg/targeting/TargetingForEach
<b>Function</b>	Performs a targeting operation and displays all items returned by the targeter
<b>Class Name</b>	atg.targeting.TargetingRandom
<b>Component</b>	/atg/targeting/TargetingRandom
<b>Function</b>	Performs a targeting operation and randomly selects and displays <i>n</i> items returned by the targeter, where <i>n</i> is a number you specify
<b>Class Name</b>	atg.targeting.TargetingRange
<b>Component</b>	/atg/targeting/TargetingRange
<b>Function</b>	Performs a targeting operation and displays a subset of the items returned by the targeter, specified as a range
<b>Class Name</b>	atg.userdirectory.droplet.TargetPrincipalSDroplet
<b>Component</b>	/atg/userdirectory/droplet/TargetPrincipalS
<b>Function</b>	Locates all organizations where a user has the specified role.
<b>Class Name</b>	atg.userdirectory.droplet.UserListDroplet
<b>Component</b>	atg/userdirectory/droplet/UserList
<b>Function</b>	Searches for all users assigned to an organization
<b>Class Name</b>	atg.userdirectory.droplet.ViewPrincipalSDroplet
<b>Component</b>	/atg/userdirectory/droplet/ViewPrincipalS
<b>Function</b>	Obtains a user's roles or organizations
<b>Class Name</b>	atg.workflow.servlet.WorkflowInstanceQueryDroplet
<b>Component</b>	None provided
<b>Function</b>	Returns information about a specified set of workflow instances



<b>Class Name</b>	atg.workflow.servlet. <a href="#">WorkflowTaskQueryDropLet</a>
<b>Component</b>	None provided
<b>Function</b>	Returns information about a specified set of workflow tasks

## Business Process Tracking Servlet Beans

The following servlet beans work with business process tracking.

<b>Class Name</b>	atg.markers.bp.dropLet. <a href="#">AddBusinessProcessStage</a>
<b>Component</b>	/atg/markers/bp/dropLet/AddBusinessProcessStageDropLet
<b>Function</b>	Adds a marker when a business object reaches a new business process stage

<b>Class Name</b>	atg.markers.bp.dropLet. <a href="#">HasBusinessProcessStage</a>
<b>Component</b>	/atg/markers/bp/dropLet/HasBusinessProcessStageDropLet
<b>Function</b>	Tests whether a business object has reached a specified business process stage

<b>Class Name</b>	atg.markers.bp.dropLet. <a href="#">MostRecentBusinessProcessStage</a>
<b>Component</b>	/atg/markers/bp/dropLet/MostRecentBusinessProcessStageDropLet
<b>Function</b>	Tests whether the business process stage most recently reached by a business object matches the specified stage

<b>Class Name</b>	atg.markers.bp.dropLet. <a href="#">RemoveBusinessProcessStage</a>
<b>Component</b>	/atg/markers/bp/dropLet/RemoveBusinessProcessStageDropLet
<b>Function</b>	Removes a specified business process stage

## AddBusinessProcessStage

Adds a marker when a business object reaches a new business process stage.



<b>Class Name</b>	atg.markers.bp.droplet.AddBusinessProcessStage
<b>Component</b>	/atg/markers/bp/droplet/AddBusinessProcessStageDroplet

## Usage Notes

AddBusinessProcessStage marks a new stage being reached in a business process, based on a business process name and business process stage.

## Required Input Parameters

### ***businessProcessStage***

The stage within the business process.

## Optional Input Parameters

### ***businessProcessName***

The name of the business process. If omitted, the default value is obtained from the servlet bean's defaultBusinessProcessName property. Setting the defaultBusinessProcessName property lets you create instances of the servlet bean that are specific to a single business process.

## Output Parameters

### ***errorMsg***

The failure error message.

## Open Parameters

### ***output***

Rendered on success.

### ***error***

Rendered on error.

## Example

```
<dsp:droplet name="AddBusinessProcessStage">
  <dsp:param name="businessProcessName" value="ShoppingProcess"/>
  <dsp:param name="businessProcessStage" value="ShippingPrepared"/>
</dsp:droplet>
```



## AddMarkerToProfile

Add a profile marker to a profile.

<b>Class Name</b>	atg.markers.userprofiling.droplet.AddMarkerToProfile
<b>Component</b>	/atg/markers/userprofiling/droplet/AddMarkerToProfileDroplet

### Required Input Parameters

#### *key*

The value to save to the marker's key property. The profile marker key property holds a string that represents a similarity among a grouping of profile markers. For example, key can represent a type of profile marker or the circumstances under which the marker is assigned.

### Optional Input Parameters

#### *itemId*

The ID of the profile item the marker is assigned to. If omitted, the value indicated in the item input parameter is used.

#### *item*

The profile item that the marker is assigned to. If omitted, the active profile is used.

#### *value*

The value that is saved to the marker's value property. The profile marker value property holds a string that is related to the key property.

#### *data*

The value that is saved to the marker's data property. The profile marker data property holds a string that is related to the key and value properties.

#### *extendedProperties*

A map that specifies any property values for marker properties other than key, value, and data that is saved to the new marker. The marker property is saved to the map's key and its property value is set to the map's value.

#### *duplicationMode*

The duplication policy used by the Profile Marker Manager. Options include ALLOW\_DUPLICATES, REPLACE\_DUPLICATES, and NO\_DUPLICATES. If a profile has a marker and a Profile Marker Manager attempts to add an identical marker to it, a mode of REPLACE\_DUPLICATES causes the existing marker to be





replaced by the new one, while a mode of NO\_DUPLICATIONS keeps the existing marker and discards the new one. If this parameter is omitted, the servlet bean uses the default mode provided in its defaultDuplicationMode property.

For more information on duplication modes, see the *Configuring the Profile Marker Manager* section of the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

### ***markerManager***

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager specified in the servlet bean's repositoryMarkerManager property is used.

### ***markerPropertyName***

The property on the profile that stores the markers created by this servlet bean. If omitted, the servlet bean uses the default specified in its defaultMarkerPropertyName property.

### ***markerItemType***

The type of RepositoryItem used by the marker. If omitted, the servlet bean uses the default specified in its defaultMarkerItemType property.

### ***markedItemType***

The type of RepositoryItem that receives a marker. If omitted, the servlet bean uses the default specified in its defaultMarkedItemType property.

## **Output Parameters**

### ***marker***

Contains the profile marker created by the servlet bean.

### ***errorMsg***

Contains any error messages generated during the servlet bean's execution.

## **Open Parameters**

### ***output***

Rendered when the marker is added to the profile.

### ***error***

Rendered when a marker should be created and attached to a profile, but one or both actions are not completed successfully.



## Usage Notes

AddMarkerToProfile communicates with the Profile Marker Manager to create a profile marker and attach it to the appropriate profile, based on the marker and profile information you specify.

For more information about attaching profile markers to profiles, see the *Marking a User Profile* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

## Example

The following example shows how to add a profile marker to the active user profile. After the marker is added, the user sees a message encouraging the use of travel site A. Because the message is displayed only when the marker is added and the duplication mode prevents the same marker from being added twice, users see the encouragement message only once.

```
<dsp: droplet
  name="/atg/markers/userprofiling/droplet/AddMarkerToProfileDroplet">
  <dsp: param name="key" value="partner"/>
  <dsp: param name="value" value="travel Site A"/>
  <dsp: param name="duplicationMode" value="NO_DUPLICATES"/>

  <dsp: oparam name="output">
    We love travel site A! Check out our joint discounts.
  </dsp: oparam>

</dsp: droplet>
```

## BeanProperty

Sets any property value by dynamically specifying the property and value to update.

<b>Class Name</b>	atg.droplet.BeanProperty
<b>Component</b>	/atg/dynamo/droplet/BeanProperty

## Required Input Parameters

### *bean*

The target bean.

***propertyName***

The target property in bean. The value of this parameter is usually a page parameter that is already set to the desired property.

**Optional Input Parameters*****propertyValue***

The value to set on the property specified by `propertyName`, either static text or a dynamic page parameter.

**Output Parameters*****propertyValue***

The current value of the specified bean property.

**Open Parameters*****output***

Lets you use the value in `propertyValue` for another operation, such as displaying it through the user interface or passing it to another bean.

**Usage Notes**

BeanProperty provides the ability to get or set a property value without knowing in advance which property you are dealing with. Although this servlet bean makes both set and get methods available, it is typically used to get current property values.

Use this bean to display a series of property values that a user defines. Because you do not know which properties the user might, your code must parse through the user choices (using a [ForEach](#) droplet, for example) and display the value of each property. Use BeanProperty to collect user-defined properties and process them through the [ForEach](#) bean as an element page parameter.

To use BeanProperty to get a property value, you might need to define these parameters:

- bean
- propertyName
- propertyValue (output parameter)
- output

To use BeanProperty to set a property value, you define the following parameters:

- bean
- propertyName
- propertyValue (input parameter)

## Examples

### *Getting property values*

Consider a situation where you want to allow users to compare the properties of two products. The users need to select which products and which properties they want to view. Assume a user specified one product and its properties, which were saved to the `myProductBean` and `compareProperties` page parameters respectively. The following example shows how to code the ATG platform to render one product's properties on a page:

---

```
<dsp: droplet name="/atg/dynamo/droplet/BeanProperty">
  <dsp: param name="bean" param="myProductBean"/>
  <dsp: param name="propertyName" value="compareProperties"/>
  <dsp: oparam name="output">
    These are the product characteristics you wanted to see:
    <p><dsp: valueof param="propertyValue">No characteristics have been
      specified</dsp: valueof>
    </dsp: oparam>
  </dsp: droplet>
```

---

In this example, the ATG platform determines the selected product using `myProductBean` and properties using `compareProperties` and displays the properties values after the text  
Here are the product characteristics you wanted to compare.

In order to display the properties values in a specified order, add a `ForEach` bean. The following code alphabetizes the properties values and displays the first five.

---

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param name="array" param="compareProperties"/>
  <dsp: param name="count" value="5"/>
  <dsp: param name="sortProperties" value="+"/>

  <dsp: oparam name="outputStart">
    Here are the properties you asked about: <br/>
  </dsp: oparam>

  <dsp: oparam name="output">
    <dsp: droplet name="/atg/dynamo/droplet/BeanProperty">
      <dsp: param name="bean" param="myProductBean"/>
      <dsp: param name="propertyName" param="element"/>
      <dsp: oparam name="output">
        <dsp: valueof param="propertyName"/>: <dsp: valueof
          param="propertyValue">not set</dsp: valueof><br/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>

  <dsp: oparam name="empty">
```



```
You did not select any properties to display.  
</dsp: oparam>  
</dsp: droplet>
```

---

### Setting properties

A Web site that sells music wants to keep track of the genres that interest users. When a user browses a music page, that page's musical genre is temporarily saved to the `currentGenre` parameter; you also want to store that genre name and browsing status in a user's Profile component property. To do this, you create a Map Profile property `genresBrowsed` that has a genre set to true if it was browsed. BeanProperty updates Profile. `genresBrowsed` Map as follows:

- Copies the genre in `currentGenre` to the Profile. `genresBrowsed` property.
- Sets that genre's browsed status to true.

```
<dsp: droplet name="/atg/dynamo/BeanProperty">  
  <dsp: param bean="Profile.genresBrowsed" name="bean" />  
  <dsp: param name="propertyName" param="genre" />  
  <dsp: param name="propertyValue" value="true" />  
</dsp: droplet>
```

---

## ArrayIncludesValue

Verifies whether an array-type property includes a value.

<b>Class Name</b>	atg.droplet.ArrayIncludesValue
<b>Component</b>	/atg/dynamo/droplet/ArrayIncludesValue

### Required Input Parameters

#### *array*

The multi-value property to test, where array can reference one of the following:

- Array
- Collection, one of: Vector, List, Set
- Iterator
- Enumeration
- Map
- Dictionary

***value***

The value to find in the array-specified property.

**Open Parameters*****true***

Rendered if *value* is found.

***false***

Rendered if *value* cannot be found.

## Cache

Caches content that changes infrequently.

<b>Class Name</b>	atg.droplet.Cache
<b>Component</b>	/atg/dynamo/droplet/Cache

**Required Input Parameters*****key***

Lets you have more than one view of content based on a value that uniquely defines the view of the content. For example, if content is displayed one way for members and another for non-members, you can pass in the value of the member trait as the key parameter.

**Optional Input Parameters*****hasNoURLs***

Determines how cached URLs are rendered for future requests.

- *false*: Subsequent requests for the cached content causes URLs to be rewritten on the fly, assuming URL rewriting is enabled.
- *true*: URLs are saved and rendered exactly as they are currently (without session or request IDs) regardless of whether URL rewriting is enabled.

***cacheCheckSeconds***

The interval after content is cached until the cached is regenerated. If omitted, the interval is set from the default `cacheCheckSeconds` property in the Cache servlet bean's properties file.



## Open Parameters

### *output*

The code enclosed by the output open parameter is cached.

## Usage Notes

Cache can help reduce repetitive time-consuming tasks by keeping content in temporary memory. When you use Cache, you specify the content to cache and how frequently to clear the cache. Cached URLs do not retain session or request IDs.

### *Which content to cache*

Caching data resolves many tasks into one. Consider a [Switch](#) servlet bean that returns one set of Repository Items to females and another set to males. Each time that code is executed two time-consuming tasks occur: the [Switch](#) servlet bean is processed and the outcome repository Item is retrieved. Embedding the [Switch](#) servlet bean in Cache causes the following to occur in future executions: DAF queries only the gender property of the Profile component (which is stored locally) and provides the appropriate content (also stored locally).

The following example is an ideal candidate for use with the Cache servlet bean because it involves expensive operations—page computations and querying the database.

In determining whether the Cache servlet bean is appropriate for a given situation, consider the following:

- When retrieving data from the data base, is the retrieved data rendered for many users? Cache can limit the number of times the database is queried. Any code in which the database is not queried should not be cached.

That said, caching might not be appropriate for all data retrieved from the database, especially when the data is accessed through a personalization servlet bean, such as retrieving images for all items purchased by a particular user. It is likely that these items are different for most users so the cached data is not reused.

- When processing JSP code, is the output useful for many users? Actions that are personalized to users, such as displaying a user name or a URL that includes a session ID, are an inefficient use of resources because they require processing to incorporate the dynamic elements, processing that occurs each time the cached code is rendered. Code that invokes personalized results is not a suitable candidate for the Cache servlet bean.
- Does using Cache conserve more resources than it expends? Caching a string is wasteful because it takes more time to cache it than to render it many times.

In summary, use Cache to hold content that is expensive to compute, so the data to cache takes longer to process on its own than it takes to render Cache and resolve its key parameter for that data. A common example is code that renders content from the database for a large subset of your user community. Although you might want Cache to involve some personalization (determining that subset), if you also want to cache programming code, make sure the output is saved by Cache. Avoid Cache in these circumstances:



1. When the content to cache is static.
2. When it takes more time to cache the content than to render the content.
3. When the cached content is not applicable for many users.

It is a good idea to measure your page performance with and without Cache as the final determination of its efficiency for a given circumstance.

### ***Clearing the cache***

You can determine how often data is flushed for a given Cache instance on a JSP or for all instances of Cache. To remove cached content associated with a particular instance of Cache, set the `cacheCheckSeconds` on input parameter in the Cache instance to the frequency by which associated data should be expired. If you omit this parameter, the `Cache.defaultCacheCheckSeconds` property is used (default value is 60 seconds).

The `Cache.purgeCacheSeconds` property determines how often content cached by any Cache servlet bean is flushed. The default is 21600 seconds (6 hours). Cache purging also occurs when a JSP is removed or recompiled.

### **Example**

The following example demonstrates a simple implementation of Cache that displays gender-specific articles to users. The fictitious `GenderSpecificArticles` servlet bean determines the current user's gender and displays one set of articles to men and a different set to women.

The following JSP code renders gender-specific content:

---

```
<dsp:droplet name="GenderSpecificArticles">
</dsp:droplet>
```

---

Content for women looks like this:

---

```
<a href="womensarticle1.html">women's article 1</a>
<a href="womensarticle2.html">women's article 2</a>
```

---

Content for men looks like this:

---

```
<a href="mensarticle1.html">men's article 1</a>
<a href="mensarticle2.html">men's article 2</a>
```

---

The following example caches the preceding code:

---

```
<dsp:droplet name="/atg/dynamo/droplet/Cache">
  <dsp:param bean="Profile.gender" name="key"/>
```





```
<dsp:oparam name="output">
  <dsp:droplet name="GenderSpecificArticles">
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

The Cache servlet bean defines `male` and `female` as cache keys. The `male` key is set to the following string:

```
"<a href="mensarticle1.html">men's article 1</a>
<a href="mensarticle2.html">men's article 2</a>"
```

The `female` key is set to the following string:

```
"<a href="womensarticle1.html">women's article 1</a>
<a href="womensarticle2.html">women's article 2</a>"
```

The first time the JSP holding the Cache servlet bean is rendered, the `GenderSpecificArticles` servlet bean executes, but future requests for this page simply look up the result of `GenderSpecificArticles` in the cache and display the appropriate string.

## CanonicalItemLink

Takes a repository item as input and generates the canonical URL for the page associated with that item.

<b>Class Name</b>	atg.repository.seo.CanonicalItemLink
<b>Component</b>	Not provided with the ATG platform

### Required Input Parameters

Specify the repository item with one of the following input parameters:

- `item`
- `id`

#### *item*

The target repository item.

***id***

The repository ID of the target repository item. If you specify this input parameter, also specify two additional parameters:

- `itemDescriptorName`
- `repository` or `repositoryName`

***itemDescriptorName***

The repository item's item descriptor.

***repository***

The repository that contains the item.

***repositoryName***

The name of the repository that contains the item.

**Output Parameters*****url***

The canonical URL generated by the servlet bean.

***errorMessage***

The error message produced if a problem occurs generating the URL.

**Open Parameters*****output***

Rendered if the URL is generated successfully.

***error***

Rendered if a problem occurs when generating the URL.

**Usage Notes**

`CanonicalItemLink` takes a repository item as input and generates the canonical URL for the page associated with that item.

You can specify the repository item in one of two ways:

- Use the `item` input parameter to pass in the actual repository item. In this case, the item is typically the output of another servlet bean, such as `RepositoryLookup` or `ItemLookupDropIt`.



- Use the `id` and `itemDescriptorName` parameters, and either the `repository` or `repositoryName` parameter, to provide the information needed to uniquely identify the repository item.

For more information about `CanonicalItemLink`, see the discussion of Search Engine Optimization in the [ATG Programming Guide](#).

## Example

The following example illustrates using a `CanonicalItemLink` servlet bean on a product detail page to render a `link` tag specifying the page's canonical URL.

---

```
<dsp: droplet name="/atg/repository/seo/CanonicalItemLink">
  <dsp: param name="id" param="productId"/>
  <dsp: param name="itemDescriptorName" value="product"/>
  <dsp: param name="repositoryName"
    value="/atg/commerce/catalog/ProductCatalog"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="pageUrl" param="url" vartype="java.lang.String"/>
    <link rel="canonical" href="{pageUrl}"/>
  </dsp: oparam>
</dsp: droplet>
```

---

## CollectionFilter

Filters objects in a collection.

<b>Class Name</b>	atg.service.collections.filter.droplet.CollectionFilter
<b>Component(s)</b>	/atg/collections/filter/droplet/StartDateFilterDroplet

## Required Input Parameters

### *collection*

Identifies the collection of objects that require filtering.

## Optional Input Parameters

### *collectionIdentifierKey*

Specifies a string that represents the collection to filter. Each unique collection should have a unique string. This parameter is required in order to cache filtered content.

***filter***

Refers to the component responsible for performing the filter operation. The default value provided to `StartEndDateFilterDropLet` for this parameter is `atg/registry/CollectionFilters/StartEndDateFilter`.

***profile***

Points to a `ProfileRepositoryItem`. Use this parameter when the filtering condition is based on profile information. If no value is provided to this parameter, the `Profile` associated with the active session is used. None of the implementations of `CollectionFilter` provided with the ATG platform use this parameter.

**Output Parameters*****filteredCollection***

Holds a collection of items that satisfy the filter criteria.

***errorMsg***

Holds error messages generated during the servlet bean execution.

**Open Parameters*****output***

Rendered when the filter component completes processing.

***empty***

Rendered if the resultant collection is empty.

***error***

Rendered if an error occurs.

**Usage Notes**

`CollectionFilter` uses collection filtering components to reduce objects in a collection. The resulting collection is accessed by an output parameter. The `StartEndDateFilterDropLet` instance passes the collection to the `StartEndDateFilter` so it can determine which items in the collection are active for the current date.

Two properties, `consultCache` and `updateCache`, govern whether `CollectionFilter` checks the cache for content and saves the resultant content to the cache respectively. These properties are set to true by default. There are other settings you need to configure to enable caching for `StartEndDateFilter` on the collection filtering component itself.

The `CollectionFilter` servlet bean class `filter` property can be set to a collection filter component, which serves as the default value for the `filter` input parameter.



See the *Filtering Collections* chapter of the [ATG Personalization Programming Guide](#) for more information on filtering collections.

## Example

In this example, all articles that are started and not ended for the current day are displayed.

---

```
<dsp: tomap bean="/atg/dynamo/service/CurrentDate" var="date" />

<dsp: droplet name="/atg/collections/filter/droplet/StartDateDropLet">
  <dsp: param name="collection" beanvalue="Articles" />
  <dsp: param name="collectionIdentifierKey"
    value="featuredArticles${date.dateAsDate}" />

  <dsp: oparam name="output">
    Featured Articles:
    <p><dsp: droplet name="/atg/dynamo/droplet/ForEach">
      <dsp: param name="array" param="filteredCollection" />

      <dsp: oparam name="output">
        <dsp: valueof param="param" />
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>

  <dsp: oparam name="empty">
    There are no articles today
  </dsp: oparam>
</dsp: droplet>
```

---

## Compare

Displays one of a set of possible outputs, depending on the relative value of its two input parameters.

<b>Class Name</b>	atg.droplet.Compare
<b>Component</b>	/atg/dynamo/droplet/Compare



## Required Input Parameters

### *obj1 and obj2*

The objects to compare.

## Open Parameters

### *greaterthan*

The value to render if obj 1 is greater than obj 2.

### *lessthan*

The value to render if obj 1 is less than obj 2.

### *equal*

The value to render if obj 1 is equal to obj 2.

### *noncomparable*

The value to render if obj 1 and obj 2 cannot be compared.

### *default*

The value to render if no open parameter corresponds to the comparison result. For example, obj 1 is greater than obj 2 and there is no greaterthan open parameter; or the obj 1 and obj 2 cannot be compared and there is no noncomparable parameter.

## Usage Notes

Compare takes two objects as input parameters and conditionally renders one of its open parameters, based on the relative values of the input parameters. For all non-number properties, the comparison is accomplished by casting the two objects to `java.lang.Comparable` and calling `Comparable.compareTo()`. A comparison must involve properties that share the same data type, unless those properties are instances of `java.lang.Number`.

## Example

The following example uses Compare to compare the value of the `securityStatus` property of the user's profile to the value of the `securityStatusLogin` property of the `PropertyManager` component, to determine whether the user has logged in.

---

```
<dsp: droplet name="Compare">
  <dsp: param name="obj1" bean="Profile.securityStatus"/>
  <dsp: param name="obj2" bean="PropertyManager.securityStatusLogin"/>
  <dsp: oparam name="lessthan">
    <!-- user has not logged in, so display the login form --%>
```



```
<dsp:include page="login_form.jsp"></dsp:include>
</dsp:oparam>
<dsp:oparam name="default">
  <%-- user has logged in, so proceed to the protected content --%>
  <dsp:include page="protected_content.jsp"></dsp:include>
</dsp:oparam>
</dsp:droplet>
```

---

## ComponentExists

Tests whether a Nucleus path refers to a non-null object.

<b>Class Name</b>	atg.droplet.ComponentExists
<b>Component</b>	/atg/dynamo/droplet/ComponentExists

### Required Input Parameters

#### *path*

The path of the Nucleus component to test.

### Open Parameters

#### *true*

Rendered if the component exists.

#### *false*

Rendered if the component does not exist

### Usage Notes

ComponentExists renders the output parameter *true* or *false*, depending on whether the path-specified Nucleus component is instantiated.

### Example

---

```
<droplet bean="/atg/dynamo/droplet/ComponentExists">
  <param name="path" value="/atg/modules/MyModule">
    <oparam name="true">MyModule has been defined</oparam>
```



```
<oparam name="false">MyModule has not been defined</oparam>
</droplet>
```

---

## ContentDroplet

Outputs the data of an SQL content repository item.

<b>Class Name</b>	atg.droplet.ContentDroplet
<b>Component</b>	Not provided

### Required Input Parameters

#### *repository*

The target repository.

#### *item*

The target content repository item.

### Optional Input Parameters

#### *bytes*

Sets a limit to the amount of content that is output, used to ensure that a large file does not create a runaway request. Byte limit is not exact, so output is liable to be unreadable by the browser.

### Output Parameters

#### *contentType*

Set to the file's MIME type.

### Open Parameters

#### *output*

Executes before the item content is rendered.





## Usage Notes

ContentDroplet can be used together with ContentFolderView to obtain the contents of all items in one or more content repository folders.

**Note:** You cannot use this servlet bean to render dynamic content. For example, if a content repository item contains JSP or HTML, only the source can be rendered.

You must create the properties file for ContentDroplet in  
<ATG10dir>/home/local/config/atg/dynamo/droplet and set it as follows:

---

```
$class=atg.droplet.ContentDroplet
mimeTypes=/atg/dynamo/servlet/pipeline/MimeType
```

---

## Example

See [ContentFolderView](#)

# ContentFolderView

Provides access to the contents of a folder in an SQL content repository:

<b>Class Name</b>	atg.droplet.ContentFolderView
<b>Component</b>	Not provided

## Required Input Parameters

### *repository*

The repository whose contents you wish to retrieve.

## Optional Input Parameters

### *folder*

Sets the folder to view within the specified repository. If this parameter is omitted, the servlet bean starts execution at the repository's root folder.

### *getContent*

If set to true, sets the contentItems parameter to an array of content items within the current folder. The default setting is false.



## Output Parameters

### *contentItems*

An array of content items in the current repository folder; set only if the input parameter `getContent` is set to `true`.

### *subFolders*

An array of the subfolder paths beneath the current folder. Paths start from the folder specified by the input parameter `folder`; otherwise, paths start from the repository's root folder.

## Open Parameters

### *output*

Renders output for the current repository folder.

## Usage Notes

`ContentFolderView` provides access to the items and subfolders of an SQL content repository. You can use this servlet bean to iterate over a hierarchy of repository folders, starting from the folder specified by the input parameter `folder`. If this parameter is omitted, execution begins at the repository's root folder.

You can also use this servlet bean with [ContentDroplet](#) to obtain the data of repository content items.

You must create the properties file for `ContentFolderView` in

`<ATG10dir>/home/localconfig/atg/dynamo/droplet` and set it as follows:

---

```
$class=atg.droplet.ContentFolderView
```

---

## Example

The following JSP fragment can be included in a JSP that supplies page parameters that are used to set the `ContentFolderView` input parameters `repository` and `folder`.

---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp"%>
<% /* page takes two parameters, repositoryName and startFolder
    If startFolder is null, the servlet bean starts execution at the
    repository's root folder */ %>

<dsp:droplet name="/atg/dynamo/droplet/ContentFolderView">
  <dsp:param name="repository" param="repositoryName"/>
  <dsp:param name="folder" param="startFolder"/>
  <dsp:param name="getContent" value="true"/>
  <dsp:oparam name="output">
    <dsp:droplet name="/atg/dynamo/droplet/ForEach">
```

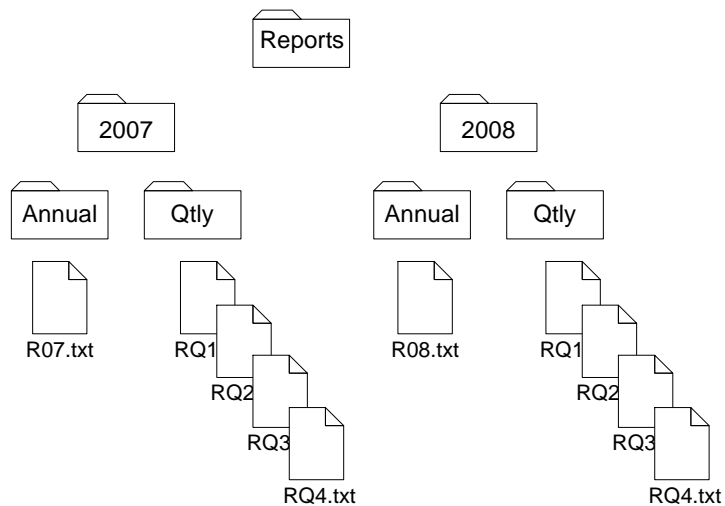


```

<dsp: param name="array" param="contentItems"/>
<dsp: oparam name="output">
  <dsp: valueof param="element.path"/><br>
  <dsp: droplet name="/atg/dynamo/droplet/ContentDroplet">
    <dsp: param name="repository" param="repositoryName"/>
    <dsp: param name="item" param="element"/>
  </dsp: droplet><br>
</dsp: oparam>
</dsp: droplet>
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param name="array" param="subFolders"/>
  <dsp: oparam name="output">
    <dsp: valueof param="element.path"/><br>
    <dsp: include page="folder.jsp">
      <dsp: param name="testFolder" param="element"/>
    </dsp: include>
  </dsp: oparam>
</dsp: droplet>
</dsp: oparam>
</dsp: droplet>

```

Given the following content repository folder hierarchy:



Execution of the JSP extract yields the following output:

```

/Reports
/Reports/2007
/Reports/2007/Annual
/Reports/2007/Annual/R07.txt
##text from R07.txt

```



```
/Reports/2007/Qtly  
/Reports/2007/Qtly/RQ1.txt  
  ##text from RQ1.txt  
/Reports/2007/Qtly/RQ2.txt  
  ##text from RQ2.txt  
/Reports/2007/Qtly/RQ3.txt  
  ##text from RQ3.txt  
/Reports/2007/Qtly/RQ4.txt  
  ##text from RQ4.txt  
/Reports/2008  
/Reports/2008/Annual  
...
```

---

## CurrencyConversionFormatter

Displays a numeric value as a currency amount, and converts a value from one currency to another, formatting it based on the locale.

<b>Class Name</b>	atg.droplet.CurrencyConversionFormatter
<b>Component</b>	/atg/dynamo/droplet/CurrencyConversionFormatter

### Required Input Parameters

#### ***currency***

The number to convert and format.

#### ***targetLocale***

The currency's conversion locale. This conversion uses exchangeRates property in the CurrencyConversionTagConverter and the locale.

### Optional Input Parameters

#### ***locale***

The currency's source locale. For example, if sourceLocale is en\_US then currency is expressed in US dollars. This value can be either a java.util.Locale object or a String that represents a locale. In order to use euro currency, the variant of the locale must be EURO. If this parameter is not provided, the default locale is used.

***euroSymbol***

Used if the target locale is using euro currency. If provided, this parameter is used as the euro symbol. This is useful because the commonly-used character set ISO Latin-1 (ISO 8859-1) does not include the euro character.

**Output Parameters*****formattedCurrency***

The formatted currency.

**Open Parameters*****output***

The oparam that is always rendered.

**Usage Notes**

CurrencyConversionFormatter can be used to convert and format a numeric amount. The amount can be converted from one currency to another.

**Example**

```
<dsp: droplet name="/atg/dynamo/droplet/CurrencyConversionFormatter">
  <dsp: param param="price.listPrice" name="currency"/>
  <dsp: param value="en_US" name="locale"/>
  <dsp: param value="de_DE_EURO" name="targetLocale"/>
  <dsp: param value="€" name="euroSymbol"/>
  <dsp: oparam name="output">
    <dsp: valueof valueishtml="true" param="formattedCurrency">no
    price</dsp: valueof>
  </dsp: oparam>
</dsp: droplet>
```

## CurrencyFormatter

Displays a numeric value as a currency amount, formatting it based on the locale

<b>Class Name</b>	atg.droplet.CurrencyFormatter
<b>Component</b>	/atg/dynamo/droplet/CurrencyFormatter



## Required Input Parameters

### *currency*

The numeric value to format. This value can be expressed as either a `java.lang.Number` or a `String`.

## Optional Input Parameters

### *locale*

The locale that determines the format of the currency amount. This value can be either a `java.util.Locale` object or a `String` that represents a locale (such as `en_US`). If omitted, the ATG platform uses the default locale supplied in the request object.

### *euroSymbol*

Lets you format the currency amount with a euro symbol even if your encoding does not support it; the character set ISO 8859-1 (Latin-1), for example, does not include the euro character. For the `value` attribute, specify the HTML code for the ASCII euro symbol as shown here:

```
<dsp: param name="euroSymbol " value="&euro; "/>
```

You must use the `valueIsHtml` attribute with a `<dsp: valueOf>` tag in the `formattedCurrency` parameter so the ATG platform displays the HTML euro symbol. Without using this attribute, the ATG platform displays the value literally rather than process it as HTML code.

```
<dsp: oparam name="output">
  <dsp: valueOf param="formattedCurrency" valueIsHtml="true">no
  price</dsp: valueOf>
</dsp: oparam>
```

## Output Parameters

### *formattedCurrency*

The formatted currency value.

## Open Parameters

### *output*

The output to render.

## Usage Notes

`CurrencyFormatter` takes a numeric value and displays that value as a currency amount, formatting it based on the locale. The formatting includes the currency symbol, the placement of the symbol, and the delimiters used. For example, if the value passed in is 20000, and the locale is `en_US`, the servlet bean formats the value as \$20,000.00; if the locale is `de_DE`, the servlet bean formats the value as 20.000,00 DM.



## Example

In this example, `CurrencyFormatter` renders the value 1059 in the appropriate format. The specified value is saved to the `priceToDisplay` parameter, which is processed through `CurrencyFormatter`. `CurrencyFormatter` applies the locale provided by `OriginatingRequest` component to the `priceToDisplay` value and saves it to the `formattedCurrency` parameter, which is displayed in the JSP. If the locale in `OriginatingRequest` is set to `en_US`, the rendered value is \$1059.00.

---

```
<dsp:setvalue param="priceToDisplay" value="1059"/>
<dsp:droplet name="/atg/dynamo/droplet/CurrencyFormatter">
  <dsp:param name="currency" param="priceToDisplay"/>
  <dsp:param name="locale" bean="/OriginatingRequest.requestLocale.locale"/>
  <dsp:oparam name="output">
    <p>Current price: <dsp:valueof param="formattedCurrency"/>
  </dsp:oparam>
</dsp:droplet>
```

---

The second example shows how to use the `euroSymbol` parameter with the `valueishtml` attribute of the `<dsp:valueof>` tag to display an amount with the euro symbol. This code displays the following value:

3,99 €

---

```
<dsp:droplet name="/atg/dynamo/droplet/CurrencyFormatter">
  <dsp:param name="currency" value="3.99"/>
  <dsp:param name="locale" value="de_DE_EURO"/>
  <dsp:param name="euroSymbol" value="&euro;"/>
  <dsp:oparam name="output">
    <dsp:valueof param="formattedCurrency" valueishtml="true">no
    price</dsp:valueof>
  </dsp:oparam>
</dsp:droplet>
```

---

## EndTransactionDroplet

Commits or rolls back the current transaction.

<b>Class Name</b>	atg.dtm.EndTransactionDroplet
<b>Component</b>	/atg/dynamo/transaction/droplet/EndTransaction



## Required Input Parameters

### *op*

Specifies how to end the transaction with the setting `commit` or `rollback`.

## Open Parameters

### *errorOutput*

Rendered if an error occurs during the commit or rollback. Within the open parameter, the error is communicated through the following parameters:

- `errorMessage`: a text message indicating the error
- `errorStackTrace`: the full stack trace of the error

### *successOutput*

Rendered if the commit or rollback operation completes successfully.

## Usage Notes

`EndTransactionDroplet` can be used in a page to manually commit or roll back the current transaction. Any errors that occur during the commit or rollback process are reported to an open parameter, thereby allowing the application to report transaction failures to the user. Any errors that occur are also logged.

# ErrorMessageForEach

Displays error messages that occur during a form submission.

<b>Class Name</b>	<code>atg.droplet.ErrorMessageForEach</code>
<b>Component</b>	<code>/atg/dynamo/droplet/ErrorMessageForEach</code> <code>/atg/userprofiling/ProfileErrorMessageForEach</code> <code>/atg/demo/QuincyFunds/FormHandlers/RepositoryErrorMessageForEach</code>

## Required Input Parameters

None





## Optional Input Parameters

### ***exceptions***

Holds a vector of `DropLetExceptions`. If omitted, `ErrorMessageForEach` uses the list of all exceptions that occurred in this request.

### ***messageTable***

Holds a text string that is a mapping of exceptions to error messages. If omitted, `ErrorMessageForEach` determines the message to use as described previously.

### ***propertyNameTable***

Holds a text string that is a mapping of property names to text strings. If omitted, `ErrorMessageForEach` determines the text string to use as described previously.

## Output Parameters

### ***message***

An error message that maps to the exception's `errorCode` property. This parameter is set once for each exception in the input vector.

### ***propertyName***

A text string that maps to the exception's `propertyName` property. This parameter is set once for each exception in the input vector.

## Open Parameters

### ***output***

Rendered once for each element in the vector of exceptions.

### ***outputStart***

If the vector of exceptions is not empty, rendered once before the output parameter is rendered.

### ***outputEnd***

If the vector of exceptions is not empty, rendered once after the output parameter is rendered for each exception.

### ***empty***

Rendered if the vector of exceptions contains no elements.

## Usage Notes

`ErrorMessageForEach` takes a vector of form exceptions and, for each exception, displays an appropriate error message. The exceptions are of class `atg.droplet.DropletException`, or a subclass of this class.

The vector of exceptions can be passed automatically by the request that produces the exceptions, or specified explicitly using the exception's input parameter. For example, any form handler of class `atg.droplet.GenericFormHandler` (or a subclass of this class) has a `formExceptions` property, which is a vector of the exceptions that occurred during form processing. To pass these exceptions to `ErrorMessageForEach`, you can use a tag like this:

```
<dsp: param name="exceptions" bean="MyFormHandler.formExceptions" />
```

For each exception, `ErrorMessageForEach` displays the message associated with that exception's `errorCode` property. For example, if the value of an exception's `errorCode` property is `invalidPassword`, `ErrorMessageForEach` displays the message associated with `invalidPassword`. The messages are stored in tables of key/value pairs, where the keys contain error codes and the corresponding values hold user-friendly messages. For example, the entry for `invalidPassword` might look like this:

```
invalidPassword=Your password is incorrect.
```

Messages can be specified in several ways. `ErrorMessageForEach` searches for a key that matches `errorCode` in the following areas, listed in order of precedence:

1. The servlet bean's `messageTableInputParameters`
2. The servlet bean's `messageTable` property
3. The resource bundle specified by servlet bean's `resourceName` property

If `ErrorMessageForEach` fails to find the key, it displays the message produced by the exception itself. The vector of exceptions is preserved across redirects, so you can redirect the user to a separate page that displays the errors.

If you specify messages using the `messageTable` input parameter or the `messageTable` property, use a comma to separate entries. In order to embed a comma in the message itself, enter two consecutive commas. In the following example, two messages are passed in using the `messageTable` input parameter:

```
<dsp: param name="messageTable"
  value="invalidPassword=Your password is incorrect. , invalidLogin=Your
  user name is incorrect. For assistance, , click Help." />
```

By default, the `messageTable` property is null, so the messages are taken from the resource specified by the `resourceName` property. The default resource bundle is `atg.droplet.ErrorMessageResources`, and the messages are stored in the `ErrorMessageResources.properties` file. This file contains messages for error codes commonly produced by the ATG platform.

Some exceptions passed to the servlet bean can be of class `DropletFormException`, which is a subclass of the `DropletException` class. In addition to an `errorCode` property, a `DropletFormException` on also has a `propertyName` property that stores the name of the property in the form handler that triggered the



exception. This `propertyName` value can be mapped to a text string, which is passed to the error message through a parameter. For example, if the user leaves a required text field empty, the error message might be:

```
missingRequiredValue=Must supply a value for the param: propertyName field.
```

`ErrorMessageForEach` replaces `param: propertyName` with the text associated with the exception's `propertyName` property. For example, if the value of the `propertyName` property is `login`, `ErrorMessageForEach` displays the text associated with `login`. As with the error messages themselves, these texts are stored in tables of key/value pairs, with the keys being the exceptions' `propertyName` properties and the values being the texts that are mapped to those keys. For example, a table of these texts might look like this:

```
login=user name,
password=PIN code,
address=home address
```

`ErrorMessageForEach` searches for a key that matches the `propertyName` in the following areas, listed in order of precedence:

1. The servlet bean's `propertyNameTable` Input Parameters
2. The servlet bean's `propertyNameTable` property
3. The resource bundle specified by servlet bean's `resourceName` property

If `ErrorMessageForEach` fails to find the key, it displays the value of `propertyName` itself. By default, the `propertyNameTable` property is null, and no mappings are defined in the `ErrorMessageResources.properties` file. Unless you define the mappings yourself, the `propertyName` value is displayed.

You can use any of the mechanisms described above to extend or modify the messages that `ErrorMessageForEach` displays. For localization purposes, you can define your own resource bundles, and use these resource bundles to specify the messages and `propertyName` parameter values. You can create resource bundles in multiple languages, and have `ErrorMessageForEach` display different messages depending on the user's locale. For more information about resource bundles, see the *Internationalizing an ATG Web Site* chapter of the [ATG Programming Guide](#).

While you can use resource bundles to define the message text and the `propertyName` values, you cannot use them to supply other arguments that are replaced by actual values because `ErrorMessageForEach` cannot retrieve these arguments from the `DropletFormException`. For example, you might want to display the following error message when a new user enters a login name that is already in use:

```
Login name {MGarcia} already in use; please choose another login name
```

In this situation, you can use the resource bundle to supply the static message text, but `ErrorMessageForEach` cannot replace the login name argument with an actual value. One alternative is to use `ForEach` rather than `ErrorMessageForEach` and display the default message from the `DropletException`. In an internationalized Web site, the form handler defines localized versions of all messages and displays the appropriate version according to the locale of the current request.



## Example

The following example displays a list of exceptions that can occur in a form submission. The Switch servlet bean at the start determines whether there were any form errors. The exceptions parameter of the ErrorMessageForEach points to the formExceptions property of the Survey component. The value of the messageTable parameter provides two error messages that can be displayed for different errors.

```
<dsp: droplet name="/atg/dynamo/droplet/Switch">
  <dsp: param bean="Test.formError" name="value"/>
  <dsp: oparam name="true">
    <h2>The following Errors were found in your survey</h2>
    <dsp: droplet name="/atg/dynamo/droplet/ErrorMessageForEach">
      <dsp: param name="exceptions" bean="Survey.formExceptions"/>
      <dsp: param name="messageTable" value="missingRequiredField=You did
not answer all the survey questions,
oneWordCheckFailed=You did not provide a single word to describe
your interests"/>
      <dsp: oparam name="output">
        <b>
          <dsp: valueof param="message"/>
        </b>
        <p>
        </dsp: oparam>
      </dsp: droplet>
    <P>
    <P>
  </dsp: oparam>
</dsp: droplet>
```

## For

Displays a single output the number of times specified.

<b>Class Name</b>	atg.droplet.For
<b>Component</b>	/atg/dynamo/droplet/For

## Required Input Parameters

### *howMany*

The number of times to render the output parameter.



## Output Parameters

### *index*

The zero-based index of the current array element, incremented each time the output parameter is rendered.

### *count*

The one-based index of the current array element, incremented each time the output parameter is rendered.

## Open Parameters

### *output*

Rendered the number of times specified by the howMany parameter.

## Usage Notes

The servlet bean For loops the number of times specified by the input parameter. Given an integer parameter howMany, For renders its output parameter that many times. The howMany parameter must be a String representation of an int. Each iteration sets the i ndex parameter to the current loop count, starting at 0, and sets the count parameter to index plus one, starting at 1.

## Example

In the following example, the user is presented with textboxes, the number of which is user-determined. On the first page (code not shown) the user specifies how the number of text boxes to be shown. This value is stored in the form's numChoi ces property. On the second page (code excerpt included below), the For servlet bean displays the user-specified number of text boxes by setting its howMany input parameter from the numChoi ces property. The servlet bean uses the count parameter to supply a consecutive number to each text box. When a user enters values into the text boxes, those values are saved to the form's choi ce property.

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/For">
  <dsp: param name="howMany" bean="SomeForm. numchoi ces"/>
  <dsp: oparam name="output">
    Text box # <dsp: val ueof param="count"/>:
    <dsp: i nput bean="SomeForm. choi ce[param: i ndex]" type="text"/>
    <br/>
  </dsp: oparam>
</dsp: dropl et>
```

---



## ForEach

Displays each element of an array.

<b>Class Name</b>	atg.droplet.ForEach
<b>Component</b>	/atg/dynamo/droplet/ForEach

### Required Input Parameters

None

### Optional Input Parameters

#### *array*

The list of items to output: a Collection (Vector, List, or Set), Enumeration, Iterator, Map, Dictionary, or array. If omit this parameter or supply an empty parameter, the open parameter empty is rendered.

#### *sortProperties*

Holds a string that specifies the order in which array items are rendered. The syntax of this parameter depends on the array item type: JavaBean, Dynamic Bean, Date, Number, or String.

To sort on the properties of a JavaBean, specify the value of `sortProperties` as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. To specify ascending sort order, prepend the property name with a plus + sign; to specify descending order, prepend with a minus - sign. The following example sorts a JavaBean array alphabetically by title, then in descending order of size, use this input parameter:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

If an array consists of Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus - sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

In order to sort Map elements by key, set the value as follows:

```
{+|-}_key
```

For example:

```
<dsp: param name="sortProperties" value="-_key" />
```



A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### *index*

The zero-based index of the current array element, incremented each time the output parameter is rendered.

### *count*

The one-based index of the current array element, incremented each time the output parameter is rendered.

### *key*

If the array parameter is a Map or Dictionary, set to the value of the Map or Dictionary key.

### *element*

Set to the current array element each time the index increments and the output parameter is rendered.

### *size*

Set to the size of the array, if applicable. If the array is an Enumeration or Iterator, size is set to -1.

## Open Parameters

### *output*

Rendered once for each array element.

### *outputStart*

If the array contains elements, rendered before any elements are output. For example, this parameter can be used to render the table heading.

### *outputEnd*

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render text following a table.

### *empty*

Rendered if the array is empty.



## Usage Notes

ForEach renders a listing of elements specified by the array parameter in the order you specify. The array parameter can be a Collection (Vector, List, or Set), Enumeration, Iterator, Map, Dictionary, or array.

## Example

The following example uses ForEach to present a list of people. The servlet bean renders the output parameter once for each entry in the people array. It defines a parameter called element (representing a person) on each iteration. Then, the servlet bean passes the element parameter to a page, displayPerson.jsp, which produces the HTML formatting. The displayPerson.jsp page itself uses the Switch servlet bean to render different output for each person on the list.

The following example uses ForEach to display a list of the students enrolled in a class. The servlet bean renders the output parameter once for each student in the enrolledStudents array. As ForEach loops through the array of enrolledStudents, each student is bound to the parameter named currentStudent (which is set to the element parameter). The remaining code shows how you can pass the currentStudent parameter to another JSP and how you can display the property values of currentStudent on the page.

---

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param name="array" bean="ClassroomService.enrolledStudents" />

  <!-- As the ForEach loops thru the array of students, each of the
        elements is bound to a parameter named "CurrentStudent": --%>
  <dsp: setvalue param="CurrentStudent" paramvalue="element"/>

  <dsp: oparam name="empty">
    There are no students in this class. <br/>
  </dsp: oparam>

  <!-- Display a header before looping thru students in the array: --%>
  <dsp: oparam name="outputStart">
    Here is the list of students in this class: <br/>
  </dsp: oparam>

  <!-- display this output for each of the student array elements: --%>
  <dsp: oparam name="output">

    Student # <dsp: valueof param="count"/> :
      <dsp: valueof param="CurrentStudent.fullName"/> at address:

    <!-- Display student's address information – which is fetched from
          the student's "address" object property: --%>
    <dsp: valueof param="CurrentStudent.address.city"/>
    <dsp: valueof param="CurrentStudent.address.State"/><br/>

    <!-- This is how you pass the "CurrentStudent" to another page
          fragment which displays more student information: --%>
```





```

has the following grades:
<dsp:include page="displayStudentGrades.jsp">
  <dsp:param name="studentInputParam" param="CurrentStudent"/>
</dsp:include>

</dsp:oparam> <%-- output param --%>

<%-- Display this after looping thru all students in the array: --%>
<dsp:oparam name="outputEnd">
  Total number of students: <dsp:valueof param="size"/> <br/>
  End of Student list.
</dsp:oparam>

</dsp:droplet>

```

## Format

Displays one or more text values, formatting them based on locale.

<b>Class Name</b>	atg.droplet.Format
<b>Component</b>	/atg/dynamo/droplet/Format

### Required Input Parameters

The Format servlet bean typically requires the following input parameters:

- `format` specifies the text to display.
- One input parameter for each format expression specified in the `format` value, where the input parameter name maps to the expression's first argument.

The ATG platform uses the `format` parameter to construct the `java.text.MessageFormat` object. The value of this parameter is a string that can embed one or more format expressions that use the following syntax:

```
{arg[, format-type, [format-type]...]}
```

where *arg* is a variable that can be formatted, and *format-type* optionally specifies any format accepted by the `java.text.MessageFormat` class.

For example:



---

```
<dsp: param name="format"
  value="At {when, time} on {when, date}, there was {what} on planet
{where, number, integer}." />
```

---

If *arg* is a variable—in the previous example, when, what, and where—the servlet bean requires an input parameter of the same name. Given the previous format parameter example, the servlet bean requires three input parameters:

---

```
<dsp: param name="when" bean="MyComponent.date" />
<dsp: param name="what" bean="MyComponent.what" />
<dsp: param name="where" bean="MyComponent.where" />
```

---

## Optional Input Parameters

### *locale*

A dynamic page parameter or static text. The value must correspond to a supported locale format defined in `java.util.Locale`. If omitted, the ATG platform uses the default locale of the request object.

## Output Parameters

### *message*

Renders the servlet bean output, or resultant string that is calculated by processing the supplied input parameters.

## Open Parameters

### *output*

Uses a `<dsp: valueof>` tag to display the final `MessageFormat` element with the desired formatting, which is represented by the message page parameter. For example:

---

```
<dsp: oparam name="output">
  <dsp: valueof param="message" />
</dsp: oparam>
```

---

## Usage Notes

Format makes the functionality of the `java.text.MessageFormat` class available to JSPs. With it, you can:

- Embed the value of one or more page parameters in a string.
- Format a numeric value as a date, time, currency, or percent format.



This servlet bean lets you format parameters—typically strings—into any format accepted by the `java.text.MessageFormat` class.

## Example

In the following example, the `Format` servlet bean concatenates two strings. This example assumes the existence of a `MyComponent` component with properties `firstName` and `lastName`.

```
<dsp: droplet name="/atg/dynamo/droplet/Format">
  <dsp: param name="format" value="Name: {last}, {first}"/>
  <dsp: param bean="/MyComponent.firstName" name="first"/>
  <dsp: param bean="/MyComponent.lastName" name="last"/>
  <dsp: oparam name="output">
    Name: <dsp: valueof param="message" />
  </dsp: oparam>
</dsp: droplet>
```

This code creates two page parameters `first` and `last`, and sets the values of each to the corresponding `MyComponent` property. `Format` constructs an object and sets each argument to a value in an array, inserting the actual value for each page parameter. `Format` passes the object equipped with the array to the `MessageFormat` object for formatting. Because this example does not require formatting, the `MessageFormat` object compiles the final message and saves it to a page parameter called `message`. The final message might appear as follows:

Name: Dover, Anita

## GetDirectoryPrincipal

Returns the `DirectoryPrincipal` for a specified type and ID.

<b>Class Name</b>	<code>atg.userprofiling.GetDirectoryPrincipal</code>
<b>Component</b>	<code>/atg/dynamo/droplet/GetDirectoryPrincipal</code>

## Required Input Parameters

### *type*

Type of principal, set to one of the following:

- `user`
- `organization`



- role
- relative-role
- profile-group

***id***

The identity to look up. The value of this parameter depends on the value specified for the type parameter:

- user: the login name of the user
- organization: the name of the organization
- role: the path name of the role
- relative-role: the organization name and functional role name separated by a backslash (/) Example: Accounting/VicePresident.

**Optional Input Parameters*****directory***

The UserDirectory to use when looking up identities. If omitted, the default UserDirectory specified by the userDirectory property is used.

**Output Parameters*****principal***

Set to the DirectoryPrincipal object that corresponds to the given type and id.

***persona***

If the userDirectoryUserAuthority property is set, set to the Persona corresponding to the DirectoryPrincipal that the principal parameter is set to.

**Open Parameters*****output***

Rendered if the DirectoryPrincipal is found.

***empty***

Rendered if the DirectoryPrincipal with the given type and id cannot be found.

**Usage Notes**

GetDirectoryPrincipal displays the atg.userdirectory.DirectoryPrincipal that corresponds to the specified input type and id parameters. GetDirectoryPrincipal can also display the atg.security.Persona corresponding to the directory principal it finds.



## Example

The following example finds the DirectoryPrincipal for the Approver role:

```
<dsp: droplet name="/atg/dynamo/droplet/GetDirectoryPrincipal">
  <dsp: param name="type" value="role"/>
  <dsp: param name="id" value="Approver"/>
  <dsp: oparam name="output">
    Found directory principal <dsp: valueof param="principal.name"/>.
  </dsp: oparam>
</dsp: droplet>
```

## GetSiteDroplet

Gets the site associated with a site ID.

<b>Class Name</b>	atg.droplet.multisite.GetSiteDroplet
<b>Component</b>	/atg/dynamo/droplet/multisite/GetSiteDroplet

## Input Parameters

### *siteId*

The ID of the site to get.

## Output Parameters

### *output*

Rendered if the site is found.

### *empty*

Rendered if the site is not found or siteId is null.

### *error*

Rendered if an error occurs.

### *errorMessage*

If an error occurs, set with the error message, if any.

**site**

Set to the site object of the returned site.

**Usage Notes**

Given a valid site ID, GetSiteDroplet returns a site object—an implementation of interface `atg.multiplesite.Site`—which encapsulates a site configuration. The output parameter enables access to that site's properties.

You can obtain the current site and its configuration through the Nucleus component `/atg/multiplesite/Site`. For example, you can obtain the current site's ID as follows:

```
<dsp:tomap bean="/atg/multiplesite/Site.id" var="siteID"/>
```

**Example**

The following JSP code obtains the configuration of the site `mySite`. The output parameter `site` provides access to all site configuration properties. In this example, the code obtains the site's `closingDate` property and compares it to the current date to determine whether the site is active:

---

```
<%-- Get the current site configuration and look at its closingDate property --%>
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
<%@ page import="java.util.Date;"%>
<dsp:page>

<dsp:droplet name="/atg/dynamo/droplet/multiplesite/GetSiteDroplet">
  <dsp:param name="siteID" value="mySite"/>
  <dsp:oparam name="output">
    <dsp:getvalueof var="closeDate" param="site.closingDate"
      vartype="java.util.Date">
      <c:choose>
        <c:when test="${System.currentTimeMillis() < closeDate.getTimeInMillis()}">
          Site is still active
        </c:when>
        <c:otherwise> Site is no longer active </c:otherwise>
      </c:choose>
    </dsp:oparam>
  </dsp:droplet>
</dsp:page>
```

---

## HasBusinessProcessStage

Tests whether a business object has reached a specified business process stage.



<b>Class Name</b>	atg.markers.bp.droplet.HasBusinessProcessStage
<b>Component</b>	/atg/markers/bp/droplet/HasBusinessProcessStageDroplet

## Required Input Parameters

### *businessProcessStage*

The stage within the business process to check. Rather than specify markers for a particular stage, you can check for markers from a given process by setting this parameter to the ANY\_VALUE property as follows:

```
<dsp: param name="businessProcessStage"
  bean="HasBusinessProcessStageDroplet.ANY_VALUE" />
```

## Optional Input Parameters

### *businessProcessName*

The name of the business process; defaults to the servlet bean's defaultBusinessProcessName property. Setting the defaultBusinessProcessName property lets you create instances of the servlet bean that are specific to a single business process.

## Output Parameters

### *errorMsg*

The failure error message.

## Open Parameters

### *true*

Rendered if the specified stage reached marker is found.

### *false*

Rendered if the specified stage reached marker is not found.

### *error*

Rendered on error.

## Usage Notes

HasBusinessProcessStage checks whether the item has reached a stage in a business process.



## Example

```
<dsp: droplet name="HasBusinessProcessStage">
  <dsp: param name="businessProcessName" value="ShoppingProcess"/>
  <dsp: param name="businessProcessStage" value="ShippingPriceViewed"/>
  <dsp: oparam name="true">
    ...
  </dsp: oparam>
  <dsp: oparam name="false">
    ...
  </dsp: oparam>
</dsp: droplet>
```

## HasEffectivePrincipal

Fires a content event if a user has the appropriate Principal (identity).

<b>Class Name</b>	atg.userprofiling.HasEffectivePrincipal
<b>Component</b>	/atg/dynamo/droplet/HasEffectivePrincipal

## Required Input Parameters

### *type*

Set to one of the following four values:

- user
- organization
- role
- relative-role
- profile-group

## Optional Input Parameters

### *id*

The identity to look up. The value you specify depends on the value of the type parameter:

- user: the login name of the user—for example, MGarcia
- organization: the path and organization name. For example:





```
AnyCorps
/AnyCorps
/AnyCorps/someOrganization
```

- **role**: the path and a global role name. For example:

```
admin
/admin
/admin/networkAdmin
```

- **relative-role**: the organization name and functional role name separated by a backslash (/). For example:

```
Accounting/VicePresident
```

You can find a path for an organization or a global role by looking in the ACC People and Organizations task area. For example, you can find all global roles in the People and Organizations task area, under the Roles option. By expanding the Global Roles tree option, you can navigate to each role by traversing its path.

Notice that `id` values can have a leading slash, but one is not required.

## Open Parameters

### ***output***

Rendered if the identity check is successful.

### ***default***

Rendered if the identity check fails.

### ***unknown***

Rendered if the user has no known identity.

## Usage Notes

`HasEffectivePrincipal` checks whether a user has a specified identity and renders content based on the result of its query. For example, you can check if the user has a specific principal (identity), if the user is a member of a specified organization, or if the user has a specified role, and you can set this servlet bean to render that information.

## Example

`HasEffectivePrincipal` determines whether the current user is assigned the role of Administrator.

```
<dsp: droplet name="/atg/dynamo/droplet/HasEffectivePrincipal">
  <dsp: param name="type" value="role"/>
  <dsp: param name="id" value="Administrator"/>
  <dsp: oparam name="output"> You are an administrator<p> </dsp: oparam>
```



```
<dsp:oparam name="default"> You are not an administrator<p> </dsp:oparam>
<dsp:oparam name="unknown"> You are not logged in<p> </dsp:oparam>
</dsp:droplet>
```

---

## HasFunction

Tests whether a given user has the specified role.

<b>Class Name</b>	atg.userdirectory.droplet.HasFunction
<b>Component</b>	/atg/userdirectory/droplet/HasFunction

### Required Input Parameters

#### *function*

The function property of the relative role held by the user.

#### *userId*

The ID of the user whose functional role to check. For example:

---

```
<dsp:param name="userId" beanvalue="/atg/userprofiling/Profile.id"/>
```

---

### Optional Input Parameters

#### *organizationId*

An organization name. If set, the servlet bean returns true only if the user is assigned the function in this organization.

### Open Parameters

#### *true*

This parameter and the tags nested within it are rendered when an item matches the conditions defined by the input parameters.

#### *false*

This parameter and the tags nested within it are rendered when no item matches the conditions defined by the input parameters.

**error**

Rendered if there is an error when the query executes.

**Usage Notes**

HasFunction determines whether a given user has a certain functional role and renders certain content based on the result. You can limit the query further by specifying an organization, meaning that a result of true is returned only if the user has the specified role in a particular organization.

Be sure to set the userDirectory property to the relevant User Directory component.

**Example**

In this example, HasFunction determines if the active user is an admin for the org1001 organization, and when he/she is, displays a link to a page for creating user accounts. Otherwise, a link to the profile page displays.

---

```
<dsp: droplet name="/atg/userdirectory/droplet/HasFunction">
  <dsp: param name="function" value="admin"/>
  <dsp: param name="userId" beanvalue="/atg/userprofiling/Profile.id"/>
  <dsp: param name="organization" value="org1001"/>

  <dsp: oparam name="true">
    <dsp: a href="UserAccountForms.jsp">Set up new user accounts</dsp: a>
  </dsp: oparam>

  <dsp: oparam name="false">
    <dsp: a href="UserAccountInfo.jsp">Edit your profile</dsp: a>
  </dsp: oparam>
</dsp: droplet>
```

---

**Invoke**

Invokes a named method on the specified Java object.

<b>Class Name</b>	atg.droplet.Invoke
<b>Component</b>	/atg/dynamo/droplet/Invoke



## Required Input Parameters

### ***object***

The object whose method is invoked.

### ***method***

The method to invoke, an instance of `java.lang.reflect.Method` or the name of a public method declared in the object-specified value as a `String`. The method `java.lang.Class.getMethod()` finds the named method.

### ***args***

An array of arguments supplied to the specified method. Omit this parameter if the method takes no arguments. If the value is of type `Object`, the method assumes it takes a single argument. If the value is an array of `Objects` the method assumes it takes these objects as arguments.

## Output Parameters

### ***return***

The object returned by the method, null if the method returns void.

### ***throwable***

An exception that is thrown when the method executes, null if no exception is thrown.

## Open Parameters

### ***output***

Rendered if no errors occur.

### ***catch***

Rendered if a checked exception occurs during method execution. The `throwable` parameter is set to the thrown exception.

### ***finally***

Rendered if an error during method execution. The `throwable` parameter is set to the exception thrown.

## IsEmpty

Displays one of two possible outputs, depending on whether its input parameter is empty.



<b>Class Name</b>	atg.droplet.IsEmpty
<b>Component</b>	/atg/dynamo/droplet/IsEmpty

## Required Input Parameters

### *value*

The value to test. The `value` parameter is tested to determine if it is empty, and the result of this test determines the name of the open parameter to render. For example, if the `value` parameter is empty, then `IsEmpty` renders the value of its `true` open parameter.

## Open Parameters

### *true*

The value to render if the `value` parameter is empty.

### *false*

The value to render if the `value` parameter is not empty.

## Usage Notes

`IsEmpty` conditionally renders one of its open parameters based on the value of its `value` input parameter. If `value` is null or if the object is empty, then the output parameter `true` is rendered. Otherwise, the output parameter `false` is rendered.

The meaning of empty depends on the data type of the value parameter:

Data type	Considered empty if...
Collection	<code>Collection.isEmpty</code> returns true
object array	<code>Object[].length == 0</code>
Map	<code>Map.isEmpty</code> returns true
Dictionary	<code>Map.isEmpty</code> returns true
String	<code>String.equals("")</code> returns true

## Example

The following example displays the a user's hobbies, using [ForEach](#) to render them. If a user identifies no hobbies, a message displays:



```
<dsp: droplet name="IsEmpty">
  <dsp: param name="value" bean="MyProfile.hobbies"/>
  <dsp: oparam name="false">
    Your hobbies are:
    <dsp: droplet name="ForEach">
      <dsp: param name="array" bean="MyProfile.hobbies"/>
      <dsp: oparam name="output">
        <dsp: valueof param="element"/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>
  <dsp: oparam name="true">
    All work and no play makes Jack a dull boy.
  </dsp: oparam>
</dsp: droplet>
```

## IsNull

Displays one of two possible outputs, depending on whether its input parameter is null.

<b>Class Name</b>	atg.droplet.IsNull
<b>Component</b>	/atg/dynamo/droplet/IsNull

### Required Input Parameters

#### *value*

The value to test. The `value` parameter is tested to determine if it is null, and the result of this test determines the name of the open parameter to render. For example, if the `value` parameter is null, then `IsNull` renders the value of its `true` open parameter.

### Open Parameters

#### *true*

The value to render if the `value` parameter is null.

#### *false*

The value to render if the `value` parameter is not null.



## Usage Notes

IsNull conditionally renders one of its open parameters based on the value of its `value` input parameter. If `value` is null, then the output parameter `true` is rendered. Otherwise, the output parameter `false` is rendered.

## Example

In this example, the ATG platform checks to see if `MyProfile.email` has a value, and if not, the ATG platform provides an input field for adding an email address and a button for saving it to the database.

```
<dsp: droplet name="IsNull">
  <dsp: param bean="MyProfile.email" name="value"/>
  <dsp: oparam name="true">
    <dsp: form action="address_book.jsp" method="POST">
      My email address:
      <dsp: input type="text" bean="MyProfileFormHandler.email"/>
      <dsp: input type="submit" bean="MyProfileFormHandler.update" value="Update"/>
    </dsp: form>
  </dsp: oparam>
</dsp: droplet>
```

## ItemLink

Takes a repository item as input and generates either a static or dynamic URL, depending on whether the page it is on is being viewed by a human visitor or a Web spider.

<b>Class Name</b>	atg.repository.seo.ItemLink
<b>Component</b>	Not provided with the ATG platform

## Required Input Parameters

Specify the repository item with one of the following input parameters:

- `item`
- `id`

### *item*

The target repository item.

***id***

The repository ID of the target repository item. If you specify this input parameter, also specify two additional parameters:

- `itemDescriptorName`
- `repository` or `repositoryName`

***itemDescriptorName***

The repository item's item descriptor.

***repository***

The repository that contains the item.

***repositoryName***

The name of the repository that contains the item.

**Output Parameters*****url***

The static or dynamic URL generated by the servlet bean.

***errorMessage***

The error message produced if a problem occurs generating the URL.

**Open Parameters*****output***

Rendered if the URL is generated successfully.

***empty***

Rendered if no template is defined for the browser type.

***error***

Rendered if a problem occurs when generating the URL.

**Usage Notes**

ItemLink takes a repository item as input and generates either a static or dynamic URL, depending on whether the page it is on is being accessed by a human visitor or a Web spider. ItemLink uses the repository item and information from the request's HTTP header to determine the type and format of the URL to render.





You can specify the repository item in one of two ways:

- Use the `item` input parameter to pass in the actual repository item. In this case, the item is typically the output of another servlet bean, such as `RepositoryLookup` or `ItemLookupDropLet`.
- Use the `id` and `itemDescriptorName` parameters, and either the `repository` or `repositoryName` parameter, to provide the information needed to uniquely identify the repository item.

**Note:** If you embed the `ItemLink` servlet bean in the `Cache` servlet bean, be sure to create separate cache keys for human visitors and Web spiders (which can be differentiated based on the User-Agent value of the request). Otherwise the page might end up containing the wrong type of URL for the visitor.

For more information about `ItemLink`, see the discussion of Search Engine Optimization in the [ATG Programming Guide](#).

## Example

The following example uses an `ItemLink` servlet bean to generate URLs on a commerce site.

---

```
<dsp: dropLet name="/atg/commerce/catalog/CategoryLookup">
  <dsp: param bean="/OriginatingRequest.requestLocale.locale"
    name="repositoryKey"/>
  <dsp: param name="id" param="id"/>
  <dsp: oparam name="output">
    <dsp: dropLet name="/atg/repository/seo/CatalogItemLink">
      <dsp: param name="item" param="element"/>
      <dsp: param name="navAction" value="jump"/>
      <dsp: param name="navCount"
        bean="/atg/commerce/catalog/CatalogNavHistory.navCount"/>
      <dsp: oparam name="output">
        <dsp: getvalueof var="finalUrl" vartype="String" param="url">
          <dsp: a href="{finalUrl}">
            <dsp: valueof param="element.displayName"/>
          </dsp: a>
        </dsp: getvalueof>
      </dsp: oparam>
      <dsp: oparam name="error">
        Failed to create URL link to category ID: <dsp: valueof param="id"/>
        <br/><dsp: valueof param="errorMessage"/>
      </dsp: oparam>
    </dsp: dropLet>
  </dsp: oparam>
</dsp: dropLet>
```

---



## ItemLookupDroplet

Looks up an item in one or more repositories based on the item's ID, and renders the item on the page.

<b>Class Name</b>	atg.repository.servlet.ItemLookupDroplet
<b>Component</b>	/atg/dynamo/droplet/ItemLookupDroplet

### Required Input Parameters

#### *id*

The ID of the item to look up.

#### *repositoryKey*

A key that maps to a secondary set of repositories.

### Optional Input Parameters

#### *repository*

The repository to search.

**Note:** Use of this parameter is not recommended. Instead, you should define different instances of the `ItemLookupDroplet` for each set of repositories.

#### *itemDescriptor*

The name of the item descriptor to use to load the item.

**Note:** Use of this parameter is not recommended. Instead, you should specify the item descriptor in the `itemDescriptor` property of the servlet bean.

### Output Parameters

#### *element*

Set to the `RepositoryItem` corresponding to the ID supplied.

#### *error*

Any exception that might occur while looking up the item.

### Open Parameters

The following list describes the output and open parameters of the `ItemLookupDroplet`.

**output**

Rendered if the item is found in the repository.

**empty**

Rendered in all other cases, such as item not found, site visitor did not specify a required parameter.

**Usage Notes**

ItemLookupDroplet displays items in a repository. If your site uses more than one repository, you can create multiple instances of this servlet bean and configure each one differently. The repository and item descriptor type can be set through properties. If the useParams property is true, then the repository and item descriptor are resolved through input parameters. You can configure this servlet bean to define a mapping from a key to an alternate set of repositories.

For example, a ItemLookupDroplet servlet bean named ProductLookup might have the following properties:

---

```

$class=atg.repository.servlet.ItemLookupDroplet

repository=/atg/commerce/catalog/ProductCatalog
itemDescriptor=product

alternateRepositories=\
    fr_FR=/atg/commerce/catalog/FrenchProductCatalog
    ja_JP=/atg/commerce/catalog/JapaneseProductCatalog
    de_DE=/atg/commerce/catalog/GermanProductCatalog

```

---

The servlet bean can then be invoked with the following parameters:

---

```

<dsp: droplet name="/atg/commerce/catalog/ProductLookup">
  <dsp: param name="id" param="productId"/>
  <dsp: param bean="/OriginalRequest.requestLocale.localeString"
name="repositoryKey"/>
  <dsp: setvalue param="product" paramvalue="element"
  <dsp: oparam name="output">
    <dsp: getvalueof var="a11" param="product.template.url"
    vartype="java.lang.String">
    <dsp: a href="{a11}">
      <dsp: param name="prod_id" param="product.repositoryId"/>
      <dsp: valueof param="product.displayName"/>
    </dsp: a>
  </dsp: getvalueof>
</dsp: oparam>
</dsp: droplet>

```

---



If the mapping for the provided key is found, that repository is used. Otherwise, the system falls back to the default repository and searches for the item.

If the item cannot be found in the repository searched (such as the French product catalog), then the servlet again falls back to the default repository and tries to find the item with the same ID. This is only useful if the items have the exact same ID in each repository.

For example, you might view the site in French and try to look at product 1234. If the ID is defined in the French product catalog, you see the French version of that product. However, if 1234 is not defined, then you see the default English version of product 1234.

This behavior can be modified with use of the `useDefaultRepository` and `getDefaultItem` properties. If `useDefaultRepository` is false and an alternate repository cannot be found, then no repository is searched and empty is used. The same is true if an alternative repository is selected, but the item cannot be found and if `getDefaultItem` is false.

## MostRecentBusinessProcessStage

Tests whether the business process stage most recently reached by a business object matches the specified stage.

<b>Class Name</b>	<code>atg.markers.bp.droplet.MostRecentBusinessProcessStage</code>
<b>Component</b>	<code>/atg/markers/bp/droplet/MostRecentBusinessProcessStageDroplet</code>

### Required Input Parameters

#### ***businessProcessStage***

The stage within the business process.

### Optional Input Parameters

#### ***businessProcessName***

The name of the business process. Defaults to the servlet bean's `defaultBusinessProcessName` property. Setting the `defaultBusinessProcessName` property lets you create instances of the servlet bean that are specific to a single business process.

### Output Parameters

#### ***errorMsg***

The error message describing a failure.

**marker**

The matching stage reached marker, if found.

**Open Parameters****true**

Rendered if the stage most recently reached matches the specified stage found.

**false**

Rendered if stage most recently reached does not match the specified stage found.

**error**

Rendered on error.

**Usage Notes**

MostRecentBusinessProcessStage checks whether the stage most recently reached matches the specified stage.

**Example**

```
<dsp: droplet name="MostRecentBusinessProcessStage">
  <dsp: param name="businessProcessName" value="ShoppingProcess"/>
  <dsp: param name="businessProcessStage" value="ShippingPrepared"/>
  <dsp: oparam name="true">
    ...
  </dsp: oparam>
  <dsp: oparam name="false">
    ...
  </dsp: oparam>
</dsp: droplet>
```

## NamedQueryForEach

Constructs an RQL query and renders its output parameter once for each element returned by the query.

<b>Class Name</b>	atg.repository.servlet.NamedQueryForEach
<b>Component</b>	/atg/dynamo/droplet/NamedQueryForEach



## Required Input Parameters

### ***queryName***

The name of the query to execute.

### ***repository***

The Nucleus path of the repository to query.

### ***itemDescriptor***

The name of the item type to query.

## Optional Input Parameters

### ***input Params***

A comma-delimited list of values for each parameter that is required by the query. To supply the value of a request parameter, use the syntax : *param-name*.

### ***transactionManager***

The transaction manager to use. For example:

```
<dsp: param name="transactionManager"
  bean="/atg/dynamo/transaction/TransactionManager"/>
```

### ***sortProperties***

Holds a string that specifies the order to render array items. For more information on using this parameter, see [ForEach](#).

## Output Parameters

### ***index***

The zero-based index of the returned row.

### ***count***

The one-based number of the returned row.

### ***element***

A dynamic bean that has properties for accessing the values returned in the result set. To return a particular property value for each item in the result set, use the convention *element.name*, where *name* is any property name supported by the item descriptor.

### ***size***

Set to the number of returned rows.

**key**

The current key if the array is a Dictionary.

**repositoryException**

If a RepositoryException is thrown, set to the exception.

**Open Parameters**

The following open parameters control the formatting for the returned results:

**output**

Rendered once for each returned row.

**outputStart**

Rendered before any output tags if the array of returned rows is not empty.

**outputEnd**

Rendered after all output tags if the array of returned rows is not empty.

**empty**

Rendered if the array contains no rows.

**error**

Rendered if there is an error when the query executes.

**Usage Notes**

NamedQueryForEach executes a named RQL query and renders its open parameter output once for each element returned by the query. The behavior of this servlet bean is identical to [RQLQueryForEach](#); it differs only in its use of the required input parameter queryName, which is used together with the input parameters repository and itemDescriptor to specify a named query.

In the following code fragment, queryName refers to a named query that is already set up in the specified repository. The bi kePromoti on value supplied to input parameter inputParams matches a parameter in that query, and is set to the value of request parameter startDate:

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/NamedQueryForEach">
  <dsp: param name="startDate" bean="CurrentDate. dateAsTime" />
  <dsp: param name="repository"
    value="/atg/userprofil ling/Profil eAdapterRepository"/>
  <dsp: param name="itemDescriptor" value="user" />
  <dsp: param name="queryName" value="getPromoti onsByNameQuery" />
  <dsp: param name="inputParams" value="bi kePromoti on, : startDate"/>
```



```
<dsp: param name="transactionManager"
    bean="/atg/dynamo/transaction/TransactionManager"/>
...

```

---

## NavHistoryCollector

Tracks each page that a user visits and makes it easy for that user to backtrack and return to pages he or she has already visited.

<b>Class Name</b>	atg.repository.servlet.NavHistoryCollector
<b>Component</b>	Not provided with the ATG platform

### Required Input Parameters

Supply either the `item` or `itemName` parameter:

#### ***item***

The item to add to the `historyList`.

#### ***itemName***

The name of the current page that is an anchor text in a link back to the current page.

### Optional Input Parameters

#### ***navAction***

Set to the desired action: push, pop, or jump. If unset, the default action is push.

#### ***navCount***

Detects when a user clicks on the Back button in order to reset the navigation path.

### Usage Notes

NavHistoryCollector can track each page that a user visits and makes it easy for that user to backtrack and return to pages he or she has already visited. NavHistoryCollector keeps track of a user's path from the top of the site to the current location in a stack of information that is manipulated differently depending on how the user navigates through the site. For more information about different ways of catalog navigation, see *Catalog Navigation* in the *Catalog Navigation and Searching* chapter of the *ATG Commerce Administration and Development Guide*.





## Example

The Pioneer Cycling Reference Application uses the component `/atg/commerce/catalog/CatalogNavHistoryCollector` to collect the locations visited and to add them to the array of visited locations. In Pioneer Cycling, the visited locations are the repository items that represent the products and categories of the product catalog. This snippet, taken from `breadcrumbs.jsp`, invokes the `CatalogNavHistoryCollector`:

---

```
<dsp:droplet name="/atg/dynamo/droplet/Switch">
  <dsp:param name="value" param="no_new_crumb"/>
  <dsp:oparam name="true">
    </dsp:oparam>
    <dsp:oparam name="default">
      <dsp:droplet name="CatalogNavHistoryCollector">
        <dsp:param name="navAction" param="navAction"/>
        <dsp:param name="item" param="element"/>
      </dsp:droplet>
    </dsp:oparam>
  </dsp:droplet>
```

---

Although both techniques used in this code sample are specific to the Pioneer Cycling implementation, they can be applied to any site. First, notice that the required parameter `navCount` is not passed to `CatalogNavHistoryCollector`. The `navCount` parameter is set in the page that invokes this JSP snippet. Because JSP files invoked as servlets from other JSP files are in a sub-scope of their callers, they have access to the same parameters as the caller. Second, the `no_new_crumb` parameter decides whether or not to invoke the snippet. This is just a switch on a parameter passed to the page to determine whether to add the current location to the NavHistory or not. However, it shows how Pioneer Cycling addresses navigation for pages that do not represent catalog items. For example, the search page, the shopping cart, and the user profile page are not added to the NavHistory like regular catalog pages. For more information on navigation in the Pioneer Cycling demo, see *Adding Catalog Navigation* in the *Displaying and Accessing the Product Catalog* chapter of the [ATG Consumer Commerce Reference Application Guide](#).

## NodeForEach

Given a DOM node, selects all nodes that match a specified pattern and iterates over each selected node.

<b>Class Name</b>	<code>atg.droplet.xml.NodeForEach</code>
<b>Component</b>	<code>/atg/dynamo/droplet/xml/NodeForEach</code>



## Required Input Parameters

### *node*

The DOM node to pass to this servlet bean.

## Optional Input Parameters

### *select*

Sets the pattern that is used to select the nodes to iterate over. The pattern language conforms to the pattern definition provided by the XPath specification at <http://www.w3.org/TR/xpath>. If no pattern is provided, then NodeForEach selects all the children of this node.

## Output Parameters

### *index*

Set to the zero-based index of the current element of the set of nodes each time the output parameter is rendered.

### *count*

Set to the one-based index of the current element of the set of nodes each time the output parameter is rendered. The value of count for the first iteration is 1.

### *element*

Set to the current element of the set of nodes each time the index increments and the output parameter is rendered.

## Open Parameters

### *output*

Rendered once for each element in the set of nodes.

### *outputStart*

If any nodes are selected, rendered before any output elements. For example, this parameter can be used to render the table heading.

### *outputEnd*

If any nodes are selected, rendered after all output elements. For example, this parameter can be used to render text following a table.

### *empty*

Rendered if no nodes are selected.



## Usage Notes

NodeForEach selects a set of Data Object Model nodes using a pattern and renders its output parameters for each of the nodes. The NodeForEach servlet bean is based on the [ForEach](#) servlet bean and inherits its output and open parameter definitions from [ForEach](#).

## Example

See [Processing XML in a JSP](#).

# NodeMatch

Given a DOM node, selects the next node that matches a specified pattern.

<b>Class Name</b>	atg.droplet.xml.NodeMatch
<b>Component</b>	/atg/dynamo/droplet/xml/NodeMatch

## Required Input Parameters

### *node*

The DOM node to pass to this servlet bean

### *match*

The pattern that is used to select the next node. This pattern can be any XPath expression, as defined in the XPath specification. See <http://www.w3.org/TR/xpath>.

Examples of patterns:

comment()	Select a comment child of the current node
order[@id='1111']	Select the widget-order node with the id attribute 1111
order	Select a widget-order node
order[not(position()=1)]	Select any widget-order node that is a subchild that is not the first



## Output Parameters

### *matched*

The node that is matched.

## Open Parameters

### *output*

Rendered when a node is selected. The result is bound to the matched output parameter.

### *failure*

Rendered when there is a failure during the XML query.

### *empty*

Rendered when no node is selected.

### *unset*

Rendered when the node parameter is not set.

## Usage Notes

NodeMatch selects the next DOM node that matches a pattern.

## Example

See [Processing XML in a JSP](#).

# PageEventTriggerDroplet

Sends page viewed events for the current page being viewed.

<b>Class Name</b>	atg.userprofiling.PageEventTriggerDroplet
<b>Component</b>	/atg/userprofiling/SendPageEvent



## Required Input Parameters

### *pageviewed*

Set to the path of the PageViewedEvent. If this value is not set, when a PageViewedEvent is generated, the URL of the request is used for the path value. This allows someone to generate a page viewed event for a page logically named store home even if the real URL is /store/home/. The logical name might make more sense to a business manager using the Control Center Scenario Editor.

## Usage Notes

PageEventTriggerDroplet fires page viewed events using a parameter or the request URI. This servlet bean has no output or open parameters.

## Example

You can use the PageEventTriggerDroplet in a JSP through the SendPageEvents component, in two ways:

```
<dsp:droplet name="/atg/userprofiling/SendPageEvent"></dsp:droplet>
```

or:

```
<dsp:droplet name="/atg/userprofiling/SendPageEvent">  
  <dsp:param name="pageviewed" value="store home"/>  
</dsp:droplet>
```

# PipelineChainInvocation

Initiates a pipeline thread.

<b>Class Name</b>	atg.service.pipeline.servlet.PipelineChainInvocation
<b>Component</b>	/atg/dynamo/droplet/PipelineChainInvocation

## Required Input Parameters

### *pipelineManager*

Refers to the Pipeline Manager that facilitates the processor chain. If you provide no value for this parameter, the ATG platform uses the default Pipeline Manager provided in the PipelineChainInvocation defaultPipelineManager property.



## Optional Input Parameters

### ***chainId***

Refers to the pipeline chain to execute. If you provide no value, the ATG platform uses the default chain supplied by the default `chainId` property of the `PipelineChainInvocationServlet` bean.

### ***paramObject***

An object that is passed as an argument in the `runProcess` method. Leaving this parameter blank invokes the default object, which is a `HashMap` that is constructed using the `extraParametersMap` property. The `extraParametersMap` represents a mapping of new `HashMap`'s keys to the request parameters that are bound to the new `HashMap`'s values.

## Open Parameters

### ***exception***

The exception objects thrown during pipeline execution. You might want to render this parameter in conjunction with the `successWithErrors` or `failure` parameters.

### ***failure***

Rendered when serious errors are encountered during pipeline execution.

### ***pipelineResult***

Refers to the object that acts as an error log by saving any errors encountered by processors in the chain.

### ***success***

Rendered when a pipeline executes without any errors.

### ***successWithErrors***

Rendered when a pipeline thread executes to completion, but acquires errors in the `pipelineResult` object during the process.

## Usage Notes

`PipelineChainInvocation` initiates a pipeline thread from a JSP. This servlet bean sends request arguments to a Pipeline Manager; when the pipeline process is complete, it receives a `pipelineResult` object that contains a list of error messages.

For more information on Pipelines, see *Processor Chains and the Pipeline Manager* in the [ATG Commerce Programming Guide](#).



## PossibleValues

Returns a list of repository items of the specified item type.

<b>Class Name</b>	atg.repository.servlet.PossibleValues
<b>Component</b>	/atg/dynamo/droplet/PossibleValues

### Required Input Parameters

#### *itemDescriptorName*

The item descriptor within the repository.

#### *propertyName*

The name of the repository item's property, required only for enumerated properties. For properties which are themselves item types, you can set *itemDescriptorName* and *propertyName* of the linked property, or set *itemDescriptorName* to the linked-to property.

#### *repository*

The name of the repository. If omitted, the default is obtained from the servlet bean's properties file.

#### *repositoryItem*

Required only if parameters *itemDescriptorName* or *repository* are unspecified, used to obtain the item descriptor and its repository. Use this parameter if a repository item's underlying type and source are unknown.

### Optional Input Parameters

#### *maxRepositoryItems*

Sets the maximum number of repository items returned by this servlet bean. If omitted, the default is obtained from the servlet bean's *maxRepositoryItems* property (initially set to 500). To specify no limit, set this parameter to -1.

#### *returnValueObjects*

If set to true, sets the output parameter *values* to an array of *PossibleValue* objects.

*PossibleValue* objects have the following properties:

- **label:** Returns the value's display string. If no display value is found, returns the *toString* value.
- **settableValue:** The property value to set on the object, one of the following:

- For a repositoryItem, returns the repository ID.
- For an enumeration, returns either the string or integer value, depending on whether the underlying property has useCodeForValue set.
- **underlyingObject:** The underlying object, one of the following:
  - repositoryItem for a repositoryItem property
  - EnumPropertyDescriptor. EnumerationOption for an enumeration value

### sortProperties

Holds a string that specifies the order to render array items. The syntax of this parameter depends on the array item type: JavaBean, Dynamic Bean, Date, Number, or String.

To sort on the properties of a JavaBean, specify the value of sortProperties as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. To specify ascending sort order, prepend the property name with a plus + sign; to specify descending order, prepend with a minus – sign.

The following example sorts a JavaBean array alphabetically by title, then in descending order of size, use this input parameter:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

If the array contains Dates, Numbers, or Strings, prepend the sortProperties value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

In order to sort Map elements by key, set the value as follows:

```
{+|-}_key
```

For example:

```
<dsp: param name="sortProperties" value="-_key" />
```

A nested servlet bean inherits the parent servlet bean's sortProperties setting, unless it is overridden by the nested servlet bean's own sortProperties setting. For example, the following setting negates any parent sortProperties setting:

```
<dsp: param name="sortProperties" value=""/>
```

### useCodeForValue

A Boolean parameter, specifies whether to return the string or integer code value for the enumerated property identified by the propertyName parameter. Each enumerated property has a string value and integer code value specified for it in the item descriptor that defines the property itself. When





useCodeForValue is set to true for the property in its item descriptor and in the PossibleValues servlet bean, the integer code value is provided to the JSP. Any other combination of values causes the string value to return.

For information on useCodeForValue in the repository item descriptor, see the *Enumerated Properties* section of the *SQL Repository Item Properties* chapter of the [ATG Repository Guide](#).

## Output Parameters

### values

Contains the array of possible values for the named itemDescriptorName and propertyName; or an array of PossibleValue objects if input parameter returnValueObjects is set to true.

## Open Parameters

### output

Rendered once with the result of the repository query. In the case of enumerated property data types, the result is an array of java. lang. String objects. In the case of properties which are themselves item types, the result is an array of repository items.

## Usage Notes

PossibleValues uses the specified (or repositoryItem-implied) repository and item descriptor to produce an array of RepositoryItems. If you specify a property name through the propertyName parameter, PossibleValues returns an array of property values, each one an enumerated list of strings. Alternatively, you can generate an array of PossibleValue objects by setting the returnValueObjects parameter to true.

## Example

The following example uses the servlet beans PossibleValues and [ForEach](#) to create a form with input fields for setting an audio component's type and manufacture. A fragment of a repository definition XML file shows the different property types to be retrieved:

---

```
<item-descriptor name="component">
  ...
  <property name="type" data-type="enumerated" ... >
    <attribute name="useCodeForValue" value="false"/>
    <option value="Receiver" code="0"/>
    <option value="Amplifier" code="1"/>
    <option value="Tuner" code="2"/>
    <option value="CD Player" code="3"/>
    <option value="DVD Player" code="4"/>
    <option value="Cassette Tape Player" code="5"/>
  </property>
```



```
<property name="manufacturer" item-type="manufacturer" ... >

</item-descriptor>

<item-descriptor name="manufacturer">
    ...
    <property name="id" column-name="manufacturer_id"/>
    <property name="displayName" data-type="string"
        column-name="manufacturer_name"/>
    <property name="description" data-type="string" column-name="description"/>
    <property name="longDescription" data-type="big string"
        column-name="long_description"/>
</item-descriptor>
```

---

The JSP form is defined as follows:

---

```
<dsp:form action="Audio.jsp" method="POST">

<!-- Generate the Component Type select tag from an enumerated type -->
<dsp:select bean="AudioComponent.type">
    <dsp:droplet name="PossibleValues">
        <dsp:param name="itemDescriptorName" value="component"/>
        <dsp:param name="propertyName" value="type"/>
        <dsp:oparam name="output">
            <dsp:droplet name="ForEach">
                <dsp:param name="array" param="values"/>
                <dsp:oparam name="output">
                    <dsp:option param="element"/>
                    <dsp:valueof param="element"/>
                </dsp:oparam>
            </dsp:droplet>
        </dsp:oparam>
    </dsp:droplet>
</dsp:select>

<!-- Generate the Component Manufacturer select tag from an item type -->
<dsp:select bean="AudioComponent.manufacturer">
    <dsp:droplet name="PossibleValues">
        <dsp:param name="itemDescriptorName" value="component"/>
        <dsp:param name="propertyName" value="manufacturer"/>
        <dsp:param name="sortProperties" value="+displayName"/>
        <dsp:oparam name="output">
            <dsp:droplet name="ForEach">
                <dsp:param name="array" param="values"/>
                <dsp:oparam name="output">
                    <dsp:option param="id"/>
                    <dsp:valueof param="element.displayName"/>
                </dsp:oparam>
            </dsp:droplet>
        </dsp:oparam>
    </dsp:droplet>
</dsp:select>
```



```
</dsp: dropl et>
</dsp: oparam>
</dsp: dropl et>
</dsp: sel ect>

</dsp: form>
```

---

## ProfileHasLastMarker

Locates the last profile marker added to a profile.

<b>Class Name</b>	atg.markers.userprofil ling.dropl et.Profi l eHasLastMarker
<b>Component</b>	/atg/markers/userprofil ling/dropl et/Profi l eHasLastMarkerDropl et

### Required Input Parameters

#### *key*

The value in the marker's key property. The last profile marker must have the value in its key property that you specify here.

Set this parameter to ANY\_VALUE to indicate that the marker's key value is irrelevant for your purposes. For example:

```
<dsp: param name="key" bean="Profi l eHasLastMarkerDropl et. ANY_VALUE" />
```

### Optional Input Parameters

#### *itemId*

The ID of the profile whose last marker you want to access. If omitted, the value of the input parameter *item* is used.

#### *item*

The profile item whose last marker you want to access. If omitted, the active profile is used.

#### *value*

The value in the marker's val ue property. The last profile marker must have the value in its val ue property that you specify here. You can set this parameter to NULL or omit it in order to return the marker only if its val ue property is null.



Set this parameter to ANY\_VALUE in order to indicate that the marker's value is irrelevant for your purposes. For example:

```
<dsp: param name="value" bean="ProfileHasLastMarkerDropLet. ANY_VALUE" />
```

### ***data***

The value in the marker's data property. The last profile marker must have the value in its data property that you specify here. You can set this parameter to NULL or omit it in order to return the marker only if its data property is null.

Set this parameter to ANY\_VALUE to indicate that the marker's data value is irrelevant for your purposes. For example:

```
<dsp: param name="data" bean="ProfileHasLastMarkerDropLet. ANY_VALUE" />
```

### ***extendedProperties***

A map that specifies any additional marker properties (set to the map's key) and property values (set to the map's value) that must exist on the last profile marker in order for to retrieve it.

### ***markerManager***

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager indicated in the servlet bean's repositoryMarkerManager property is used.

### ***markerPropertyName***

The property on the profile to check for markers. If omitted, the servlet bean uses the default value specified in its defaultMarkerPropertyName property.

### ***markedItemType***

The type of RepositoryItem with markers to work with. If omitted, the servlet bean uses the item's defaultMarkedItemType property.

## **Output Parameters**

### ***marker***

Contains the marker located by the servlet bean.

### ***errorMsg***

Contains any error messages generated during the servlet bean's execution.

## **Open Parameters**

### ***error***

Rendered when an error occurs during servlet bean execution.

**true**

Rendered when the last profile marker added to the profile has the specified property values.

**false**

Rendered when the last profile marker added to the profile does not have the specified property values.

**Usage Notes**

ProfileHasLastMarker uses the Profile Marker Manager to determine whether the last profile marker added to a particular profile has the parameters you specify. For example, you can find out if the last marker includes specific values for its key, value, data or other properties. If the last marker has the indicated parameters, the true open parameter is rendered and marker is accessible through the marker output parameter.

For more information on working with marked profiles, see the *Using Marked Profiles* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

**Example**

The following example shows how to locate the last profile marker on the current user profile. If the profile's last profile marker has a key of partner, the servlet bean displays the partner's name, which is stored in the marker value property, followed by  
is one of our favorite affiliates. Learn why!.

---

```
<dsp: droplet
  name="/atg/markers/userprofiling/droplet/ProfileHasLastMarkerDroplet">
  <dsp: param name="key" value="partner" />
  <dsp: param name="value" bean="ProfileHasLastMarkerDroplet.ANY_VALUE" />
  <dsp: param name="data" bean="ProfileHasLastMarkerDroplet.ANY_VALUE" />

  <dsp: oparam name="true">
    <dsp: param name="marker">
      <dsp: valueof paramvalue="marker.value">
    </dsp: param>
    is one of our favorite affiliates. Learn why!
  </dsp: oparam>

</dsp: droplet>
```

---

**ProfileHasLastMarkerWithKey**

Locates the last profile marker with a particular key that was added to a profile.



<b>Class Name</b>	atg.markers.userprofiling.droplet.ProfileHasLastMarkerWithKey
<b>Component</b>	/atg/markers/userprofiling/droplet/ProfileHasLastMarkerWithKeyDroplet

## Required Input Parameters

### *key*

The value in the marker's key property. Set this parameter to ANY\_VALUE to indicate that the marker's key value is irrelevant for your purposes. For example:

```
<dsp: param name="key"
bean="ProfileHasLastMarkerWithKeyDroplet.ANY_VALUE" />
```

## Optional Input Parameters

### *itemId*

The ID of the profile with a marker to access. If omitted, the value of the input parameter *item* is used.

### *item*

The profile item that has a marker you want to access. If omitted, the active profile is used.

### *value*

The value in the marker's value property. You can set this parameter to NULL or omit it in order to return the marker only if its value property is null.

Set this parameter to ANY\_VALUE to indicate that the marker's value value is irrelevant for your purposes. For example:

```
<dsp: param name="value"
bean="ProfileHasLastMarkerWithKeyDroplet.ANY_VALUE" />
```

### *data*

The value in the marker's data property. You can set this parameter to NULL or omit it in order to return marker only if its data property is null.

Set this parameter to ANY\_VALUE, to indicate that the marker's data value is irrelevant for your purposes. For example:

```
<dsp: param name="data"
bean="ProfileHasLastMarkerWithKeyDroplet.ANY_VALUE" />
```

***extendedProperties***

A map that specifies any additional marker properties (set to the map's key) and property values (set to the map's value) that must exist on the profile marker that is retrieved.

***markerManager***

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager indicated in the servlet bean's repositoryMarkerManager property is used.

***markerPropertyName***

The property on the profile to check for markers. If omitted, the servlet bean uses the default value specified in its defaultMarkerPropertyName property.

***markedItemType***

The type of RepositoryItem that has markers you want to work with. If omitted, the servlet bean uses the default value specified in its defaultMarkedItemType property.

**Output Parameters*****marker***

Contains the marker located by the servlet bean.

***errorMsg***

Contains any error messages generated during the servlet bean's execution.

**Open Parameters*****error***

Rendered when an error occurs during servlet bean execution.

***true***

Rendered when the last marker added with a certain key matches the specified criteria.

***false***

Rendered when the last marker added with a certain key does not match the specified criteria.

**Usage Notes**

ProfileHasLastMarkerWithKey determines whether the last profile marker added to the profile with a particular key has the additional parameters you specify. These additional parameters can be specific values for the value, data, or other marker properties you specify. If the last marker with a particular key matches the other parameters you specify, the true open parameter is rendered and the marker is made available to the page through the marker output parameter.



For more information on working with marked profiles, see the *Using Marked Profiles* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

## Example

The following example shows how to determine if the last profile marker added to the current profile with a key of partner has a value of travel site A. Users who have such a marker see a message informing them about airfare discounts.

```
<dsp: droplet
  name="/atg/markers/userprofiling/droplet/ProfileHasLastMarkerWithKeyDroplet">
  <dsp: param name="key" value="partner" />
  <dsp: param name="value" value="travel Site A" />
  <dsp: param name="data" bean="ProfileHasMarkerWithKeyDroplet.ANY_VALUE" />

  <dsp: oparam name="true">
    When you book your hotel room with us, you'll receive a discount on airfare
    if you book your flight with travel site A!
  </dsp: oparam>
</dsp: droplet>
```

## ProfileHasMarker

Determines whether a profile has a profile marker.

<b>Class Name</b>	atg.markers.userprofiling.droplet.ProfileHasMarker
<b>Component</b>	/atg/markers/userprofiling/droplet/ProfileHasMarkerDroplet

## Required Input Parameters

### key

The value in the marker's key property. Set this parameter to ANY\_VALUE to indicate that the marker's key value is irrelevant for your purposes. For example:

```
<dsp: param name="key" bean="ProfileHasMarkerDroplet.ANY_VALUE" />
```





## Optional Input Parameters

### ***itemId***

The ID of a profile to check for markers. If omitted, the value indicated in the `item` input parameter is used.

### ***item***

The profile item to check for markers. If omitted, the active profile is used.

### ***value***

The value in the marker's `value` property. You can set this parameter to `NULL` or omit it if the marker's `value` property must be null.

Set this parameter to `ANY_VALUE` to indicate that the marker's `value` value is irrelevant for your purposes. For example:

```
<dsp: param name="value" bean="ProfileHasMarkerDropLet ANY_VALUE" />
```

### ***data***

The value in the marker's `data` property. You can set this parameter to `NULL` or omit it if the marker's `data` property must be null.

Set this parameter to `ANY_VALUE` to indicate that the marker's `data` value is irrelevant for your purposes. For example:

```
<dsp: param name="data" bean="ProfileHasMarkerDropLet ANY_VALUE" />
```

### ***extendedProperties***

A map that specifies any additional marker properties (set to the map's key) and property values (set to the map's `value`) that must exist on the profile marker.

### ***markerManager***

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager indicated in the servlet bean's `repositoryMarkerManager` property is used.

### ***markerPropertyName***

The property on the profile to check for markers. If omitted, the servlet bean uses the default value specified in its `defaultMarkerPropertyName` property.

### ***markedItemType***

The type of `RepositoryItem` that has markers to work with. If omitted, the servlet bean uses the default value specified in its `defaultMarkedItemType` property.



## Output Parameters

### ***errorMsg***

Contains any error messages generated during the servlet bean's execution.

## Open Parameters

### ***error***

Rendered when an error occurs during servlet bean execution.

### ***true***

Rendered when the profile has a profile marker that matches the criteria you specify.

### ***false***

Rendered when the profile does not have a marker that matches the criteria you specify.

## Usage Notes

ProfileHasMarker class uses the Profile Marker Manager to determine whether a user profile has a profile marker with the specified key, value, and data property values. In order to check whether a profile has any profile markers, set all three parameters to ANY\_VALUE.

For more information on working with marked profiles, see the *Using Marked Profiles* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

## Example

In the following example, the servlet bean checks the current user profile for partner profile markers. When none exist, an advertisement displays:

---

```
<dsp: droplet name="/atg/markers/userprofiling/droplet/ProfileHasMarkerDroplet">
  <dsp: param name="key" value="partner" />
  <dsp: param name="value" bean="ProfileHasMarkerDroplet.ANY_VALUE" />
  <dsp: param name="data" bean="ProfileHasMarkerDroplet.ANY_VALUE" />

  <dsp: oparam name="false">
    Purchase your airline tickets with Travel Site A! If you need to rent a car,
    check out the deals offered by Travel Site B!
  </dsp: oparam>

</dsp: droplet>
```

---



## ProtocolChange

Enables switching between HTTP and HTTPS protocols.

<b>Class Name</b>	atg.droplet.ProtocolChange
<b>Component</b>	/atg/dynamo/droplet/ProtocolChange

### Required Input Parameters

#### *inUrl*

A full or relative local URL that uses the HTTP or HTTPS protocol.

### Output Parameters

#### *secureUrl*

If the enable property is set to true, set to a full local URL that uses the HTTPS protocol. If the enable property is false, set to the value of *inUrl*.

#### *nonSecureUrl*

If the enable property is set to true, set to a full local URL that uses the HTTP protocol. If the enable property is false, this parameter is set to the value of *inUrl*.

### Open Parameters

#### *output*

Rendered once.

### Usage Notes

ProtocolChange takes a full or relative local URL and renders a full URL that uses either the secure HTTPS protocol or the standard protocol, depending on which output parameter you specify. You can use this servlet bean to switch between protocols. For example, on a commerce site, you might want to switch from HTTP to HTTPS before the customer enters personal information such as a credit card number.

In order to change the protocol, the servlet bean's enable property must be set to true. By default, this property is set to false. When this property is false, the servlet bean sets the *secureUrl* and *nonSecureUrl* output parameters to the exact value of *inUrl*; it does not change the protocol or change a relative URL to a full URL. This property enables you to develop your application for a secure server even if your development environment's server is not secure. To do this, leave the enable property set to false in your development environment, and set it to true in the production environment.



## Example

```
<dsp: droplet name="/atg/dynamo/droplet/Protocol Change">
  <dsp: param name="inUrl" value=".. /checkout/creditcard.jsp"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="a6" param="secureUrl" vartype="java.lang.String">
      <dsp: a href="{a6}">Enter credit card information</dsp: a>
    </dsp: getvalueof>
  </dsp: oparam>
</dsp: droplet>
```

## Range

Displays a subset of array elements.

<b>Class Name</b>	atg.droplet.Range
<b>Component</b>	/atg/dynamo/droplet/Range

## Required Input Parameters

### *array*

The list of items to output: a Collection (Vector, List, or Set), Iterator, Enumeration, Map, Dictionary, or array.

### *howMany*

The number of array elements to display. If howMany is greater than the number of array elements, Range renders the number of elements available.

## Optional Input Parameters

### *start*

The initial array element, where a setting of 1 (the default) specifies the array's first element. If start is greater than the number of elements in the array, the empty parameter (if specified) is rendered.

### *sortProperties*

A string that specifies the order in which array items are rendered. The value assigned to this parameter depends on the item type: JavaBeans, Dynamic Beans, Dates, Numbers, or Strings.



To sort on the properties of a JavaBean, specify the value of `sortProperties` as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. To specify ascending sort order, prepend the property name with a plus + sign; to specify descending order, prepend with a minus – sign.

The following input parameter sorts a JavaBean array alphabetically by title, then in descending order of size:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

If the array contains Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

In order to sort Map elements by key, set the value as follows:

```
{+|-}_key
```

For example:

```
<dsp: param name="sortProperties" value="-_key" />
```

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### *index*

The zero-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `index` for the first iteration is 0.

### *count*

The one-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `count` for the first iteration is 1.

### *key*

If the array parameter is a Map or Dictionary, set to the Map or Dictionary key.

### *element*

The current array element each time the `index` increments and the output parameter is rendered.

***size***

The total number of items in the source array.

***end***

When the result set is divided into subsets, causes the last element in a subset to display its numbered value. This parameter uses a one-based count.

***hasPrev***

Set to true before any output parameters are rendered, indicates whether the array includes any items that precede the current array set.

***prevStart***

Set before any output parameters are rendered and only if *hasPrev* is true, indicates the value of *start* that should be used to display the previous array set. Use this parameter to create a link or form submission that displays the previous elements of the array.

***prevEnd***

Set before any output parameters are rendered, and only if *hasPrev* is true, indicates the (one-based) count of the last element in the previous array set.

***prevHowMany***

Set before any output parameters are rendered, and only if *hasPrev* is true, indicates the number of elements in the previous array set.

***hasNext***

Set to true before any output parameters are rendered, indicates whether the array includes any items that follow the current array set.

***nextStart***

Set before any output parameters are rendered, and only if *hasNext* is true, indicates the value of *start* that should be used to display the next array set.

***nextEnd***

Set before any output parameters are rendered, and only if *hasNext* is true, indicates the (one-based) count of the last element in the next array set.

***nextHowMany***

Set before any output parameters are rendered, and only if *hasNext* is true, indicates the number of elements in the next array set.



## Open Parameters

### *output*

Rendered once for each element in the subset of the array defined by the `start` and `howMany` parameters.

### *outputStart*

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

### *outputEnd*

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

### *empty*

Rendered if the array or the specified subset of the array contains no elements.

## Usage Notes

Range is similar to the [ForEach](#) servlet bean, except it can render a subset of the output array rather than the entire array. Range renders its output parameter for each element in its array parameter, beginning with the array element that corresponds to the `start` parameter and continuing until it has rendered a number of elements equal to the `howMany` parameter. The array parameter can be a Collection (Vector, List, or Set), Iterator, Enumeration, Map, Dictionary, or array.

## Example

The following example displays the values of the `initialServices` property of ATG's `Initial` component. The values are displayed two at a time. This example uses a [Switch](#) servlet bean to create navigational links after each pair of values. Links are provided to the sets of values that precede and follow the current pair, if any.

---

```
<dsp:importbean bean="/atg/dynamo/dropLet/Range"/>
<dsp:importbean bean="/atg/dynamo/dropLet/Switch"/>

<dsp:dropLet name="Range">
  <dsp:param bean="/Initial.initialServices" name="array"/>
  <dsp:param name="howMany" value="2"/>
  <dsp:oparam name="outputStart"><h3>Initial Services: </h3><ul></dsp:oparam>
  <dsp:oparam name="outputEnd">
    </ul>
  <dsp:dropLet name="Switch">
    <dsp:param name="value" param="hasPrev"/>
    <dsp:oparam name="true">
      <dsp:getValueof var="a27" bean="/OriginalRequest.pathInfo"
```



```
vartype="j ava. l ang. Stri ng">
  <dsp: a href="{a27}">
    <dsp: param name="start" param="prevStart"/>
    Previous <dsp: val ueof param="prevHowMany"/>
  </dsp: a>
</dsp: getval ueof>
</dsp: oparam>
</dsp: dropl et>
&nbsp;
<dsp: dropl et name="Swi tch">
  <dsp: param name="val ue" param="hasNext"/>
  <dsp: oparam name="true">
    <dsp: getval ueof var="a46" bean="/Ori gi nati ngRequest. pathI nfo"
    vartype="j ava. l ang. Stri ng">
      <dsp: a href="{a46}">
        <dsp: param name="start" param="nextStart"/>
        Next <dsp: val ueof param="nextHowMany"/>
      </dsp: a>
    </dsp: getval ueof>
  </dsp: oparam>
</dsp: dropl et>
</dsp: oparam>
<dsp: oparam name="output">
  <l i ><dsp: val ueof param="el ement. absol uteName">&nbsp;</dsp: val ueof>
</dsp: oparam>
</dsp: dropl et>
```

## Redirect

Redirects the user to the specified page.

<b>Class Name</b>	atg. dropl et. Redi rect
<b>Component</b>	/atg/dynamo/dropl et/Redi rect

### Required Input Parameters

#### *url*

The full or relative URL of the page where users are redirected.





Usage Notes

Redirect takes a full or relative URL and redirects the user to the specified page. When Redirect is invoked, the target page displays and an IOException is thrown, thereby discarding the request.

Example

```
<dsp: droplet name="/atg/dynamo/droplet/Redirect">
  <dsp: param name="url" value="http://www.atg.com"/>
</dsp: droplet>
```

RemoveAllMarkersFromProfile

Removes all profile markers from a profile.

Class Name	atg.markers.userprofiling.droplet.RemoveAllMarkersFromProfile
Component	/atg/markers/userprofiling/droplet/RemoveAllMarkersFromProfile Droplet

Required Input Parameters

None

Optional Input Parameters

itemId

The profile ID that has profile markers to be removed. If omitted, the value indicated in the item input parameter is used.

item

The profile item that that has profile markers to be removed. If omitted, the active profile is used.

markerManager

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager indicated in the servlet bean's repositoryMarkerManager property is used.

markerPropertyName

The property on the profile whose markers are to be removed. If omitted, the servlet bean uses the default value specified in its defaultMarkerPropertyName property.



## Output Parameters

### ***markerCount***

Contains the number of profile markers removed from this servlet bean's execution.

### ***errorMsg***

Contains any error messages generated during the servlet bean's execution.

## Open Parameters

### ***output***

Rendered when all profile markers are removed from the profile.

### ***error***

Rendered when an error occurred during servlet bean execution.

## Usage Notes

`RemoveAllMarkersFromProfile` works with the Profile Marker Manager to remove all profile markers from a particular user profile.

For more information on removing profile markers, see the *Removing Profile Markers* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

## Example

The following example removes all profile markers from the current user's profile. When the markers are removed, the user sees a message that states the number of markers that were removed.

---

```
<dsp: droplet
  name="/atg/markers/userprofiling/droplet/RemoveAllMarkersFromProfileDropLet">

  <dsp: param name="output">
    There were <dsp: param name="markerCount"/> markers removed from your
    profile.
  </dsp: param>

</dsp: droplet>
```

---

## RemoveBusinessProcessStage

Removes a specified business process stage.



<b>Class Name</b>	atg.markers.bp.droplet.RemoveBusinessProcessStage
<b>Component(s)</b>	/atg/markers/bp/droplet/RemoveBusinessProcessStageDroplet

## Required Input Parameters

### ***businessProcessStage***

The stage within the business process. Rather than specify markers for a particular stage, you can remove all markers from a given process by setting this parameter to the ANY\_VALUE property as follows:

```
<dsp: param name="businessProcessStage"
  bean="RemoveBusinessProcessStageDroplet.ANY_VALUE" />
```

## Optional Input Parameters

### ***businessProcessName***

The name of the business process. Defaults to the servlet bean's default `tBusinessProcessName` property. Setting the default `tBusinessProcessName` property lets you create instances of the servlet bean that are specific to a single business process.

## Output Parameters

### ***errorMsg***

The error message describing a failure.

### ***markerCount***

The number of stage reached markers removed.

## Open Parameters

### ***output***

Rendered on successful completion.

### ***error***

Rendered on error.

## Usage Notes

`RemoveBusinessProcessStage` removes existing business process stage markers that match the stage specified.



## Example

The following example removes a ShoppingProcessVended stage:

```
<dsp: droplet name="RemoveBusinessProcessStageDropLet">
  <dsp: param name="businessProcessName" value="ShoppingProcess"/>
  <dsp: param name="businessProcessStage" value="ShoppingProcessVended"/>
</dsp: droplet>
```

The following example removes all shopping process stage markers that are found:

```
<dsp: droplet name="RemoveBusinessProcessStageDropLet">
  <dsp: param name="businessProcessName" value="ShoppingProcess"/>
  <dsp: param name="businessProcessStage"
    beanvalue="RemoveBusinessProcessStageDropLet.ANY_VALUE"/>
</dsp: droplet>
```

## RemoveMarkersFromProfile

Removes a profile marker from a profile.

<b>Class Name</b>	atg.markers.userprofiling.dropLet.RemoveMarkersFromProfile
<b>Component</b>	/atg/markers/userprofiling/dropLet/RemoveMarkersFromProfileDropLet

## Required Input Parameters

### *key*

The value in the marker's key property. Set this parameter to ANY\_VALUE to indicate that the marker's key value is irrelevant for your purposes. For example:

```
<dsp: param name="key" bean="RemoveMarkersFromProfileDropLet.ANY_VALUE"/>
```

## Optional Input Parameters

### *itemId*

The ID for the profile that has markers to be removed. If omitted, the value indicated in the item input parameter is used.

***item***

The profile item that has profile markers to be removed. If omitted, the active profile is used.

***value***

The value in the markers' value property. Set this parameter to NULL or omit it in order to remove the markers only if their value property is null.

Set this parameter to ANY\_VALUE to indicate that the markers' value are irrelevant for your purposes. For example:

```
<dsp: param name="value"
bean="RemoveMarkersFromProfileDropLet ANY_VALUE" />
```

***data***

The value in the markers' data property. Set this parameter to NULL or omit it to remove the markers only if their data property is null.

Set this parameter to ANY\_VALUE to indicate that the markers' data value is irrelevant for your purposes. For example:

```
<dsp: param name="data"
bean="RemoveMarkersFromProfileDropLet ANY_VALUE" />
```

***extendedProperties***

A map that specifies any additional marker properties (set to the map's key) and property values (set to the map's value) that must exist on the profile marker in order for it to be removed.

***markerManager***

The Profile Marker Manager component to use. If omitted, the Profile Marker Manager indicated in the servlet bean's repositoryMarkerManager property is used.

***markerPropertyName***

The property on the profile whose markers are to be removed. If omitted, the servlet bean uses the default value specified in its defaultMarkerPropertyName property.

***markedItemType***

The type of RepositoryItem with markers to be removed. If omitted, the servlet bean uses the default value specified in its defaultMarkedItemType property.

**Output Parameters*****markerCount***

Contains the number of profile markers removed from the profile during this servlet bean's execution.

***errorMsg***

Contains any error messages generated during the servlet bean's execution.

**Open Parameters*****output***

Rendered when the profile markers are removed from the profile.

***error***

Rendered when an error occurs during marker removal.

**Usage Notes**

RemoveMarkersFromProfile works with the Profile Marker Manager to remove the markers from a particular profile that have the key, value, data, and other marker property values that you specify.

For more information on removing profile markers, see the *Removing Profile Markers* section in the *Using Profile Markers* chapter of the [ATG Personalization Programming Guide](#).

**Example**

The following example shows how to remove all markers with a key of partner and a value of travel Si teA from the active user's profile.

---

```
<dsp: droplet
  name="/atg/markers/userprofiling/droplet/RemoveMarkersFromProfileDropet">

  <dsp: param name="key" value="partner"/>
  <dsp: param name="value" value="travel Si teA"/>
  <dsp: param name="data" bean="RemoveMarkersFromProfileDropet.ANY_VALUE"/>

</dsp: droplet>
```

---

## RepositoryLookup

Looks up an item in a specific repository, based on the item's ID, and renders the item on the page.

<b>Class Name</b>	atg.targeting.RepositoryLookup
<b>Component</b>	/atg/targeting/RepositoryLookup



## Required Input Parameters

### *id*

The repository ID of the repository item to look up.

### *repository*

The name of the repository that holds the item to look up.

### *url*

The JNDI URL of the repository item. Use this parameter instead of the *id* and *repository* parameters.

**Note:** You can use this parameter only if the repository is registered. To register a repository, add it to the list of repositories in the *initialRepositories* property of the `/atg/registry/ContentRepositories` component.

## Optional Input Parameters

### *itemDescriptor*

The name of the item descriptor used to load the item.

### *fireViewItemEvent*

Specifies whether to trigger a JMS view item event when a page is accessed. If set to true or omitted, an event fires. To prevent the ATG platform from firing a view item event, set this parameter to false.

### *fireContentEvent*

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires, and when the accessed page is an item in a Content repository, a content viewed event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, use the *fireViewItemEvent* parameter.

### *fireContentTypeEvent*

Specifies whether to trigger an event when a page is accessed. If this parameter is set to true or omitted, a JMS view event fires, and when the accessed page is an item in a Content repository, a content type event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, use the *fireViewItemEvent* parameter.

### *fireContentEventItemDescriptor*

Holds an item descriptor name. When the servlet bean retrieves an item, it fires a content viewed event that contains the item's type. Use this parameter to replace the item's type in the content viewed event with the item's item descriptor name.



## Output Parameters

### *element*

Set to the repository item specified by the input parameters.

## Open Parameters

### *output*

Specifies the HTML formatting when the *element* parameter is rendered.

### *empty*

Rendered if the targeting operation returns no matching items.

## Usage Notes

RepositoryLookup looks up a single repository item based on the item's ID, and renders the item on the page. To obtain a repository item's ID in the appropriate form, use its *repositoryId* property.

## Example

The Quincy Funds demo application uses RepositoryLookup in its `sendprospectus.jsp` page. This page is passed the ID number of a repository item as a page parameter named `ElementId`. This repository item holds information about a mutual fund. RepositoryLookup uses the `ElementId` parameter as its `id` input parameter, and renders the name of the requested mutual fund:

---

```
<p>You have requested the application and prospectus for the
<dsp: droplet name="/atg/targeting/RepositoryLookup">
  <dsp: param bean="/atg/demo/QuincyFunds/repositories/Funds/Funds"
name="repository"/>
  <dsp: param name="id" param="ElementId"/>
  <dsp: oparam name="output">
    <b><nobr><dsp: valueof param="element.fundName"/></b>. </nobr>
  </dsp: oparam>
</dsp: droplet>
```

---

## RQLQueryForEach

Constructs an RQL query and renders its output parameter once for each element returned by the query.





<b>Class Name</b>	atg.repository.servlet.RQLQueryForEach
<b>Component</b>	/atg/dynamo/drop1et/RQLQueryForEach

## Required Input Parameters

### *repository*

The repository to query.

### *itemDescriptor*

The name of the item type to query.

### *queryRQL*

The RQL query to execute.

## Optional Input Parameters

### *transactionManager*

The Transaction Manager to use. For example:

```
<dsp: param name="transactionManager"
  bean="/atg/dynamo/transaction/TransactionManager"/>
```

### *sortProperties*

Holds a string that specifies the order to render array items. The syntax of this parameter depends on the array item type: JavaBean, Dynamic Bean, Date, Number, or String.

To sort on the properties of a JavaBean, specify the value of `sortProperties` as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. To specify ascending sort order, prepend the property name with a plus + sign; to specify descending order, prepend with a minus - sign.

The following input parameter sorts a JavaBean array alphabetically by title, then in descending order of size:

```
<dsp: param name="sortProperties" value="+title, -size"/>
```

If the array contains Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus - sign to specify ascending or descending sort order..

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+"/>
```



In order to sort Map elements by key, set the value as follows:

`{+|-}_key`

For example:

```
<dsp: param name="sortProperties" value="-_key" />
```

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### ***index***

The zero-based index of the returned row.

### ***count***

The one-based number of the returned row.

### ***element***

A dynamic bean that has properties for accessing the values returned in the result set. To return a particular property value for each item in the result set, use the convention `element.name`, where `name` is any property name supported by the item descriptor.

### ***repositoryException***

If a `RepositoryException` is thrown, set to the exception.

## Open Parameters

The following open parameters control the formatting for the returned results:

### ***output***

Rendered once for each array element.

### ***outputStart***

Rendered before any output tags if the array is not empty.

### ***outputEnd***

Rendered after all output tags if the array is not empty.

**empty**

Rendered if the array contains no elements.

**error**

Rendered if there is an error when the query executes.

**Usage Notes**

RQLQueryForEach executes a RQL query and renders its output open parameter once for each element returned by the query. The query can be a simple query or can include parameters. The syntax for specifying a parameter is different from the syntax normally used in RQL. For example, you might want to issue this RQL query:

```
age > ?0
```

RQLQueryForEach can specify the query as follows:

```
<dsp: param name="queryRQL" value="age > :whatAge" />
```

The value for the whatAge argument is supplied through a parameter. For example, the link to the page containing the servlet bean might be:

```
<dsp: a href="http://www.example.com/over35.jsp">Click here if you are  
over 35<dsp: param name="whatAge" value="35" />  
</dsp: a>
```

For more information about RQL, see the *Repository Queries* chapter of the [ATG Repository Guide](#).

**Example**

The following example uses RQLQueryForEach to display a list of all users in the Personalization profile repository whose profiles indicate they are eligible to receive email.

---

```
<p>The following customers are eligible to receive email:
```

```
<dsp: droplet name="/atg/dynamo/droplet/RQLQueryForEach">  
  <dsp: param name="queryRQL" value="receiveEmail=true"/>  
  <dsp: param name="repository"  
    value="/atg/userprofiling/ProfileAdapterRepository"/>  
  <dsp: param name="itemDescriptor" value="user"/>  
  <dsp: oparam name="output">  
    <p><dsp: valueof param="element.lastName"/>  
  </dsp: oparam>  
</dsp: droplet>
```

---



## RQLQueryRange

Constructs an RQL query of an SQL database and renders its output parameter for a selected subset of the elements returned by the query.

<b>Class Name</b>	atg.repository.servlet.RQLQueryRange
<b>Component</b>	/atg/dynamo/dropInet/RQLQueryRange

### Required Input Parameters

#### ***repository***

The repository to query.

#### ***itemDescriptor***

The name of the item type to query.

#### ***queryRQL***

The RQL query to execute.

### Optional Input Parameters

#### ***transactionManager***

The Transaction Manager to use. For example:

```
<dsp: param name="transactionManager"
  bean="/atg/dynamo/transaction/TransactionManager"/>
```

#### ***calculateSize***

Specifies whether to calculate the number of items in the array returned by this query. This parameter must be set to true in order to use the output parameter size.

#### ***howMany***

The number of array elements to display. If howMany is greater than the number of array elements, RQLQueryRange renders the number of elements available.

#### ***start***

The initial array element, where a setting of 1 (the default) specifies the array's first element. If start is greater than the number of array elements, the empty parameter (if specified) is rendered.



### ***sortProperties***

A string that specifies how to sort the list of items in the output array. Sorting can be performed on properties of JavaBeans, Dynamic Beans, or on Dates, Numbers, or Strings.

To sort JavaBeans, specify the value of `sortProperties` as a comma-separated list of property names. The first name specifies the primary sort, the second specifies the secondary sort, and so on. If the first character of each keyword is a +, this sort is performed in ascending order. If it has a -, it is performed in descending order.

The following example sorts an output array of JavaBeans first alphabetically by `title` property and in descending order of the `size` property:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

To sort Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

When the items to sort are stored in a Map or Dictionary, use the following syntax:

```
{+|-}_name
```

where *name* is the key or Dictionary name.

For example:

```
<dsp: param name="sortProperties" value="-id" />
```

If this example involves a Map property, `id` is a key of the Map indicated in the array attribute. All IDs associated with the `id` key are returned in descending order.

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## **Output Parameters**

### ***index***

The zero-based index of the returned row.

### ***count***

The one-based number of the returned row.

***element***

A dynamic bean that has properties for accessing the values returned in the result set. To return a particular property value for each item in the result set, use the convention `element.name`, where `name` is any property name supported by the item descriptor.

***size***

Set to the number of array items returned by the query. This output parameter is set only if the input parameter `calculateSize` is set to true; otherwise, it is empty.

***end***

When the result set is divided into subsets, set to the one-based value of the last element in a subset.

***hasPrev***

Set to true before any output parameters are rendered, indicates whether the array includes any items that precede the current array set.

***prevStart***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the value of `start` to use in order to display the previous array set. You can use this parameter to create a link or form submission that displays the previous elements of the array.

***prevEnd***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the (one-based) count of the last element in the previous array set.

***prevHowMany***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the number of elements in the previous array set.

***hasNext***

Set to true before any output parameters are rendered, indicates whether the array includes any items that follow the current array set.

***nextStart***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the value of `start` that should be used to display the next array set.

***nextEnd***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the (one-based) count of the last element in the next array set.

***nextHowMany***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the number of elements in the next array set.

***repositoryException***

If a `RepositoryException` is thrown, set to the exception.

**Open Parameters**

The following open parameters control the formatting for the returned results:

***output***

Rendered once for each array element.

***outputStart***

Rendered before any output tags if the array is not empty.

***outputEnd***

Rendered after all output tags if the array is not empty.

***empty***

Rendered if the array contains no elements.

***error***

Rendered if there is an error when the query executes.

**Usage Notes**

`RQLQueryRange` executes an RQL query and renders its output open parameter for a selected subset of the elements returned by the query. The query can be a simple query or can include parameters. The syntax for specifying a parameter is different from the syntax normally used in RQL. For example, you might want to issue this RQL query:

```
age > ?0
```

`RQLQueryRange` can specify the query as follows:

```
<dsp: param name="queryRQL" value="age > :whatAge" />
```

The value for the `whatAge` argument is supplied through a parameter. For example, the link to the page containing the servlet bean might be:



```
<dsp:a href="http://www.example.com/over35.jsp">Click here if you are  
  over 35<dsp:param name="whatAge" value="35"/>  
</dsp:a>
```

For more information about RQL, see the *Repository Queries* chapter of the [ATG Repository Guide](#).

## RuleBasedRepositoryItemGroupFilter

Renders a collection of repository items that belong to the specified RuleBasedRepositoryItemGroup.

<b>Class Name</b>	atg.targeting.RuleBasedRepositoryItemGroupFilter
<b>Component</b>	/atg/targeting/RuleBasedRepositoryItemGroupFilter

### Required Input Parameters

#### *inputItems*

A collection of repository items to filter.

#### *repositoryItemGroup*

A rule-based group of repository items.

### Output Parameters

#### *filteredItems*

A collection of repository items which are a subset of the input items.

### Open Parameters

#### *output*

The list of filtered repository items.

### Usage Notes

RuleBasedRepositoryItemGroupFilter takes a collection (either an array or a List) of repository items. The servlet bean then renders a collection of repository items of the same type as the input collection but containing only the items that belong to a specific RuleBasedRepositoryItemGroup.

To use RuleBasedRepositoryItemGroupFilter, you must create a set of rules to filter the repository items and a form handler that enables users to choose the items they want to filter. For more information on





creating these pieces, see the Pioneer Cycling PartsFilterFormHandler.java file and the FilterImpl.jsp file in the ATG platform distribution.

For more information on how the Pioneer Cycling Reference Application uses the RuleBasedRepositoryItemGroup servlet bean, see the *Searching and Filtering the Catalog* section of the *Displaying and Accessing the Product Catalog* chapter in the [ATG Consumer Commerce Reference Application Guide](#).

## Example

The following example displays an array of items that are filtered based on a given set of rules:

```
<dsp:importbean bean="/FilterRulesCreator"/>
<dsp:importbean bean="/Filterer"/>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>

<dsp:droplet name="Filterer">
  <dsp:param name="inputItems" param="anArrayOfRepositoryItems"/>
  <dsp:param bean="/FilterRulesCreator.repositoryItemGroup"
    name="repositoryItemGroup"/>
  <dsp:oparam name="output">
    <dsp:droplet name="ForEach">
      <dsp:param name="array" param="filteredItems"/>
      <!-- other code to display repository item properties -->
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

## SharingSitesDroplet

Returns sites in a sharing group.

<b>Class Name</b>	atg.droplet.multiplesite.SharingSitesDroplet
<b>Component</b>	/atg/dynamo/droplet/multiplesite/SharingSitesDroplet

## Input Parameters

### *siteld*

A member site of the sharing group returned by this servlet bean. If omitted, SharingSitesDroplet uses the current site.

***excludeInputSite***

If set to true, specifies to exclude the site ID-specified site or current site from returned sites.

***includeActiveSites***

If set to true, specifies to return only sites that are active.

***shareableTypeId***

The ID property of the ShareableType component that is configured for the sharing group returned by this servlet bean. If omitted, SharingSitesDroplet uses its configured ShareableType property.

**Output Parameters*****output***

Rendered if any sites are found.

***error***

Rendered if an error occurs.

***empty***

Rendered if no sites are returned.

***sites***

A collection of sites in the sharing group. share the ShareableType component specified by input parameter shareableTypeId.

***errorMessage***

If an error occurs, set to the error message.

**Usage Notes**

Given a ShareableType ID and site ID, this servlet bean identifies a sharing group and returns its member sites. If no site ID is supplied, the current site is used. You can constrain output to active sites by setting `includeActiveSites` to true. If desired, you can also exclude the site ID-specified site or current site, by setting `excludeInputSite` to true.

The output parameter `sites` returns a collection of the sharing sites; you typically iterate over these with the servlet bean [ForEach](#).

To test whether two sites are in the same sharing group, use the servlet bean [SitesShareShareableDroplet](#).



## SiteContextDroplet

Establishes a site context for this servlet bean's output.

<b>Class Name</b>	<code>atg.droplet.multiplesite.SiteContextDroplet</code>
<b>Component</b>	<code>/atg/dynamo/droplet/multiplesite/SiteContextDroplet</code>

### Input Parameters

#### *siteId*

The site ID of the requested site context, ignored if `emptySite` is set to `true`. This parameter must not be null if the `emptySite` parameter is set to `false`.

#### *emptySite*

If set to `true`, creates an empty site context. The default setting is `false`.

### Output Parameters

#### *output*

Rendered once for the specified site context.

#### *error*

Rendered if the requested site context cannot be found or an empty site context cannot be created.

#### *errorMessage*

If an error occurs, contains the error message.

#### *siteContext*

Set to the `siteContext` object of the specified site ID.

### Usage Notes

The `SiteContextDroplet` changes the site context from the supplied site ID and its output parameter acts as a wrapper for the new site context. Before the output parameter executes, the `SiteContextManager` uses the site ID to push a new `SiteContext` object onto the `SiteContext` stack and establish it as the new site context. This site context remains in effect until `SiteContextDroplet` exits; the `SiteContextManager` then pops the site context off the stack and restores the previous site context to the page.

You can use `SiteContextDroplet` to set an empty site context by setting `emptySite` to `true`. An empty site context removes all site context information from the page for the duration of output parameter



execution. While the site context is empty, the page has access to the data of all repository items, regardless of how their `siteIds` property is set.

If desired, you can make an entire page ignore site-specific settings in repository items by wrapping it entirely inside `SiteContextDroplet`'s output parameter and setting `emptySite` to `true`. In this way, you can ensure that behavior of a given page in a multisite application conforms to its behavior in a non-multisite application. For example, given multiple store sites, you might want to set an empty context in order to promote certain product items across all sites.

## Example

The following example changes the site context:

```
<dsp: page>
<%-- change site context to the supplied site parameter --%>

<dsp: getvalueof var="newSite" param="site"/>

<dsp: droplet name="/atg/dynamo/droplet/multisite/SiteContextDroplet">
  <dsp: param name="siteId" value="${newSite}" />
  <dsp: oparam name="output">
    You might also be interested in this:
    <dsp: valueof param="cross-sell.description" />\! <BR/>
  </dsp: oparam>
  <dsp: oparam name="error">
    The following error occurred:
    <dsp: valueof param="errorMessage" />
  </dsp: oparam>
</dsp: droplet>

</dsp: page>
```

## SiteIdForItem

Obtains a site ID from a `RepositoryItem`.

<b>Class Name</b>	<code>atg.droplet.multisite.SiteIdForItemDroplet</code>
<b>Component</b>	<code>/atg/dynamo/droplet/multisite/SiteIdForItem</code>



## Required Input Parameters

### *item*

The RepositoryItem to evaluate for sites. By default, this servlet bean queries the item's `siteIds` property. You can specify a different property to query by setting the servlet bean property `sitesPropertyName`.

## Optional Input Parameters

### *includeInactiveSites*

If `true`, includes inactive sites among the sites that SiteIdForItem can return. Set this parameter and `includeDisabledSites` to `true` in order to allow return of a site that is both inactive and disabled.

By default, only active sites are returned.

### *includeDisabledSites*

If `true`, includes disabled sites among the sites that SiteIdForItem can return. To include disabled sites, `includeInactiveSites` must also be set to `true`.

By default, only enabled sites are returned.

### *currentSiteFirst*

If `true` (the default value), gives precedence to the current site.

### *siteId*

Specifies a site to have precedence, if it is among the site IDs set in the `item`-specified RepositoryItem.

### *shareableTypeId*

The ID of a ShareableType that is registered with the SiteGroupManager. This parameter specifies to give precedence to sites within a site group that share a Nucleus component of this ShareableType, over sites that are outside the group.

## Output Parameters

### *siteId*

The site ID returned by this servlet bean.

### *offsite*

Set to `true` if the returned site ID is not of the current site.

### *active*

Set to `true` if the returned site ID is for an active site.

***enabled***

Set to `true` if the returned site ID is for an enabled site.

***errorMessage***

Set if this servlet bean cannot return a site ID, or another error occurs.

***inGroup***

Set to `true` if the returned site belongs to a shareableType-specified sharing group that includes the current site.

**Note:** If shareableType is null, inGroup always returns true. Because no sharing group is specified, all sites are eligible for selection, subject to other input parameter settings.

**Usage Notes**

SiteldForItem returns a site ID from the list of site IDs that are stored in the specified RepositoryItem's siteIds property. For example, you might define cross-sell and upsell lists that include items that belong to multiple sites. When a user selects an item, SiteldForItem can be invoked in order to determine the best site ID to return for the selected item.

***Site Selection Algorithm***

You can modify SiteldForItem's site selection algorithm by setting one or more input parameters. By default, SiteldForItem only returns an enabled and active site. If desired, you can allow return of a site that is inactive and enabled by setting the input parameters includeInactiveSites to `true`. You can also allow return of a site that is both inactive and disabled by setting the input parameters includeInactiveSites and includeDisabledSites to `true`. By default, both input parameters are set to `false`.

Among all sites that are eligible to be returned, SiteldForItem determines its selection by prioritizing the sites as follows:

1. The only site in the item's siteIds list. This site is always returned regardless of the settings for includeInactiveSites and includeDisabledSites.
2. The current site, if input parameter currentSiteFirst is set to `true`.
3. The site ID specified by input parameter siteId.
4. A site in a sharing group specified by input parameter shareableType.

A site's configured priority can affect its precedence in the SiteldForItem selection process. For more information, see [Site Priority and Precedence](#) in Chapter 4, [Coding a Page for Multiple Sites](#).

***sitesPropertyName***

By default, SiteldForItem assumes that repository items store site IDs in their siteIds property. If a repository item uses a different property to store site IDs, you can use SiteldForItem to obtain site IDs from that property by setting its name in the servlet bean's sitesPropertyName property.



### ***SiteldForItem versus SiteldForCatalogItem***

SiteldForItem and the ATG Commerce servlet bean SiteldForCatalogItem (Nucleus component /atg/commerce/multi site/SiteldForCatalogItem/) use the same class, atg.droplet.multi site.SiteldForItemDroplet. As installed, they are almost identical, differing in only one respect: SiteldForCatalogItem sets its shareableType property to atg.ShoppingCart. SiteldForItem can achieve the same results if its ShareableType input parameter is set to atg.ShoppingCart.

### **Example**

The following example shows how you might use SiteldForItem with [SiteLinkDroplet](#) to identify a product's site ID, then generate an absolute URL to the product on that site. The generated URL is used to set a request-scope variable that other JSPs can access:

---

```
<dsp: page>
<%--
    Generate a cross site URL and set in request scope parameter 'siteLinkUrl'

    Expected page parameters:
    product - product item
    siteld - the site to use in generated URL; if not supplied, use SiteldForItem
              to obtain the site ID from the product item.
--%>
<dsp: importbean bean="/atg/dynamo/droplet/multi site/SiteLinkDroplet" />
<dsp: importbean bean="/atg/commerce/multi site/SiteldForItem" />

<dsp: getvalueof var="product" param="product" />
<dsp: getvalueof var="siteld" param="siteld" />

<c: choose>

    <%-- No site ID supplied, so get one from SiteldForItem --%>
    <c: when test="{empty siteld}">
        <dsp: droplet name="SiteldForItem">
            <dsp: param name="item" param="product" />
            <dsp: param name="shareableType" value="atg.ShoppingCart" />
            <dsp: oparam name="output">
                <dsp: getvalueof var="productSiteld" param="siteld" />

                <dsp: droplet name="SiteLinkDroplet">
                    <dsp: param name="siteld" value="{productSiteld}" />
                    <dsp: param name="path" param="product.template.url" />
                    <dsp: oparam name="output">
                        <dsp: getvalueof var="siteLinkUrl" scope="request" param="url" />
                    </dsp: oparam>
                </dsp: droplet>
            </dsp: oparam>
        </dsp: droplet>
    </c: when>
</c: choose>
```



```
</c: when>

<%-- Site ID supplied --%>
<c: otherwise>
  <dsp: droplet name="SiteLinkDroplet">
    <dsp: param name="siteId" value="{siteId}" />
    <dsp: param name="path" param="product.template.url" />
    <dsp: oparam name="output">
      <dsp: getvalueof var="siteLinkUrl" scope="request" param="url" />
    </dsp: oparam>
  </dsp: droplet>
</c: otherwise>
</c: choose>

</dsp: page>
```

## SiteLinkDroplet

Generates an absolute URL to a registered site.

<b>Class Name</b>	atg.droplet.multiplesite.SiteLinkDroplet
<b>Component</b>	/atg/dynamo/droplet/multiplesite/SiteLinkDroplet

### Input Parameters

#### ***siteId***

Identifies the site used to generate the URL. If set to null, the generated URL uses the URL of the current site.

#### ***path***

Path string to include in the generated URL.

#### ***queryParams***

Query parameters to include in the generated URL.

#### ***protocol***

The protocol to use—http or https—in the generated link. If omitted, the protocol is set from the SiteURLManager property defaultProtocol.



***inInclude***

A Boolean flag, specifies whether relative paths should use the URL of the included page.

**Output Parameters*****output***

Rendered if the URL is generated.

***url***

Set to the generated URL.

***error***

Rendered if an error occurs.

***errorMessage***

Contains the error message if an error occurs.

**Usage Notes**

Given a site ID and optional path, SiteLinkDroplet generates an absolute URL to another site. SiteLinkDroplet constructs URLs for a requested site according to the following rules:

- No path supplied: New URL retains old path relative to the new site's root.
- Path contains leading forward slash (/): Path is relative to the new site's root.
- Path omits leading forward slash (/): Path is relative to the current page, on the new site.

URL construction varies depending on whether production site URLs follow a domain-based or path-based strategy. The following examples show how SiteLinkDroplet generates URLs according to these different strategies.

***Production-site URLs use path-based strategy***

Target site's production site URL:

/foo

Current URL:

http://domain.com/bar/directory/index.jsp

Input path	New URL
""	http://domain.com/foo/directory/index.jsp
/	http://domain.com/foo



Input path	New URL
/path/hel p. j sp	http: //domai n. com/foo/path/hel p. j sp
path/hel p. j sp	http: //domai n. com/foo/di r/path/hel p. j sp

***Production-site URLs use domain-based strategy***

Target site's production site URL:  
foo. com

Current URL:  
http: //domai n. com/di r/i ndex. j sp

Input path	New URL
""	http: //foo. com/di r/i ndex. j sp
/	http: //foo. com
/path/hel p. j sp	http: //foo. com/path/hel p. j sp
path/hel p. j sp	http: //foo. com/di r/path/hel p. j sp

***Preview query parameters***

If a site link is generated on a preview server—that is, a server that is assembled with the `-preview` switch—SiteLinkDroplet automatically appends two *sticky site* query parameters to the generated link: `pushSite` and `stickySite`. For more information about sticky site parameters, see the Multisite Request Processing chapter in the [ATG Programming Guide](#).

**Note:** If a site ID contains unconventional characters—for example, @ (at) and ~ (tilde)—you can enter them directly for the input parameter `siteId`. However, settings for input parameters `path` and `queryParams` must use URL encoding for non-alphanumeric characters—for example, %40 and %7E for at (@) and tilde (~) characters, respectively.

**Example**

See [SiteIdForItem](#).

## SitesShareShareableDroplet

Tests whether two sites are in the same sharing group.



<b>Class Name</b>	atg.droplet.multiplesites.ShareableDroplet
<b>Component</b>	/atg/dynamo/droplet/multiplesites/ShareableDroplet

## Required Input Parameters

### *otherSiteId*

The site to test with the siteId-specified site.

## Optional Input Parameters

### *siteId*

The site to test with the otherSiteId-specified site. If omitted, SitesShareableDroplet uses the current site.

### *shareableTypeId*

The ShareableType of the sharing group to query. If this parameter is null, SitesShareableDroplet uses the value configured in its ShareableType property.

## Output Parameters

### *true*

Rendered if the sites are in the same sharing group.

### *false*

Rendered if the sites are not in the same sharing group.

### *error*

Rendered if an error occurs.

### *errorMessage*

If an error occurs, set to the error message.

## Usage Notes

Given two site IDs and a ShareableType that identifies a sharing group, SitesShareableDroplet tests whether the two sites are members of that group. For example, before changing the site context on a given page, you might first want to check whether the current site and new site share the same shopping cart.

If no value is supplied for the input parameter siteId, the current site is used. If the same ShareableType is tested on multiple pages, you can configure a default ShareableType for this servlet bean.



To get all sites within the same sharing group, use [SharingSitesDroplet](#).

## Example

The following example tests whether the current site and rcSite share the same shopping cart:

```
<dsp: droplet name="SitesShareShareableDroplet">
  <dsp: param name="shareableId" value="ShoppingCart"/>
  <dsp: param name="otherSiteId" value="rcSite"/>
  <dsp: oparam name="true">
    The current site shares a shopping cart with Really Cool Site!
  </dsp: oparam>
  <dsp: oparam name="false">
    The current site doesn't share a shopping cart with Really Cool Site!
  </dsp: oparam>
  <dsp: oparam name="error">
    Only called if the SiteGroupManager isn't set.
    The following error occurred:
    <dsp: valueof param="errorMessage"/>
  </dsp: oparam>
</dsp: droplet>
```

## SQLQueryForEach

Constructs a query of an SQL database and renders its output parameter once for each row returned by the database query.

<b>Class Name</b>	atg.droplet.sql.SQLQueryForEach
<b>Component</b>	/atg/dynamo/droplet/SQLQueryForEach

## Required Input Parameters

You must specify one of the following sets of input parameters:

- queryBean
- querySQL input parameter with either connect ionURL or dataSource and transacti onManager

### *queryBean*

A component of class atg.servi ce.uti l . SQLQuery that encapsulates the query and data source. For example:



```
<dsp: param name="queryBean" bean="/mySite/MySQLQuery" />
```

### ***querySQL***

The SQL statement used to query the database. For example:

```
<dsp: param name="querySQL" value="select * from person  
where age=:myAge(INTEGER)" />
```

When you specify an SQL parameter that should not be treated as a string or VARCHAR, specify the parameter's SQL data type in parentheses, following the parameter name, as in the previous example.

If you specify the querySQL parameter, you must also supply either the connect i onURL parameter or the dataSource parameter.

### ***connectionURL***

Identifies the database to query. For example:

```
<dsp: param name="connect i onURL" value="jdbc:atgpool:Connect i onPool" />
```

### ***dataSource***

Identifies the Java Transaction DataSource to query. For example:

```
<dsp: param name="dataSource"  
bean="/atg/dynamo/servi ce/j dbc/JTDataSource" />
```

If you specify the dataSource parameter, you must also supply the transact i onManager parameter.

### ***transactionManager***

Identifies the Transaction Manager for the DataSource. For example:

```
<dsp: param name="transact i onManager"  
bean="/atg/dynamo/transact i on/Transact i onManager" />
```

## **Optional Input Parameters**

### ***sortProperties***

A string that specifies how to sort the list of items in the output array. Sorting can be performed on properties of JavaBeans, Dynamic Beans, or on Dates, Numbers, or Strings.

To sort JavaBeans, specify the value of sortPropert i es as a comma-separated list of property names. The first name specifies the primary sort, the second specifies the secondary sort, and so on. If the first character of each keyword is a +, this sort is performed in ascending order. If it has a -, it is performed in descending order.

The following example specifies to sort an output array of JavaBeans alphabetically by t i t l e property and in descending order of the si ze property:



```
<dsp: param name="sortProperties" value="+title, -size" />
```

To sort Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus – sign to specify ascending or descending sort order. The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

When the items to sort are stored in a Map or Dictionary, use the following syntax:

```
{+|-}_name
```

where *name* is the key or Dictionary name.

For example:

```
<dsp: param name="sortProperties" value="-id" />
```

If this example involves a Map property, `id` is a key of the Map indicated in the array attribute. All IDs associated with the `id` key are returned in descending order.

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### *index*

The zero-based index of the returned row.

### *count*

The one-based number of the returned row.

### *element*

A dynamic bean that has properties for accessing the values returned in the result set. You can access the values either by name or by column index. The name refers to either the column name or the column alias if one is used in the query. To access values by name, use `element.name`. To access values by column index, use `element.column[i]` where *i* is the zero-based index of the returned column.

## Open Parameters

The following open parameters control the formatting for the returned results:

***outputStart***

Rendered before any output tags if the array is not empty.

***outputEnd***

Rendered after all output tags if the array is not empty.

***output***

Rendered once for each array element.

***empty***

Rendered if the array contains no elements.

***error***

Rendered if there is an error when the query executes. Errors are accessed by this parameter as `SQLException` objects.

**Usage Notes**

`SQLQueryForEach` executes an SQL query and renders its output open parameter once for each element returned by the query. You can specify a query in several ways:

- Use the `queryBean` parameter to specify an `SQLQuery` bean, which encapsulates the query and the data source.
- Specify the query directly with the `querySQL` input parameter, and identify the data source either by specifying the `connecti onURL` parameter or by specifying the `dataSource` and `transacti onManager` parameters.

You might prefer to use the servlet bean [RQLQueryForEach](#), as it provides enhanced performance and scalability by using ATG's caching capabilities.

**Example**

The following example displays the number of votes each mountain received. This is done by counting the number of rows in the `MOUNTAINVOTE` table where the `mountai n` column is the same as the `mountai n. name` parameter.

---

```
<dsp: dropl et name="/atg/dynamo/dropl et/SQLQueryForEach">
  <dsp: param name="dataSource" bean="/atg/dynamo/servi ce/j dbc/JTDataSource" />
  <dsp: param name="transacti onManager"
    bean="/atg/dynamo/transacti on/Transacti onManager" />
  <dsp: param name="querySQL"
    value="select count(*) from MOUNTAI NVOTE where mountai n=: mountai n. name" />

  <dsp: oparam name="output">
    <!-- number="#" prevents us from seeing the decimal of count since
```



```
        some DB's return floating point from the query -->
        <dsp: valueof param="element.column[0]" number="#"></dsp: valueof>
    </dsp: oparam>
</dsp: droplet>
```

---

## SQLQueryRange

Constructs a query of an SQL database and renders its output parameter once for each of a specified range of the rows returned by the database query.

<b>Class Name</b>	atg.droplet.sql.SQLQueryRange
<b>Component</b>	/atg/dynamo/droplet/SQLQueryRange

### Required Input Parameters

You must specify one of the following sets of input parameters:

- queryBean
- querySQL input parameter with either connect ionURL or dataSource and transact i onManager

#### **queryBean**

Component of class atg.servi ce. uti l . SQLQuery that encapsulates the query and data source. For example:

```
<dsp: param name="queryBean" beanval ue="/mySi te/MySQLQuery" />
```

#### **querySQL**

The SQL statement used to query the database. For example:

```
<dsp: param name="querySQL" val ue="sel ect * from person
  where age=: myAge(I NTEGER)" />
```

When you specify an SQL parameter that should not be treated as a string or VARCHAR, specify the parameter's SQL data type in parentheses following the parameter name, as in the above example.

If you specify the querySQL parameter, you must also supply either the connect i onURL parameter or the dataSource parameter.

#### **connectionURL**

Identifies the database to perform the query against. For example:





```
<param name="connecti onURL" val ue="j dbc: atgpoo l : Connecti onPool ">
```

### ***dataSource***

Identifies the Java Transaction DataSource to perform the query against. For example:

```
<dsp: param name="dataSource"  
  bean="/atg/dynamo/servi ce/j dbc/JTDataSource"/>
```

If you specify the `dataSource` parameter, you must also supply the `transacti onManager` parameter.

### ***transactionManager***

Identifies the Transaction Manager for the DataSource. For example:

```
<dsp: param name="transacti onManager"  
  bean="/atg/dynamo/transacti on/Transacti onManager"/>
```

## **Optional Input Parameters**

### ***howMany***

The number of array elements to display. If `howMany` is greater than the number of array elements, `SQLQueryRange` renders the number of elements available.

### ***start***

The initial array element, where a setting of 1 (the default) specifies the array's first element. If `start` is greater than the number of elements in the array, the empty parameter (if specified) is rendered.

### ***sortProperties***

A string that specifies how to sort the list of items in the output array. Sorting can be performed on properties of JavaBeans, Dynamic Beans, or on Dates, Numbers, or Strings.

To sort JavaBeans, specify the value of `sortPropert i es` as a comma-separated list of property names. The first name specifies the primary sort, the second specifies the secondary sort, and so on. If the first character of each keyword is a +, this sort is performed in ascending order. If it has a -, it is performed in descending order.

The following example sorts an output array of JavaBeans first alphabetically by `ti tle` property and second in descending order of the `si ze` property:

```
<dsp: param name="sortPropert i es" val ue="+ti tle, -si ze"/>
```

To sort Dates, Numbers, or Strings, prepend the `sortPropert i es` value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortPropert i es" val ue="+"/>
```



When the items to sort are stored in a Map or Dictionary, use the following syntax:

`{+|-}_name`

where *name* is the key or Dictionary name.

If this example involved a Map property, *id* is a key of the Map indicated in the array attribute. All IDs associated with the *id* key are returned in descending order.

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### ***index***

The zero-based index of the returned row.

### ***count***

The one-based number of the returned row.

### ***element***

A dynamic bean that has properties for accessing the values returned in the result set. You can access the values either by name or by column index. The name refers to either the column name or the column alias if one is used in the query. To access values by name, use `element.name`. To access values by column index, use `element.column[i]` where *i* is the zero-based index of the returned column.

### ***hasPrev***

Set to true before any output parameters is rendered. It indicates whether there are any array items before the current array set.

### ***prevStart***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the value of *start* that should be used to display the previous array set. You can use this parameter to create a link or form submission that displays the previous elements of the array.

### ***prevEnd***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the (one-based) count of the last element in the previous array set.

***prevHowMany***

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the number of elements in the previous array set.

***hasNext***

Set to true before any output parameters are rendered, indicates whether there are any array items after the current array set.

***nextStart***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the value of `start` that should be used to display the next array set.

***nextEnd***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the (one-based) count of the last element in the next array set.

***nextHowMany***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the number of elements in the next array set.

**Open Parameters*****outputStart***

Rendered before any output tags if the array is not empty.

***outputEnd***

Rendered after all output tags if the array is not empty.

***output***

Rendered once for each array element.

***empty***

Rendered if the array contains no elements.

***error***

Rendered if there is an error when the query executes. Errors are accessed by this parameter as `SQLException` objects.

**Usage Notes**

`SQLQueryRange` executes an SQL query and renders its output open parameter for a selected subset of the elements returned by the query. You can specify a query in two ways:



- Use the queryBean parameter to specify an SQLQuery bean, which encapsulates the query and the data source.
- Specify the query directly with the querySQL input parameter, and identify the data source either by specifying the connect i onURL parameter or by specifying the dataSource and transact i onManager parameters.

## Example

SQLQueryRange queries a database that includes entries for people listed in a table named SKIER. It renders the skiers in the table five at a time, as specified by the howMany parameter.

The following example uses [Switch](#) servlet beans, together with the hasPrev and hasNext parameters, to render links to pages that display entries from the SKIER table that are before or after the five entries specified by the range. So, if the start I ndex parameter is 4, this example displays skiers 4 – 9 and Previ ous 3 and Next 5 links (assuming the SKIER table has at least 14 rows).

---

```
<dsp: dropl et name="SQLQueryRange">
  <dsp: param name="dataSource" bean="/atg/dynamo/servi ce/j dbc/JTDataSource" />
  <dsp: param name="transact i onManager"
    bean="/atg/dynamo/transact i on/Transact i onManager" />
  <dsp: param name="querySQL" value="select * from SKIER order by name" />
  <dsp: param name="start" value="start I ndex 1" />
  <dsp: param name="howMany" value="5" />
  <!--
    SQLQueryRange renders this "output" parameter once for each row
    in the SKIER table, 5 skiers at a time. The startIndex parameter is
    passed in as an HTML query parameter using the "prevStart" and
    "nextStart" parameters set by SQLQueryRange.
  -->
  <dsp: oparam name="outputStart">
    <dsp: dropl et name="Swi tch">
      <dsp: param name="value" param="hasPrev" />
      <dsp: oparam name="true">
        <tr><td col span=5>
          <dsp: a href="I i stPeopl e. j sp">
            <dsp: param name="start I ndex" param="prevStart" />
            Previous <dsp: val ueof param="prevHowMany"></dsp: val ueof>
          </dsp: a>
        </td></tr>
      </dsp: oparam>
    </dsp: dropl et>
  </dsp: oparam>

  <dsp: oparam name="output">
    <dsp: i ncl ude src="di spl ayPerson. j sp">
      <dsp: param name="person" param="el ement" />
    </dsp: i ncl ude>
  </dsp: oparam>
```



```
<dsp:oparam name="outputEnd">
  <dsp:droplet name="Switch">
    <dsp:param name="value" param="hasNext"/>
    <dsp:oparam name="true">
      <tr><td colspan=5>
        <dsp:a href="listPeople.jsp">
          <dsp:param name="startIndex" param="nextStart"/>
          Next <dsp:valueof param="nextHowMany"></dsp:valueof>
        </dsp:a>
      </td></tr>
    </dsp:oparam>
  </dsp:droplet>
</dsp:oparam>
</dsp:droplet>
```

---

## Switch

Displays one of several possible outputs, depending on input parameter value.

<b>Class Name</b>	atg.droplet.Switch
<b>Component</b>	/atg/dynamo/droplet/Switch

### Required Input Parameters

#### *value*

The test value used to determine which output parameter to execute. The Switch value is turned into a string and used as the name of the parameter to render. For example, if the value parameter is a Boolean that is set to true, the Switch servlet bean executes the true output parameter.

### Open Parameters

You can define an open parameter for each possible value of the value input parameter. For example, if the Switch value is a Boolean, you can define open parameters true and false as follows:

---

```
<dsp:oparam name="false">
  ...
</dsp:oparam>

<dsp:oparam name="true">
```



```
...
</dsp:oparam>
```

---

**Note:** Open parameter names cannot embed dot (.) or backslash (/) characters in their names or values.

### ***unset***

The value to render if the value parameter is empty.

### ***default***

Rendered if the value parameter is undefined or its value (converted to a String) does not match any output parameters.

## **Usage Notes**

Switch conditionally renders one of its open parameters based on the value of its value input parameter. This value is turned into a string and used as the name of the parameter to render for this case. You can use Switch to compare two dynamically defined values by setting the name of one open parameter to one value and the Switch servlet bean's value parameter to the other.

## **Example**

In this example, the ATG platform imports the Switch servlet bean with `dsp:importbean`. For each person, this example outputs a row in a table that lists the person's name, skill level, and whether the person has a car. Depending on the value of the person. hasCar parameter, Switch displays the capacity of the person's car. A second use of Switch outputs 1 person if the car capacity is one, and n people if the car capacity is a different value.

---

```
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>

<tr>
  <td valign=top><dsp:valueof param="person.name"/></td>
  <td valign=top><dsp:valueof param="person.skillLevel"/></td>
  <td valign=top>
    <dsp:droplet name="Switch">
      <dsp:param name="value" param="person.hasCar"/>
      <dsp:oparam name="false">
        none
      </dsp:oparam>
      <dsp:oparam name="true">
        can carry <dsp:valueof param="person.carCapacity"/>
      <dsp:droplet name="Switch">
        <dsp:param name="value" param="carCapacity"/>
        <dsp:oparam name="1"> person</dsp:oparam>
        <dsp:oparam name="default"> people</dsp:oparam>
      </dsp:droplet>
    </dsp:oparam>
  </td>
</tr>
```



```

        </dsp: dropl et>
    </td>
</tr>

```

## TableForEach

Displays each element of an array, arranging the output in a two-dimensional format.

<b>Class Name</b>	atg. dropl et. Tabl eForEach
<b>Component</b>	/atg/dynamo/dropl et/Tabl eForEach

### Required Input Parameters

#### *array*

The list of items to output: a Collection (Vector, List, or Set), Iterator, Enumeration, Map, Dictionary, or array.

#### *numColumns*

The number of columns used to display the array.

### Optional Input Parameters

#### *sortProperties*

A string that specifies the order to render array items. The syntax of this parameter depends on the array item type: JavaBean, Dynamic Bean, Date, Number, or String.

To sort on the properties of a JavaBean, specify the value of `sortProperti es` as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. Prepend the property name with a plus + or minus – sign in order to specify ascending or descending sort order.

For example, the following input parameter specifies to sort an array of JavaBeans alphabetically by title, then in descending order of si ze:

```
<dsp: param name="sortProperti es" val ue="+ti tle, -si ze"/>
```

If an array consists of Dates, Numbers, or Strings, prepend the `sortProperti es` value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:



```
<dsp: param name="sortProperties" value="+"/>
```

In order to sort Map elements by key, set the value as follows:

```
{+|-}_key
```

For example:

```
<dsp: param name="sortProperties" value="-_key"/>
```

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value=""/>
```

## Output Parameters

### ***index***

Set to the zero-based index of the current array element each time the output parameter is rendered.

### ***count***

Set to the one-based index of the current array element each time the output parameter is rendered. The value of `count` for the first iteration is 1.

### ***rowIndex***

Set to the current row index each time an output parameter is rendered.

### ***columnIndex***

Set to the current column index each time an output parameter is rendered.

### ***key***

If the array parameter is a Map or Dictionary, set to the Map or Dictionary key.

### ***element***

Set to the current array element each time the output parameter is rendered. If the array size is not an even multiple of the number of columns, `element` is set to null for the remaining items in the last row.

## Open Parameters

### ***output***

Rendered once for each array element.



***outputStart***

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render the table heading.

***outputEnd***

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render text following a table.

***outputRowStart***

If the array is not empty, rendered once for each row in the output at the beginning of the row. For example, this parameter can be used to render the row heading of a table.

***outputRowEnd***

If the array is not empty, rendered once for each row in the output at the end of the row. For example, this parameter can be used to render text following a row in a table.

***empty***

Rendered if the array contains no elements.

**Usage Notes**

TableForEach is an extension of the servlet bean [ForEach](#). TableForEach renders its output parameter in a two-dimensional format, such as a table. The array parameter can be a Collection (Vector, List, or Set), Iterator, Enumeration, Map, Dictionary, or array. The numColumns parameter specifies the number of columns across which to display the array. The elements of the array are arranged across the columns.

The output parameter is rendered for each cell in the table. If the array size is not a multiple of the number of columns, output is rendered with a null element parameter for the remaining columns in the last row. A [Switch](#) servlet bean can be used to conditionally fill in the missing items in the row.

**Example**

The following example displays the values of the `initialServices` property of the ATG platform's `Initial` component. The values are displayed in a two-column table. The `outputStart`, `outputEnd`, `outputRowStart`, and `outputRowEnd` open parameters define the table formatting.

---

```
<dsp:importbean bean="/atg/dynamo/droplet/TableForEach"/>

<dsp:droplet name="TableForEach">
  <dsp:param bean="/Initial.initialServices" name="array"/>
  <dsp:param name="numColumns" value="2"/>
  <dsp:oparam name="outputStart"><table border=1></dsp:oparam>
  <dsp:oparam name="outputEnd"></table></dsp:oparam>
  <dsp:oparam name="outputRowStart"><tr></dsp:oparam>
  <dsp:oparam name="outputRowEnd"></tr></dsp:oparam>
```



```
<dsp:oparam name="output">
  <td>
    <dsp:val ueof param="element.absoluteName">&nbsp; </dsp:val ueof>
  </td>
</dsp:oparam>
</dsp:droplet>
```

---

## TableRange

Displays a subset of the elements of an array, arranging the output in a two-dimensional format.

<b>Class Name</b>	atg.droplet.TableRange
<b>Component</b>	/atg/dynamo/droplet/TableRange

### Required Input Parameters

#### ***array***

Defines the list of items to output. This parameter can be a Collection (Vector, List, or Set), Iterator, Enumeration, Map, Dictionary, or array.

#### ***howMany***

The number of array elements to display. If howMany is greater than the number of array elements, TableRange renders the number of elements available.

#### ***numColumns***

The number of columns used to display the array. If you specify more columns than elements (numColumns is greater than howMany), the additional columns display empty. In the reverse situation, (numColumns is fewer than howMany), the excess elements display in a second row.

### Optional Input Parameters

#### ***start***

The initial array element, where a setting of 1 (the default) specifies the array's first element. If start is greater than the number of elements in the array, the empty parameter (if specified) is rendered.

#### ***sortProperties***

Holds a string that specifies the order to render array items. The syntax of this parameter depends on the array item type: JavaBean, Dynamic Bean, Date, Number, or String.



To sort on the properties of a JavaBean, specify the value of `sortProperties` as a comma-separated list of property names. You can sort on an unlimited number of properties, where the first property name specifies the primary sort, the second property name specifies the secondary sort, and so on. To specify ascending sort order, prepend the property name with a plus + sign; to specify descending order, prepend with a minus – sign.

For example, the following input parameter specifies to sort an array of JavaBeans alphabetically by title, then in descending order of size:

```
<dsp: param name="sortProperties" value="+title, -size" />
```

If the array contains Dates, Numbers, or Strings, prepend the `sortProperties` value with a plus + or minus – sign to specify ascending or descending sort order.

The following example sorts an output array of Strings in alphabetical order:

```
<dsp: param name="sortProperties" value="+" />
```

In order to sort Map elements by key, set the value as follows:

```
{+|-}_key
```

For example:

```
<dsp: param name="sortProperties" value="-_key" />
```

A nested servlet bean inherits the parent servlet bean's `sortProperties` setting, unless the nested servlet bean has its own `sortProperties` setting. For example, the following setting negates any parent `sortProperties` setting:

```
<dsp: param name="sortProperties" value="" />
```

## Output Parameters

### *index*

Set to the zero-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `index` for the first iteration is 0.

### *count*

Set to the one-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `count` for the first iteration is 1.

### *key*

If the array parameter is a Map or Dictionary, set to the Map or Dictionary key.

### *rowIndex*

Set to the current row index each time an output parameter is rendered.

***columnIndex***

Set to the current column index each time an output parameter is rendered.

***element***

Set to the current array element each time the index increments and the output parameter is rendered.

***hasPrev***

Set to true before any output parameters are rendered, indicates whether the array includes any items that precede the current array set.

***prevStart***

Set before any output parameters are rendered, and only if hasPrev is true, indicates the value of start that should be used to display the previous array set. You can use this parameter to create a link or form submission that displays the previous elements of the array.

***prevEnd***

Set before any output parameters are rendered, and only if hasPrev is true, indicates the (one-based) count of the last element in the previous array set.

***prevHowMany***

Set before any output parameters are rendered, and only if hasPrev is true, indicates the number of elements in the previous array set.

***hasNext***

Set to true before any output parameters are rendered, indicates whether the array includes any items that follow the current array set.

***nextStart***

Set before any output parameters are rendered, and only if hasNext is true, indicates the value of start that should be used to display the next array set.

***nextEnd***

Set before any output parameters are rendered, and only if hasNext is true, indicates the (one-based) count of the last element in the next array set.

***nextHowMany***

Set before any output parameters are rendered, and only if hasNext is true, indicates the number of elements in the next array set.



## Open Parameters

### ***output***

Rendered once for each element in the subset of the array defined by the `start` and `howMany` parameters.

### ***outputStart***

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

### ***outputEnd***

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

### ***outputRowStart***

If the array is not empty, rendered once for each row in the output, at the beginning of the row. For example, this parameter can be used to render the row heading of a table.

### ***outputRowEnd***

If the array is not empty, rendered once for each row in the output, at the end of the row. For example, this parameter can be used to render text following a row in a table.

### ***empty***

Rendered if the array or the specified subset of the array contains no elements.

## Usage Notes

`TableRange` combines the functions of the [Range](#) and [TableForEach](#) servlet beans. Like [TableForEach](#), `TableRange` renders its output parameter in a two-dimensional format, such as a table. Like [Range](#), `TableRange` renders a subset of the output array. The array parameter can be a `Vector`, `Enumeration`, `Dictionary`, or array.

`TableRange` renders its output parameter for each element in its array parameter, beginning with the array element that corresponds to the `start` parameter and continuing until it has rendered a number of elements equal to the `howMany` parameter. When the `howMany` and `numColumns` parameters are equal, the table displays one element in each table column.

## Example

The following example displays the values of the `initialServices` property of ATG's `Initial` component. The values are displayed in a two-column table, 2 at a time. This example uses a [Switch](#) servlet bean to create navigational links after each 2 values. If there are values in the array that precede the ones currently displayed, there is a link to the previous values. If there are values in the array that follow the ones currently displayed, there is a link to the next set of values.



```
<dsp:importbean bean="/atg/dynamo/droplet/TableRange"/>
<dsp:importbean bean="/atg/dynamo/droplet/Switch"/>

<dsp:droplet name="TableRange">
  <dsp:param bean="/Initial.InitialServices" name="array"/>
  <dsp:param name="numColumns" value="2"/>
  <dsp:param name="howMany" value="2"/>
  <dsp:oparam name="outputStart">
    <table border=1></dsp:oparam>
    <dsp:oparam name="outputEnd">
      <tr><td colspan=2>
        <dsp:droplet name="Switch">
          <dsp:param name="value" param="hasPrev"/>
          <dsp:oparam name="true">
            <dsp:getvalueof var="a27" bean="/OriginatingRequest.pathInfo"
              vartype="java.lang.String">
              <dsp:a href="{a27}">
                <dsp:param name="start" param="prevStart"/>
                Previous <dsp:valueof param="prevHowMany"/>
              </dsp:a>
            </dsp:getvalueof>
          </dsp:oparam>
        </dsp:droplet>
        &nbsp;
        <dsp:droplet name="Switch">
          <dsp:param name="value" param="hasNext"/>
          <dsp:oparam name="true">
            <dsp:getvalueof var="a46" bean="/OriginatingRequest.pathInfo"
              vartype="java.lang.String">
              <dsp:a href="{a46}">
                <dsp:param name="start" param="nextStart"/>
                Next <dsp:valueof param="nextHowMany"/>
              </dsp:a>
            </dsp:getvalueof>
          </dsp:oparam>
        </dsp:droplet>
      </table>
    </dsp:oparam>
    <dsp:oparam name="outputRowStart"><tr></dsp:oparam>
    <dsp:oparam name="outputRowEnd"></tr></dsp:oparam>
    <dsp:oparam name="output">
      <td>
        <dsp:valueof param="element.absoluteName">&nbsp;</dsp:valueof>
      </td>
    </dsp:oparam>
  </dsp:droplet>
```



## TargetingArray

Performs a targeting operation and passes the results to another servlet bean for display.

<b>Class Name</b>	atg.targeting.TargetingArray
<b>Component</b>	/atg/targeting/TargetingArray

### Required Input Parameters

#### *targeter*

The targeter service to perform the targeting operation; must be an instance of a class that implements the Targeter interface, such as atg.targeting.DynamicContentTargeter.

### Optional Input Parameters

#### *filter*

A filter service that filters the items returned by the targeter before they are output; must be an instance of a class that implements the ItemFilter interface, such as atg.targeting.conflict.TopichistoryConflictFilter.

#### *sourceMap*

Represents a mapping between names of source objects associated with a targeting operation and their values; must be an instance of atg.targeting.TargetingSourceMap. If this parameter is omitted, the source Map specified by the servlet bean's defaultSourceMap property is used. The default value is /atg/targeting/TargetingSourceMap.

#### *sortProperties*

The display order of result set elements. See [Sorting Results](#) in the [Serving Targeted Content with ATG Servlet Beans](#) chapter on information about specifying this parameter, and the relationship between targeter sorting and servlet bean sorting.

### Output Parameters

#### *elements*

Set to the results of the targeting operation, an array of repository items.

## Open Parameters

### *output*

The parameter to render. Nest a formatting servlet bean such as the [ForEach](#) servlet bean in this parameter in order to display the output of the targeting operation.

### *empty*

Rendered if the targeting operation returns no matching items.

## Usage Notes

TargetingArray performs a targeting operation with the help of its `targeter` and `sourceMap` parameters, and sets the `outputElements` parameter to the resulting array of target objects. TargetingArray is the most general form of targeting servlet bean. It returns all content items that satisfy the targeter's rules, but does not format the output. Use this servlet bean with another servlet bean such as [ForEach](#) to specify formatting for each of the content items returned by the targeting operation. TargetingArray's output parameter contains the inner formatting servlet bean.

## Example

The following example displays all ski resorts returned from the targeting result set. It uses [ForEach](#) to display the output. For each resort, the JSP containing this servlet bean displays the name (in a hypertext link) and an image. This example is identical in output to the [TargetingForEach](#) example; you are more likely to use the TargetingArray servlet bean with your own customized servlet bean instead of a basic servlet bean such as [ForEach](#).

```
<!-- outer ATG Servlet Bean obtains the targeted content --%>
<dsp:droplet name="/atg/targeting/TargetingArray">
  <dsp:param name="targeter"
    bean="/atg/registry/RepositoryTargeters/TargetedContent/SkiResortTargeter"/>
  <dsp:oparam name="output">
    <!-- inner ATG Servlet Bean displays the results --%>
    <dsp:droplet name="/atg/dynamo/droplet/ForEach">
      <dsp:param name="array" param="elements"/>
      <dsp:param name="sortProperties" value="+displayName"/>
      <dsp:oparam name="output">
        <dsp:getvalueof var="URL" param="element.URL"
          vartype="java.lang.String"/>
        <dsp:a href="{URL}">
          <dsp:valueof param="element.displayName"/>
        </dsp:a>
        <dsp:getvalueof var="image" param="element.image"
          vartype="java.lang.String"/>
        
        <p>
      </dsp:oparam>
    </dsp:droplet>
```





```
</dsp:oparam>  
</dsp:droplet>
```

---

## TargetingFirst

Performs a targeting operation and displays the first specified number of items returned by the targeter.

<b>Class Name</b>	atg.targeting.TargetingFirst
<b>Component</b>	/atg/targeting/TargetingFirst

### Required Input Parameters

#### ***targeter***

The targeter service that performs the targeting operation; must be an instance of a class that implements the Targeter interface, such as atg.targeting.DynamicContentTargeter.

### Optional Input Parameters

#### ***filter***

A filter service that filters the items returned by the targeter before they are output; must be an instance of a class that implements the ItemFilter interface, such as atg.targeting.conflict.TopichistoryConflictFilter.

#### ***sourceMap***

Represents a mapping between names of source objects associated with a targeting operation and their values; must be an instance of atg.targeting.TargetingSourceMap. If this parameter is omitted, the source Map specified by the servlet bean's defaultSourceMap property is used. (By default, the value of this property is /atg/targeting/TargetingSourceMap.)

#### ***howMany***

The number of items to display. If howMany is greater than the number of items returned by the targeting operation, TargetingFirst renders the number of items available. If you omit this parameter, only the first item is displayed.

#### ***sortProperties***

The display order of result set elements. See [Sorting Results](#) in the [Serving Targeted Content with ATG Servlet Beans](#) chapter for information about how to specify this parameter, and for information about the relationship between targeter sorting and servlet bean sorting.

***fireViewItemEvent***

Specifies whether to trigger a JMS view item event when a page is accessed. The default is true. To prevent the ATG platform from firing a view item event, set this parameter to false.

***fireContentEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content viewed event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Use the `fireViewItemEvent` parameter instead of the `fireContentEvent` parameter.

***fireContentTypeEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content type event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Use the `fireViewItemEvent` parameter instead of this parameter.

**Output Parameters*****index***

Set to the zero-based index of the current target item each time the output parameter is rendered. The value for the first loop iteration is 0.

***count***

Set to the index plus one. The value for the first loop iteration is 1.

***element***

Each time the output parameter is rendered, set to the next item taken from the array returned by the targeting operation.

***empty***

Rendered if the targeting operation returns no matching items.

**Open Parameters*****output***

Rendered once for each element in the subset of the array of targeting results, as specified by the `howMany` parameter.

***outputStart***

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

***outputEnd***

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

***empty***

Rendered if the results array contains no elements.

**Usage Notes**

TargetingFirst takes the results of the targeting operation and displays the first *n* items returned, where *n* is the number of items specified by the *howMany* parameter. The *targeter* component used for the targeting operation is set by the *targeter* and *sourceMap* input parameters. On each iteration, the output parameter *element* is set to the next item in the resulting array of target objects. If the targeting rules contain sorting directives, TargetingFirst can return the best *howMany* matches among the resulting target objects.

**Example**

The following example displays the first 10 ski resorts in the targeting result set, because the value of its *howMany* parameter is set to 10. For each resort, the JSP containing this targeting servlet bean displays the name (in a hypertext link) and an image. The *sortProperties* parameter specifies that the resorts should be listed alphabetically by their *displayName* property.

---

```
<%-- displays the first 10 ski resorts in the targeting result set --%>

<dsp: droplet name="/atg/targeting/TargetingFirst">
  <dsp: param=
    "bean: /atg/registry/repositorytargeters/targetedcontent/ski resorttargeter"
    name="targeter" value="\ ">
  <dsp: param name="howMany" value="10">
  <dsp: param name="sortProperties" value="+displayName">
  <dsp: oparam name="output">
    <dsp: getvalueof var="a11" param="element. URL" vartype="java. lang. String">
      <dsp: a href="{a11}">
        <dsp: valueof param="element. displayName"/>
      </dsp: a>
    </dsp: getvalueof>
    <dsp: getvalueof var="img18" param="element. image"
      vartype="java. lang. String">
      <dsp: img src="{img18}">
    </dsp: getvalueof>
  <p>
</dsp: oparam>
</dsp: droplet>
```

---



## TargetingForEach

Performs a targeting operation and displays all items returned by the targeter.

<b>Class Name</b>	atg.targeting.TargetingForEach
<b>Component</b>	/atg/targeting/TargetingForEach

### Required Input Parameters

#### ***targeter***

The targeter service that performs the targeting operation; must be an instance of a class that implements the Targeter interface, such as atg.targeting.DynamicContentTargeter.

### Optional Input Parameters

#### ***filter***

A filter service that filters the items returned by the targeter before they are output; must be an instance of a class that implements the ItemFilter interface, such as atg.targeting.conflict.TopichistoryConflictFilter.

#### ***sourceMap***

Represents a mapping between names of source objects associated with a targeting operation and their values; must be an instance of atg.targeting.TargetingSourceMap. If this parameter is omitted, the source Map specified by the servlet bean's defaultSourceMap property is used. (By default, the value of this property is /atg/targeting/TargetingSourceMap.)

#### ***sortProperties***

The order to display result set elements. See [Sorting Results](#) in the [Serving Targeted Content with ATG Servlet Beans](#) chapter for information about how to specify this parameter, and for information about the relationship between targeter sorting and servlet bean sorting. Because TargetingForEach displays all targeting results, the sort order, if present, overrides the targeter's sort order unless you set the maxNumber property to limit the number of results returned.

#### ***maxNumber***

If the maxNumber property is set, the targeter only returns at most maxNumber items in the targeting result set. Use this property to prevent loading potentially large result sets into memory.

#### ***fireViewItemEvent***

Specifies whether to trigger a JMS view item event when a page is accessed. The default is true. To prevent the ATG platform from firing a view item event, set this parameter to false.

***fireContentEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content viewed event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Use the `fireViewItemEvent` parameter instead of this parameter.

***fireContentTypeEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content type event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Use the `fireViewItemEvent` parameter instead of this parameter.

**Output Parameters*****index***

Set to the index of the current array element returned by the targeter service each time the index increments and the output parameter is rendered.

***count***

Set to the index plus one. The value for the first loop iteration is 1.

***element***

Set to the current array element each time the index increments and the output parameter is rendered.

**Open Parameters*****output***

Rendered once for each element in the array of targeting results.

***outputStart***

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

***outputEnd***

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

***empty***

Rendered if the results array contains no elements.



## Usage Notes

TargetingForEach is a specialized version of the [ForEach](#) servlet bean; it performs a targeting operation using the targeter component set by its `targeter` and `sourceMap` parameters. TargetingForEach displays every item returned by the targeting operation in the manner specified by its `output` parameter.

## Example

The following example displays all ski resorts returned in the targeting result set. For each resort, the JSP containing this servlet bean displays the name (in a hypertext link) and an image.

```
<%-- displays all the ski resorts in the result set --%>
<dsp: droplet name="/atg/targeting/TargetingForEach">
  <dsp: param
    bean="/atg/registry/repositorytargeters/targetedcontent/ski resorttargeter"
    name="targeter"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="a7" param="element.URL" vartype="java.lang.String">
      <dsp: a href="{a7}">
        <dsp: valueof param="element.displayName"/>
      </dsp: a>
    </dsp: getvalueof>
    <dsp: getvalueof var="img14" param="element.image"
      vartype="java.lang.String">
      <dsp: img src="{img14}"/>
    </dsp: getvalueof>
  </p>
</dsp: oparam>
</dsp: droplet>
```

## TargetingRandom

Performs a targeting operation and randomly selects and displays *n* items returned by the targeter, where *n* is a number you specify.

<b>Class Name</b>	atg.targeting.TargetingRandom
<b>Component</b>	/atg/targeting/TargetingRandom



## Required Input Parameters

### ***targeter***

The targeter service that performs the targeting operation; an instance of a class that implements the Targeter interface, such as `atg.targeting.DynamicContentTargeter`.

## Optional Input Parameters

### ***filter***

A filter service that filters the items returned by the targeter before they are output; an instance of a class that implements the `ItemFilter` interface, such as `atg.targeting.conflict.TopichistoryConflictFilter`.

### ***sourceMap***

A mapping between names of source objects associated with a targeting operation and their values; must be an instance of `atg.targeting.TargetingSourceMap`. If omitted, the source Map specified by the servlet bean's defaultSourceMap property is used. The default value of this property is `/atg/targeting/TargetingSourceMap`.

### ***howMany***

The number of target items to display. If `howMany` is greater than the number of items returned by the targeting operation, TargetingRandom renders the number of items available. If you omit this parameter, only one item is displayed.

### ***fireViewItemEvent***

Specifies whether to trigger a JMS view item event when a page is accessed. The default is true. To prevent the ATG platform from firing a view item event, set this parameter to false.

### ***fireContentEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content viewed event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, you should use the `fireViewItemEvent` parameter.

### ***fireContentTypeEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content type event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, use the `fireViewItemEvent` parameter.



## Output Parameters

### *index*

Set to the current index each time the output parameter is rendered. The value for the first loop iteration is 0.

### *count*

Set to the index plus one. The value for the first loop iteration is 1.

### *element*

Each time the output parameter is rendered, set to an item randomly picked from the array returned by the targeting operation.

## Open Parameters

### *output*

Rendered once for each element randomly selected from the array of targeting results.

### *outputStart*

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

### *outputEnd*

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

### *empty*

Rendered if the results array contains no elements.

## Usage Notes

TargetingRandom takes the results of the targeting operation and displays a random selection of the items returned. The number of items displayed is set by the `howMany` parameter (the default is 1). The targeter component used for the targeting operation is set by the `targeter` and `sourceMap` input parameters. On each iteration, the `element` parameter is set to an item randomly selected from the resulting array of target objects.

## Example

The following example displays 10 ski resorts randomly selected from the targeting result set. For each resort, the JSP containing this servlet bean displays the name (in a hypertext link) and an image.





---

```
<%-- displays 10 ski resorts randomly selected from the result set --%>

<dsp: droplet name="/atg/targeting/TargetingRandom">
  <dsp: param
    bean="/atg/registry/repositorytargeters/targetedcontent/ski resorttargeter"
    name="targeter"/>
  <dsp: param name="howMany" value="10"/>
  <dsp: oparam name="output">
    <dsp: getvalueof var="a9" param="element.URL" vartype="java.lang.String">
      <dsp: a href="{a9}">
        <dsp: valueof param="element.displayName"/>
      </dsp: a>
    </dsp: getvalueof>
    <dsp: getvalueof var="img16" param="element.image"
      vartype="java.lang.String">
      <dsp: img src="{img16}" />
    </dsp: getvalueof>
  <p>
</dsp: oparam>
</dsp: droplet>
```

---

## TargetingRange

Performs a targeting operation and displays a subset of the items returned by the targeter, specified as a range.

<b>Class Name</b>	atg.targeting.TargetingRange
<b>Component</b>	/atg/targeting/TargetingRange

### Required Input Parameters

#### *targeter*

The targeter service that performs the targeting operation; must be an instance of a class that implements the Targeter interface, such as atg.targeting.DynamicContentTargeter.



## Optional Input Parameters

### ***filter***

A filter service that filters the items returned by the targeter before they are output; must be an instance of a class that implements the `ItemFilter` interface, such as `atg.targeting.config.TopichistoryConfigFilter`.

### ***sourceMap***

Represents a mapping between names of source objects associated with a targeting operation and their values; must be an instance of `atg.targeting.TargetingSourceMap`. If this parameter is omitted, the source Map specified by the servlet bean's default `sourceMap` property is used. (By default, the value of this property is `/atg/targeting/TargetingSourceMap`.)

### ***howMany***

The number of array elements to display. If `howMany` is greater than the number of array elements, TargetingRange renders the number of elements available.

### ***start***

The initial array element, where a setting of 1 (the default) specifies the array's first element. If `start` is greater than the number of elements in the array, the empty parameter (if specified) is rendered.

### ***sortProperties***

The order to display result set elements. See [Sorting Results](#) in the [Serving Targeted Content with ATG Servlet Beans](#) chapter for information about how to specify this parameter, and for information about the relationship between targeter sorting and servlet bean sorting.

### ***fireViewItemEvent***

Specifies whether to trigger a JMS view item event when a page is accessed. The default is true. To prevent the ATG platform from firing a view item event, set this parameter to false.

### ***fireContentEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content viewed event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, use the `fireViewItemEvent` parameter.

### ***fireContentTypeEvent***

Specifies whether to trigger an event when a page is accessed. If set to true or omitted, a JMS view event fires; if the accessed page is a Content repository item, a content type event also fires. To prevent the ATG platform from firing these two events, set this parameter to false.

**Note:** Instead of this parameter, use the `fireViewItemEvent` parameter.



## Output Parameters

### *index*

Set to the zero-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `index` for the first iteration is 0.

### *count*

Set to the one-based index of the current array element each time the output parameter is rendered. For example, if `start` is set to 1, the value of `count` for the first iteration is 1.

### *element*

Set to the current array element each time the `index` increments and the output parameter is rendered.

### *hasPrev*

Set to true before any output parameters are rendered, indicates whether there are any array items before the current array set.

### *prevStart*

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the value of `start` that should be used to display the previous array set.

### *prevEnd*

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the (one-based) count of the last element in the previous array set.

### *prevHowMany*

Set before any output parameters are rendered, and only if `hasPrev` is true, indicates the number of elements in the previous array set.

### *hasNext*

Set to true before any output parameters are rendered, indicates whether there are any array items after the current array set.

### *nextStart*

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the value of `start` that should be used to display the next array set.

### *nextEnd*

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the (one-based) count of the last element in the next array set.

***nextHowMany***

Set before any output parameters are rendered, and only if `hasNext` is true, indicates the number of elements in the next array set.

**Open Parameters*****output***

Rendered once for each element in the subset of the results array defined by the `start` and `howMany` parameters.

***outputStart***

If the array is not empty, rendered before any output elements. For example, this parameter can be used to render a heading.

***outputEnd***

If the array is not empty, rendered after all output elements. For example, this parameter can be used to render a footer.

***empty***

Rendered if the array or the specified subset of the array contains no elements.

**Usage Notes**

`TargetingRange` is a specialized version of the `Range` servlet bean. `TargetingRange` takes the results of the targeting operation and displays a selection of the items returned, beginning with the item whose count is the `start` parameter. The number of items displayed is set by the `howMany` parameter. The targeter component used for the targeting operation is set by the `targeter` and `sourceMap` input parameters.

`TargetingRange` renders its output parameter for each element in the resulting collection of target objects, starting at the count (inclusive) specified by the `start` input parameter, and ending when it has satisfied the `howMany` input parameter. For example, if `start` = 1 and `howMany` = 10, the first 10 elements of the result set are displayed; if `start` = 11, the next 10 elements are displayed

**Example**

The following example displays 10 ski resorts, selected in the range from items 11 through 20 in the targeting result set. If there are fewer than 20 items, the example displays items 11 through last. For each resort, the JSP containing this servlet bean displays the name (in a hypertext link) and an image.

---

```
<%-- displays ski resorts 11-20 (1-indexed) of the result set--%>
<dsp:droplet name="/atg/targeting/TargetingRange">
  <dsp:param
bean="/atg/registry/repositorytargeters/targetedcontent/ski resorttargeter"
name="targeter"/>
```



```

<dsp: param name="start" value="11"/>
<dsp: param name="howMany" value="10"/>
<dsp: oparam name="output">
  <dsp: getvalueof var="a11" param="element.URL"
    vartype="java.lang.String">
    <dsp: a href="{a11}">
      <dsp: valueof param="element.displayName"/>
    </dsp: a>
  </dsp: getvalueof>
  <dsp: getvalueof var="img18" param="element.image"
    vartype="java.lang.String">
    <dsp: img src="{img18}" />
  </dsp: getvalueof>
</p>
</dsp: oparam>
</dsp: droplet>

```

## TargetPrincipalsDroplet

Locates all organizations where a user has the specified role.

<b>Class Name</b>	atg.userdirectory.droplet.TargetPrincipalsDroplet
<b>Component</b>	/atg/userdirectory/droplet/TargetPrincipals

### Required Input Parameters

#### *userId*

The ID of the user to be queried.

#### *roleName*

Supplies a relative role that the user is assigned.

### Output Parameters

#### *principals*

Holds the organizations returned from the query in a `java.util.Collection` object.



## Open Parameters

### *empty*

Rendered when no items are returned from the query.

### *error*

Rendered when an error occurs and the `TargetPrincipal.s` servlet bean is unable to perform the query.

### *output*

Rendered when organizations are returned from the query.

## Usage Notes

`TargetPrincipals` locates all organizations where a specific user has a relative role. Based on the search results, additional content is rendered by the open parameters you specify.

Be sure to set the `userDirectory` property to the relevant User Directory component.

## Example

In this example, `TargetPrincipals` searches the User Directory for all organizations where the active user holds the role of `admin`. If no organizations are returned or an error occurs, text displays reporting this circumstance. Otherwise, a collection of organizations displays.

---

```
<dsp: droplet name="/atg/userdirectory/droplet/TargetPrincipal.s">
  <dsp: param name="userId" beanvalue="/atg/userprofiling/Profile.id"/>
  <dsp: param name="roleName" value="admin"/>

  <dsp: oparam name="empty">
    The user does not have that role in any organization.
  </dsp: oparam>

  <dsp: oparam name="error">
    Cannot perform query. Try again later.
  </dsp: oparam>

  <dsp: oparam name="output">
    <dsp: droplet name="/atg/dynamo/droplet/ForEach">
      <dsp: param name="array" param="principals"/>

      <dsp: oparam name="output">
        <dsp: valueof param="element.name"/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>
</dsp: droplet>
```

---



# TransactionDroplet

Marks the bounds of a transaction within a JSP.

<b>Class Name</b>	atg.dtm.TransactionDroplet
<b>Component</b>	/atg/dynamo/transaction/droplet/Transaction

## Required Input Parameters

None

## Optional Input Parameters

### *transAttribute*

Specifies how to handle the current transaction and whether to create a transaction. Its value must be one of the following:

- **requiresNew** (default): The current transaction, if any, is suspended then resumed at the end of the droplet. A transaction is created before calling the output open parameter, then committed at the end of the servlet bean.
- **notSupported**: The current transaction, if any, is suspended, then resumed at the end of the servlet bean. The output open parameter is executed without any transaction context.
- **supports**: The current transaction, if any, is used. Otherwise, no transaction is used.
- **required**: The current transaction, if any, is used. If there is no current transaction, then one is created before calling the output open parameter, then committed at the end of the servlet bean.
- **mandatory**: If no transaction is in place, the output open parameter is not executed and the `errorOutput` is executed instead with an error message. If a transaction is in place, this setting behaves the same as **supports**.
- **never**: If a transaction is in place, the output open parameter is not executed and the `errorOutput` is executed instead with an error message. If no transaction is in place, this setting behaves the same as **supports**.

## Open Parameters

### *output*

Executed in the transaction context defined by the `transAttribute` parameter; if an earlier transaction error occurs, the servlet bean does not execute this parameter.

***errorOutput***

Executed if a transaction error occurs before or after the output open parameter. Within the open parameter, the error is communicated through the following parameters:

- `errorMessage`: a text message indicating the error
- `errorStackTrace`: the full stack trace of the error

***successOutput***

Executed after the output parameter if the commit or rollback operation completes successfully.

**Usage Notes**

TransactionDroplet can be used to enclose an area of a page within its own transaction context behavior. Several transactional behaviors are available—the same set that are available for Enterprise JavaBeans.

The open parameter output is rendered within the context of the specified transactional behavior. If a transaction error occurs before the servlet bean can execute the output parameter, then output is not rendered.

If any transaction error occurs, the `errorOutput` open parameter is rendered; otherwise the `successOutput` open parameter is rendered after execution of the output open parameter.

**Example**

The following example shows execution of a demarcated portion of a page in its own transaction, as specified by the `requi resNew` directive.

---

```
<dsp: dropl et name="/atg/dynamo/transacti on/dropl et/Transacti on">
  <dsp: param name="transAttri bute" val ue="requi resNew"/>
  <dsp: oparam name="output">

    ... portion of page executed in demarcated area ...

  </dsp: oparam>
</dsp: dropl et>
```

---

## UserListDroplet

Searches for all users assigned to an organization.





<b>Class Name</b>	atg.userdirectory.droplet.UserListDroplet
<b>Component</b>	atg/userdirectory/droplet/UserList

## Required Input Parameters

### *organizationId*

The organization that contains the users to locate.

## Optional Input Parameters

### *excludedId*

Holds a user ID string that should be removed from the result set.

## Output Parameters

### *users*

Contains users of the specified organization in a `java.util.Collection`.

## Open Parameters

### *empty*

Rendered when no items are returned from the query.

### *error*

Rendered when an invalid organization ID or no Organizational ID is provided.

### *output*

Rendered when items are returned from the query.

## Usage Notes

UserListDroplet returns all users assigned to a particular organization. The input parameter `excludedId` lets you exclude a user from the search. Based on the search results, additional content is rendered by the specified open parameters.

Be sure to set the `userDirectory` property to the relevant User Directory component.



## Example

The following example shows how `UserListDroplet` locates all users assigned to the `org212` organization, skipping user `80314`. Descriptive text is provided when no items are returned or an error occurs. Users that are found for the organization are displayed in a list.

```
<dsp: droplet name="/atg/userdirectory/droplet/UserList">
  <dsp: param name="organization" value="org212"/>
  <dsp: param name="excludedId" value="80314"/>

  <dsp: oparam name="empty">
    That organization doesn't have any users assigned to it.
  </dsp: oparam>

  <dsp: oparam name="error">
    Error: try your search again. Make sure you are using a valid organizational
    ID.
  </dsp: oparam>

  <dsp: oparam name="output">
    <dsp: droplet name="/atg/dynamo/droplet/ForEach">
      <dsp: param name="array" param="users"/>

      <dsp: oparam name="output">
        The following users are part of the organization:
        <dsp: valueof param="element.lastName"/>,
        <dsp: valueof param="element.firstName"/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>
</dsp: droplet>
```

## ViewPrincipalsDroplet

Obtains a user's roles or organizations.

<b>Class Name</b>	<code>atg.userdirectory.droplet.ViewPrincipalsDroplet</code>
<b>Component</b>	<code>/atg/userdirectory/droplet/ViewPrincipals</code>



## Required Input Parameters

### *userId*

The user whose principals are returned.

## Optional Input Parameters

### *principalType*

The type of principals to return. Options include *organization* and *role*. When this parameter is omitted, the default, *organization*, is used.

## Output Parameters

### *principals*

A `java.util.Collection` object that holds the organizations returned from the query.

## Open Parameters

### *empty*

Rendered when no items are returned from the query.

### *error*

Rendered when an invalid user ID or no user ID is provided.

### *output*

Rendered when items are returned from the query.

## Usage Notes

`ViewPrincipalsDroplet` lets you locate a user's roles or organizations. Based on the search results, additional content is rendered by the open parameters you specify.

Be sure to set the `userDirectory` property to the relevant User Directory component.

## Example

In this example, `ViewPrincipals` locates all organizations of the current user and displays them.

---

```
<dsp: droplet name="/atg/userdirectory/droplet/ViewPrincipals">
  <dsp: param name="userId" bean="Profile.id"/>
  <dsp: param name="principalType" value="organization"/>

  <dsp: oparam name="output">
```



```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param name="array" param="principals"/>

  <dsp: oparam name="output">
    You are part of the following organizations:
    <dsp: valueof param="element.name"/>
  </dsp: oparam>

</dsp: droplet>
</dsp: oparam>
</dsp: droplet>
```

---

## WorkflowInstanceQueryDroplet

Returns information about a specified set of workflow instances.

<b>Class Name</b>	atg.workflow.servlet.WorkflowInstanceQueryDroplet
<b>Component</b>	None provided; see Usage Notes.

### Required Input Parameters

None

### Optional Input Parameters

#### ***subjectId***

Repository ID of the workflow subject whose instances should be returned. If omitted, instances for all workflow subjects are returned.

#### ***processName***

Name of the workflow process whose instances should be returned. If omitted, instances for all workflow processes are returned.

#### ***segmentName***

Name of the workflow process segment whose instances should be returned. This parameter takes effect only if the processName parameter is set. If omitted, instances for all segments of the specified workflow process are returned.

***taskElementId***

Process element ID of the workflow task where the returned instances must be waiting—that is, in order for the query to return an instance, the specified task must be active for that instance. This parameter takes effect only if both the `processName` and `segmentName` parameters are set. If omitted and `taskElementIds`, `taskName`, and `taskNames` are also omitted, all instances for the specified workflow segment are returned.

***taskElementIds***

Process element IDs of the workflow tasks where the returned instances must be waiting—that is, in order for the query to return an instance, at least one of the specified tasks must be active for that instance. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId` is not set. If omitted and `taskName` and `taskNames` are also omitted, all instances for the specified workflow segment are returned.

***taskName***

Name of the workflow task where the returned instances must be waiting—that is, in order for the query to return an instance, the specified task must be active for that instance. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId` and `taskElementIds` are not set. If omitted and `taskNames` is also omitted, all instances for the specified workflow segment are returned.

***taskNames***

Names of the workflow tasks where the returned instances must be waiting—that is, in order for the query to return an instance, at least one of the specified tasks must be active for that instance. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId`, `taskElementIds`, and `taskName` are not set. If omitted, all instances for the specified workflow segment are returned.

**Output Parameters*****instances***

Set to the results of the instance query, a `Collection` of `ProcessInstanceInfo` objects.

***errorMessage***

Set to the error message if an error occurs in the course of executing the query.

**Open Parameters*****output***

Rendered after the instance query is complete and the results are placed into the `instances` parameter.

***empty***

Rendered if no instances are returned from the query.

**error**

Rendered if an error occurs while executing the query. The `errorMessage` parameter is set to the corresponding error message.

**Usage Notes**

`WorkflowInstanceQueryDroplet` performs a workflow instance query and returns the resulting `Collection` of `atg.process.ProcessInstanceInfo` objects. `WorkflowInstanceQueryDroplet` is typically used in conjunction with another servlet bean, such as [ForEach](#) or [Range](#), which can iterate through the `Collection` to display the `ProcessInstanceInfo` objects. In addition, the information contained in each `ProcessInstanceInfo` (that is, the `subjectId`, `processName`, and `segmentName` properties) can be passed as input parameters to a [WorkflowTaskQueryDroplet](#), in order to obtain the tasks (active or otherwise) corresponding to the workflow instance.

For some types of workflows, the number of returned workflow instances can be very large. It is a good idea to constrain the query as much as possible—for example, to a particular task element.

The Scenarios module does not include a Nucleus component of this class, because each component is typically configured to work with a specific type of workflow. When you create a workflow type, you can create a `WorkflowInstanceQueryDroplet` component and configure it by setting the following properties in the component's properties file:

- `processManager`: Used to find the workflow instances
- `workflowManager`: Used to access workflow information

**Example**

The following example shows how to use a `WorkflowInstanceQueryDroplet` component to display all outstanding workflow instances:

---

```
<dsp: droplet name="WorkflowInstanceQuery">
  <dsp: oparam name="empty">
    No outstanding workflow instances.
  </dsp: oparam>
  <dsp: oparam name="output">
    <dsp: droplet name="/atg/dynamo/droplet/ForEach">
      <dsp: param name="array" param="instances"/>
      <dsp: oparam name="output">
        Workflow subject <dsp: valueof param="element.subjectId"/><br/>
      </dsp: oparam>
    </dsp: droplet>
  </dsp: oparam>
</dsp: droplet>
```

---



## WorkflowTaskQueryDroplet

Returns information about a specified set of workflow tasks.

<b>Class Name</b>	atg.workflow.servlet.WorkflowTaskQueryDroplet
<b>Component</b>	None provided; see Usage Notes.

### Required Input Parameters

None

### Optional Input Parameters

#### ***activeOnly***

If true, the query returns only active tasks; otherwise, it returns all tasks, including inactive and completed tasks. Default is true.

#### ***accessibleOnly***

If true, the query returns only tasks that are accessible to a particular principal. The default is true. If the `principal` parameter is set, accessibility is checked against the specified directory principal; otherwise, accessibility is checked against the current user.

#### ***principal***

The `atg.userdirectory.DirectoryPrincipal` (which can represent a user, role, group, or organization) whose access rights determine the tasks' accessibility. If omitted, accessibility is checked against the current user. This parameter takes effect only if the `accessibleOnly` parameter is set to true.

#### ***accessRight***

Determines the tasks' accessibility, set to one of the following values:

- `write`: represents the ability to change various attributes of a task, such as its priority, owner, and access control list
- `execute`: represents the ability to execute, claim, and release a task

This parameter takes effect only if the `accessibleOnly` parameter is set to true.

For example, if `accessRight` is set to `execute`, only those tasks which can be executed, claimed, and released by the current user or principal are returned.

#### ***ownership***

Specifies how tasks should be filtered by ownership, set to one of the following values:



- any: Accept all tasks regardless of ownership (default).
- self: Accept only tasks owned by the current user or principal.
- unowned: Accept only tasks that are not owned.
- selfOrUnowned: Accept only tasks that are either not owned or owned by the current user or principal.

This parameter takes effect only if the `accessibleOnly` parameter is set to true.

### ***subjectId***

Repository ID of the workflow subject whose tasks should be returned. If omitted, tasks for all workflow subjects are returned. The `subjectId` parameter must be set if the `activeOnly` parameter is set to false; in other words, inactive and completed tasks can be returned relative to a particular workflow subject only.

### ***processName***

Name of the workflow process whose tasks should be returned. If omitted, tasks for all workflow processes are returned.

### ***segmentName***

Name of the workflow process segment whose tasks should be returned. This parameter takes effect only if the `processName` parameter is set. If omitted, tasks for all segments of the specified workflow process are returned.

### ***taskElementId***

Process element ID of the workflow task which should be returned. This parameter takes effect only if both the `processName` and `segmentName` parameters are set. If omitted and `taskElementIds`, `taskName`, and `taskNames` are also omitted, all tasks for the specified workflow segment are returned.

### ***taskElementIds***

Process element IDs of the workflow tasks which should be returned. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId` is not set. If omitted and `taskName` and `taskNames` are also omitted, all tasks for the specified workflow segment are returned.

### ***taskName***

Name of the workflow task which should be returned. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId` and `taskElementIds` are not set. If omitted and `taskNames` is also omitted, all tasks for the specified workflow segment are returned.

### ***taskNames***

Names of the workflow tasks which should be returned. This parameter takes effect only if both the `processName` and `segmentName` parameters are set, and if `taskElementId`, `taskElementIds`, and `taskName` are not set. If omitted, all tasks for the specified workflow segment are returned.



***sorter***

The Comparator used to sort the tasks returned by the query. If omitted, the tasks are returned sorted first by the workflow process and segment names and subject IDs, then by their order of appearance in the workflow definition.

**Output Parameters*****tasks***

Set to the results of the task query, a Collection of TaskInfo objects.

***errorMessage***

Set to the error message if an error occurs in the course of executing the query.

**Open Parameters*****output***

Rendered after the task query has been completed, and the results are placed into the tasks parameter.

***empty***

Rendered if no tasks are returned from the query.

***error***

Rendered if an error occurs while executing the query. The errorMessage parameter is set to the corresponding error message.

**Usage Notes**

WorkflowTaskQueryDroplet performs a workflow task query and returns the resulting Collection of atg.workflow.TaskInfo objects. WorkflowTaskQueryDroplet is typically used in conjunction with another servlet bean, such as ForEach or Range, which can iterate through the Collection to display the TaskInfo objects.

The Scenarios module does not include a Nucleus component of this class, because each component is typically configured to work with a specific type of workflow. When you create a workflow type, you can create a WorkflowTaskQueryDroplet and configure it by setting the following properties in the component's properties file:

- workflowManager: atg.workflow.WorkflowManager used to find the tasks.
- workflowViewPath: Nucleus path of the session-scoped atg.workflow.WorkflowView component used to find the tasks accessible to the current user.
- userDirectoryUserAuthority:  
atg.userdirectory.UserDirectoryUserAuthority used to resolve directory principals.



## Example

The following example shows how to use a WorkflowTaskQueryDroplet component to display all active tasks that are owned by the current user, and which he or she can execute, claim, and release:

```
<dsp:droplet name="WorkflowTaskQuery">
  <dsp:param name="accessRight" value="execute"/>
  <dsp:param name="ownership" value="self"/>
  <dsp:oparam name="empty">
    No tasks to execute.
  </dsp:oparam>
  <dsp:oparam name="output">
    <dsp:droplet name="/atg/dynamo/droplet/ForEach">
      <dsp:param name="array" param="tasks"/>
      <dsp:oparam name="output">
        Workflow subject <dsp:valueof param="element.subjectId"/>:
        task <dsp:valueof param="element.taskDescriptor.name"/><br/>
      </dsp:oparam>
    </dsp:droplet>
  </dsp:oparam>
</dsp:droplet>
```

## XMLToDOM

Parses an XML document and transforms it into a DOM document.

<b>Class Name</b>	atg.droplet.xml.XMLToDOM
<b>Component</b>	/atg/dynamo/droplet/xml/XMLToDOM

## Required Input Parameters

None

## Optional Input Parameters

### *input*

The XML document to parse. This can be either the absolute or relative URL for an XML document. If omitted, the open parameter unset executes.

***validate***

Determines whether the parser should validate this document against the DTD specified in the `input` parameter. Options include:

- `true`: Read the DTD, validate the document, and apply all information from the DTD.
- `false` (default): Read the DTD, do not validate the document, but substitute default attributes, and apply other information from the DTD at the parser's discretion (Apache Xerces supports this mode).
- `noDtd`: Do not read the DTD, validate the document, or apply any defaults or other information. Use this mode when you do not want the parser to try and resolve the DTD location, perhaps because the remote URL location is temporarily unreachable. The resulting document might be incomplete, might have incorrect attribute values, and does not have substitution of external entities. (Apache Xerces supports this mode, but it might not be available for every parser).

If the XML document lacks a DTD, you must omit this parameter.

**Output Parameters*****document***

Set to a DOM document if the XML document is successfully retrieved and parsed.

***errors***

Set to an enumeration of Exceptions if failures occurred when parsing or retrieving the XML document.

**Open Parameters*****unset***

Rendered if the `input` parameter is not set.

***output***

Rendered if the XML document is retrieved and parsed.

***failure***

Rendered when the XML document cannot be parsed or retrieved.

**Usage Notes**

XMLToDOM parses an XML document that is specified by the `input` parameter. The result of the parse is bound to the `document` parameter, which can then be manipulated inside the `output` open parameter.

XMLToDOM does not have any inherent way to output its results. You typically nest within its `output` parameter a servlet bean or code that can handle the resulting Data Object Model (DOM) component.



## Example

See [Processing XML in a JSP](#).

# XMLTransform

Given an XML document and an XSLT or JSP template, transforms and outputs the XML document.

<b>Class Name</b>	atg.droplet.xml.XMLTransform
<b>Component</b>	/atg/dynamo/droplet/xml/XMLTransform

## Required Input Parameters

None

## Optional Input Parameters

### *input*

The XML document to transform, either the URL for an XML document, DOM document object, or SAX `InputSource` or a text `InputStream`. If omitted, the open parameter `unset` executes.

### *validate*

Determines whether the parser should validate this document against the DTD specified in the `input` parameter. Options include:

- `true`: Read the DTD, validate the document, and apply all information from the DTD.
- `false` (default): Read the DTD, do not validate the document, but substitute default attributes, and apply other information from the DTD at the parser's discretion (Apache Xerces supports this mode).
- `noDTD`: Do not read the DTD, validate the document, or apply any defaults or other information. Use this mode when you do not want the parser to try and resolve the DTD location, perhaps because the remote URL location is temporarily unreachable. The resulting document might be incomplete, might have incorrect attribute values, and does not have substitution of external entities. (Apache Xerces supports this mode, but it might not be available for every parser).

If the XML document lacks a DTD, you must omit this parameter.

### *template*

The template to use to transform the XML document. This can be either an XSLT template or a JSP. Both absolute paths and relative paths are accepted.

***passParams***

When the template is an XSL stylesheet, sets a flag to control the passing of parameters from the request and servlet bean environment, to top-level <xsl : param> elements in the XSL stylesheet. Set to one of the following values:

- none: no parameters are passed (default)
- query: URL query parameters from a GET request
- post: form parameters from a POST request
- local : only parameters in this servlet bean scope
- all : parameters from all servlet bean and request scopes

If more than one parameter exists for a given name, only the first is passed. The order of parameter scopes is from local (within the XMLTransform servlet bean), up through other enclosing scopes, and eventually to the request query or post parameters.

***documentParameterName***

The name of the parameter to which the input DOM Document is bound when using a JSP template. The default value is `inputDocument`.

**Output Parameters*****documentOutputName***

Set to the DOM document that results from the transformation. The default is `document`.

***errors***

Set to an enumeration of Exceptions if failures occurred when transforming the XML document.

**Open Parameters*****unset***

Rendered if the `input` parameter is omitted.

***output***

Rendered when the XML document is transformed. If the output open parameter is specified, the document is not automatically output in the JSP.

***failure***

Rendered when the XML document cannot be transformed.



## Usage Notes

XMLTransform lets you transform XML documents with a template. The template can be either an XSLT template or a JSP. The document to transform is specified by the `input` parameter. The template to apply is determined in the following order of precedence:

1. Template input parameter of the XMLTransform servlet bean
2. Mapping specified in the XMLTemplateMap Component
3. Default template specified in the XMLTemplateMap Component
4. XSL stylesheet directive embedded in the XML source document

## Example

See [Applying a Template to an XML Document](#)



# Appendix C: SimpleSQLFormHandler

You can interact directly with an SQL database with a SimpleSQLFormHandler, which is an instance of the class `atg.droplet.sql.SimpleSQLFormHandler`. A subclass of `atg.droplet.GenericFormHandler`, this class supports SQL database table queries, inserts, updates, and deletions. The form handler requires that each row be uniquely identified by a set of key columns; consequently, the form handler can act on only one row at a time.

The SimpleSQLFormHandler lets you interact with a SQL database directly, by specifying the table rows to query or update. This form handler relies only minimally on the repository layer, it requires you to know exactly which items are required.

When you create a SimpleSQLFormHandler, you set the following properties:

- The URL of a JDBC driver
- The name of a table to edit
- Identifiers for the rows and columns to edit

You can embed this component in JSP forms. The form fields are used to set the form handler's `value` property. The `value` property is a Dictionary whose subproperties store values that identify the row that is currently accessed by the form, indexed by column name. The `value` subproperties can be keys used to look up an item or new values for update and insert operations. After lookup operation, these subproperties are set to the returned row.

## SimpleSQLFormHandler Properties

The following properties in a SQL form handler component control mapping between the SQL table and the form:

Property	Function
<code>columns</code>	The names of columns to look up, update, and insert. If this property is not set, all columns in the table are used.
<code>connectionURL</code>	JDBC URL of the database connection to use. Typically, this is the URL of a connection pool in the form:  <code>jdbc:atgpool:pool-name</code>



Property	Function
keyColumns	Names of columns to use to find the row on which to perform lookup, update, and delete operations.
metadataCatalogName	String representing a catalog name. If the active database user does not own the tables accessed by the SQL repository, this property is referenced once during initialization of the logger in a call to determine the column types.
metadataSchemaPattern	String representing a schema name pattern. If the active database user does not own the tables accessed by the SQL Repository, this property is referenced once during initialization of the repository in a call to determine the column types. Set this property to the username of the database owner in the case required by your database for object identifiers.
state	<p>The current state of the SimpleSQLFormHandler component, expressed by one of the following integer/constants:</p> <p>1 / STATE_NOTSET The bean does not represent a valid item in the table. Any attempt to get properties of the value returns only the properties explicitly set.</p> <p>2 / STATE_INVALID The key properties are set, but no record was found for a lookup, update, or delete.</p> <p>3 / STATE_VALID Values for the key properties are set and a query has successfully returned one or more items.</p> <p>4 / STATE_CONFIGURATION_ERROR A database operation failed due to a configuration error: a lookup, update, or delete operation failed because one of the specified key properties is not set; or an update operation failed because no key properties are defined or a value for a specified column property is missing.</p> <p>5 / STATE_DB_ERROR A database error occurred.</p>
tableName	Database table used for this form handler.
tablePrefix	String that can be prepended to the tableName value. If the active database user does not own the table where log entries are saved, this property constructs a qualified table name. This property is not used during the initial metadata query, but if present is prepended to the table name when inserts or updates are made.
valid	Indicates whether or not the last operation succeeded.





Property	Function
value	Dictionary that stores the values of the current row. These values are split into two groups: those in the keyColumns list, and those in the columns list. keyColumns properties are used to look up the row for lookup, update, and delete operations. columns properties specify the columns that are the targets of lookup, update and insert operations.

## Submit Handler Methods

The SimpleSQLFormHandler has the following submit handler methods:

Method	Function
handleLookup	Performs a query that sets the row from key column values.
handleUpdate	Uses key property values to update one or more rows.
handleDelete	Uses key properties to remove one or more rows.
handleInsert	Uses the current value properties to insert one or more new rows.
handleReset	Removes the current subproperties of the value property, clears any exceptions, and sets the state of the form handler to STATE_NOTSET.

## SimpleSQLFormHandler Navigation Properties

After a form operation—lookup, update, delete, or insert—is complete, the action attribute of the `dsp: form` tag specifies the page to display. You might want to specify several different pages, where the user's destination depends on the nature of the form operation and whether or not it succeeds.

A SimpleSQLFormHandler has a set of properties you can use to control navigation after a form operation. These properties specify the URLs where the user is redirected on certain error and success conditions. Each submit handler method except handleReset has corresponding success and failure URL properties:

Handler method	Success/failure URL properties
handleLookup	lookupSuccessURL lookupErrorURL



Handler method	Success/failure URL properties
handleUpdate	updateSuccessURL updateErrorURL
handleInsert	insertSuccessURL insertErrorURL
handleDelete	deleteSuccessURL deleteErrorURL

The `dbErrorURL` property specifies the URL to use if a database error occurs when submitting the form.

The value of each property is a URL relative to the page specified by the form's `action` attribute or a local URL defined absolutely from the root. The default value of each property is null, so you must set these properties explicitly. You can specify the values of these URL properties in the form handler component, or with hidden input tags in the JSP. If these property values are not set, the action-specified page is displayed on form submission.

The following example sets the value of `lookupErrorURL`, redirecting the user to `lookupError.jsp`. If the lookup operation succeeds, and the value of `lookupSuccessURL` is not set, the user is redirected to the action-specified page `index.jsp`:

```
<dsp:form action="index.jsp" method="post">
<dsp:input
  bean="MyFormHandler.lookupErrorURL" type="hidden" value="lookupError.jsp"/>
...
</dsp:form>
```

## SQL Form Handler Example

The following example defines properties for the `SimpleSQLFormHandler` component `SkierHandler`, which tracks information about skiers and is configured as follows:

- The `tableName` and `keyColumns` properties let you look up a particular person in the database.
- The navigation property `DBErrorURL` handles errors.

```
$class=atg.droplet.sql.SimpleSQLFormHandler
$scope=session
```

```
connectionURL^=TableManager.connectionURL
keyColumns=NAME
```



```
tableName=SKI ER  
DBErrorURL=sql Error. j sp
```

---

You can use the `Ski erHandl er` component to look up or change the properties of skiers who are registered with an application. For example, skiers can change the value of their `preferredActi vi ty` property with this portion of the `editPerson. j sp` page:

---

```
<tr>  
  <td align=right><b>Your preferred acti vi ty: </b></td>  
  <td><dsp: select bean="Ski erHandl er. val ue. preferredActi vi ty">  
    <dsp: opti on val ue="Ski i ng">Ski i ng</dsp: opti on>  
    <dsp: opti on val ue="Snowboardi ng">Snowboardi ng</dsp: opti on>  
    <dsp: opti on val ue="X-Country">X-Country</dsp: opti on>  
  </dsp: select>  
</td>  
</tr>
```

---

When a user submits this form, the `preferredActi vi ty` property of the `val ue` property of the `Ski erHandl er` is set to the selected option, and is inserted in the appropriate row of the `SKI ER` database table.





# Appendix D: Managing Nucleus Components

This appendix shows how to manage Nucleus components in the ATG Control Center. The following topics are covered:





- [Viewing Components in Nucleus](#)
- [Creating Nucleus Components](#)
- [Configuring Nucleus Components](#)
- [Starting and Stopping Nucleus Components](#)
- [Rebuilding the Component Index](#)
- [Managing Site Pages](#)

## Viewing Components in Nucleus

To see the components currently registered in Nucleus, click **Pages and Components** in the main ATG Control Center navigation bar. When the Components window opens, two options — **Components By Module** and **Components By Path** — appear in the upper left corner of the navigation bar, allowing you to switch between two component views.

### Module View

Click **Components By Module** to see the components within a particular category and module. A *category* describes a component's particular function, such as caching or logging. ATG Control Center uses the following icons to represent the different types of components within each category:

Icon	Component Type
	ATG Servlet Bean
	Form Handler
	Database Connection Pool
	Profile



Icon	Component Type
	Profile Group
	Repository
	SQL Query
	Sensor
	Targeter
	Browser Type
	Data Listener
	Log Listener
	Pipeline Servlet
	Generic component

A *module* is a higher level grouping of components based on general functionality. The standard module Local contains custom components that you create in the ATG Control Center, or by writing .properties files by hand. If your ATG software includes solutions in addition to the ATG platform, the Components window displays these modules too.

In Components By Module view, the Components window displays three panels:

- Modules
- Categories
- Components

### **Filtering components list**

To see all Nucleus components, select **[all modules]** and **[all categories]**. You can also filter the list of components by selecting a specific module and category. For example, if you select the Dynamo module and Caching category, the Components list contains Dynamo's two standard caching components: FileCache and LRUCache. When you select a component from this list, the information panel at the bottom of the Components window displays the component's full path name, class, scope and description. For example:

---

/atg/dynamo/servlet/pipeline/FileCache  
Class: atg.servlet.filecache.FileCache  
Scope: global  
Description: A cache of file data by file name

---

To get more detailed information about a component, opening it in the Component Editor: double-click on the component or choose File | Open Component. For more information about the Component Editor and configuring Nucleus components, see [Configuring Nucleus Components](#).

## Path View

Click **Components By Path** to see the hierarchy of folders and Nucleus components within the Nucleus framework. The hierarchy has an expandable tree structure, just like an ordinary file system. However, the folders shown here do not necessarily map directly to the directories on the server machine.

In the previous section, you looked at a component called `FileCache` in the Caching category of the Dynamo module. If this component was still selected in the Components By Module view when you switched to the Components By Path view, it will be selected in your path view now. If it's not, you can get to it by opening the folders in its path name: `/atg/dynamo/servlet/pipeline/FileCache`.

As you saw in the Components By Module view, the information panel at the bottom of the window displays a path name, class, scope and description whenever you select a component in the Components By Path view. You can get more detailed information about any component by opening it the Component Editor. Double-click on the component you want to open or select **Open Component** from the File menu. To learn more about how to use the Component Editor and how to configure Nucleus components, see the [Configuring Nucleus Components](#) section of this chapter.

The icons you saw in the [Module View](#) section are used here as well. One difference you may notice, though, is that some components have a red dot next to them like this:



This dot means that the component has been started and is currently running on the live Dynamo module. To learn more, see the [Starting and Stopping Nucleus Components](#) section.

The Components By Path view provides additional options for managing components and folders. Using the menu commands in Components By Path view, you can copy, cut, paste, rename, duplicate, refresh, and delete components and folders.

### *Managing Component Folders in Path View*

To create a folder in ATG's component hierarchy:

1. Click **New Folder** on the toolbar or choose **File | New Folder**.
2. In the New Component Folder dialog, specify where to place the new folder. If you selected a folder in the Component Browser, that folder is highlighted in the dialog box.
3. In the **New Folder Name** field, type the new folder's name.
4. Click **OK**. The ATG Control Center adds the new folder to the location you specified and returns to the Component Browser.

**Note:** If the new folder is not immediately visible in the Component Browser, choose **View | Refresh**.

### *Deleting Components, Documents, and Folders*

To delete a component, document, or folder:



1. Select the item to delete from the Component Browser or the Document Browser.  
**Note:** You cannot delete components that are configured as part of the Dynamo Base configuration layer, which are read-only.
2. Click **Delete** or choose Edit | Delete. In the confirmation dialog, click **Yes**.

## Module and Path View Synchronization

The Components By Module view and Components By Path view are synchronized. If you switch from the Components By Module view to the Components By Path view while you have a component selected, ATG Control Center takes you directly to the component's path location within the Nucleus directory structure.

## Searching for a Component

You can search for a component by its name or its Java class or interface:

1. On the Component Browser toolbar, click **Find Component**.
2. In the Find Component dialog box, choose **Search by component name** and type CurrentDate in the **Search For** field. If you want the search to be case-sensitive, mark the checkbox **Match case**.  
**Note:** You can search for partial names by using the asterisk (\*) or question mark (?) wildcard symbols.
3. You can narrow down a search by specifying a directory in the **Look In** field. To search the entire component hierarchy, leave this field blank. To search all folders, mark the checkbox **Include subfolders**.
4. Click **Find**. Components that match the search criteria appear at the bottom of the dialog. Double-clicking on a component takes you directly to its path location in the Components window.

## Creating Nucleus Components

ATG Control Center includes templates to help you create Nucleus components quickly and easily. There are the standard templates you'll see when you select **[all modules]**:

Standard template	Purpose
Empty Targeter	Create an iterator component that lets you specify the order in which content is rendered to a subset of users.
Generic Component	Create components from existing Nucleus classes or other Java classes.
HTML File Repository	Define an HTML repository by specifying the files it will contain and the attributes that describe them.





Standard template	Purpose
Slot	Set up a slot by letting you determine the repository items it manages and how it manages them.
SQL Form Handler	Create a component for inserting, updating, and deleting objects in a specified SQL table.
SQL Query	Create a component that performs queries against a SQL database.
XML File Repository	Define an XML repository by specifying the files it contains and the attributes that describe them.

**Note:** If your ATG software includes solutions in addition to the ATG platform, you will see several additional templates.

## Configuring Nucleus Components

After you create a component or open an existing one, the ATG Control Center opens a Component Editor window. The editor window is divided into several tabs that let you examine and manage various aspects of the component:

- Properties
- Methods
- Events
- Basics
- Configuration

**Note:** If your ATG software includes solutions in addition to the ATG platform, the Component Editor may include additional tabs for certain components.

The component's name and icon, class, configuration layer, pathname, module, and category appear at the top of the editor window. The menu bar and the toolbar in the upper right corner lets you change the state of the component, save any changes you make to it, refresh the editor window, open the Component Browser and get online Help. The information panel at the bottom of the editor gives you descriptive information about the component's properties and methods.

The following sections show how to use the Component Editor to configure Nucleus components:

- [Editing Property Values](#)
- [Invoking Methods](#)
- [Changing Registered Event Listeners](#)
- [Changing a Component's Scope and Description](#)
- [Viewing Configuration Layers and Properties Files](#)



## Editing Property Values

The Properties tab in the Component Editor displays all of the properties defined for the open component. If the component is “live” (indicated by a red dot), the Properties tab shows two columns of property values: Configured Value and Live Value. (Otherwise, it displays the Configured Value column only.) The configured value is the value specified by the component’s properties file; the live value is the current value, which may be different from the configured value. If you want to view read-only properties, select the **Include read-only properties** checkbox. Read-only properties appear grayed-out to indicate that they are unavailable for editing.

**Note:** Not all component properties appear by default in the Component Editor. Certain “expert level” properties are visible only if you choose Tools > Preferences from the ACC window and select the **Show expert-level information** checkbox in the Edit Preferences dialog box.

To edit component properties:

1. Click in the property value cell you want to edit. Editing options depend on the type of property you select:
  - String values provide a text field for editing. You can type values directly into this field or you can click ... to open a pop-up editing window.
  - Int, long, float and double values provide a number field for editing.
  - Boolean values provide a drop-down list with two options: true.
  - Enumerated values provide a drop-down list of options.
  - Array, hash table, and component values have a ... button that opens a corresponding pop-up editing window.
  - All property types have an @ button that lets you set the property value to another component or component property.
2. When you finish, click **Save** on the editor toolbar. If the component is live, a dialog box appears, asking whether to save the configuration changes to the component’s live state.

**Note:** If you changed a property’s configured value, the change is used to configure the live component only when you restart the ATG platform. If you edit the live value of a property, the change takes place immediately, but it is not retained if you stop the component. If another component refers to the live component, references to the component are not updated when you stop or restart the component; you must restart the ATG platform to register the changes. Also, stopping or restarting a cross-referenced component can leave the ATG platform in an unstable state.

### Setting Property Values to Other Components

You can set property values for the component you are editing by linking them to other components or component properties. This lets you set a property value in one component, and have its value immediately reflected in the linked properties of other components.

To link to another component or component property:



1. In the Properties tab of the Component Editor, click the "@" button that appears in the property value cell you are editing. The Select Link Target dialog box opens, displaying the available components by module or by path. When you click on a component, the Component Properties panel on the right displays all of the properties for that component.
2. Select the component or property you want to link to.
  - To link the property of your active component to a component, click on the name of the component in the Components panel.
  - To link the property of your active component to a property of a component, click on the name of the property in the Component Properties panel.
3. Click **Link**. When you return to the Component Editor, the new property link appears in the appropriate value cell.
4. Click the **Save** button on the editor toolbar.

### **Unlinking Component Properties**

To unlink component properties:

1. In the Properties tab of the Component Editor, click the "@" button that appears next to the property you want to unlink.
2. When the Select a Component or Property dialog box opens, click the **Unlink Property** button. The ATG Control Center removes the link and returns you to the Component Editor.
3. Click **Save** in the editor toolbar.

### **Invoking Methods**

The Methods tab in the Component Editor lists the component's Java methods that do not take arguments. The information panel at the bottom of the Methods tab displays the return type of the method you select.

The Methods tab is available only if the component is currently running. If the component is not running, select **Component > Start Component** in the Component Editor.

You cannot add, remove, or edit methods in the Methods tab, but you can invoke them by clicking the Invoke button to the left of the method name. For instance, you may want to call a method that resets a counter or gathers statistics. The method result, if applicable, appears in a separate dialog box.

**Note:** Do not call methods from this tab unless there is a specific task you want to accomplish. In particular, calling `doStartService` or `doStopService` can cause components to become unstable.

### **Changing Registered Event Listeners**

The Events tab in the Component Editor shows you the types of events generated by the open component and the event listeners registered to receive those events. You cannot add, remove, or edit events in the Events tab, but you can edit the list of registered event listeners. (Note that the `Student_01` component that you've been working with does not generate any events.)



To edit the list of event listeners:

1. Select an event listener from the Registered Event Listeners list and click the “...” button that appears to the right. A dialog box opens, displaying a list of the components that listen for this type of event.
2. Click on the component you want to change, then click on the “@” button that appears to the right. (**Note:** You can also add or remove components from the list using the **Insert Before**, **Insert After**, and **Remove** buttons.)
3. When the Select a Component dialog box opens, select a component of the suggested type and click **OK** to return to the previous dialog box.
4. Click **OK**.
5. In the Component Editor, click the **Save** button on the editor toolbar.

## Changing a Component's Scope and Description

The Basics tab in the Component Editor lets you edit the scope and description of the open component.

To change a component's scope or description:

1. Select the new scope (global, session, or request) from the drop-down list.
2. Type a new description to set the component's \$description property. This description appears in the Component Browser information panel when you select the component.
3. Click the **Save** button in the editor toolbar.

## Viewing Configuration Layers and Properties Files

The ATG platform establishes separate configuration layers so you can make changes to a component's properties without modifying Dynamo's base configuration. In a default ATG installation, there are two configuration layers, Dynamo Base and Local, which represent the configuration directories in the ATG platform's default CONFIGPATH (/confi g/dynamo. j ar: l ocal confi g). By default, when you open a component in ATG Control Center, you open the Local configuration layer. Any changes you make to the component will be preserved in the l ocal confi g directory, even if you install a new version of the ATG platform. (**Note:** If necessary, you can open components in a different layer or change the default layer.)

The Configuration tab in the Component Editor shows you the hierarchy of configuration layers and . proper ti es files that configure the open component. Each configuration layer is shown by name with its path. You cannot add, remove, or edit configuration layers from the Configuration tab, but you can double-click on . proper ti es files to view them in a separate display-only file viewer. For example, if you open /atg/dynamo/Confi gurati on, the ATG platform's main Configuration component, you'll see that there are at least two Confi gurati on. proper ti es files controlling the Dynamo Application Framework: a Dynamo Base configuration file in <ATG10di r>/DAS/confi g/confi g. j ar and a Local configuration file in the <ATG10di r>/home/l ocal confi g directory. If you change a Configuration component property, such as ht tpPort (the port number of the ATG platform's internal HTTP server), your change will be stored in the l ocal confi g directory. The next time you start the ATG platform, Nucleus will take the value of the ht tpPort property from l ocal confi g, the last directory in your configuration path.



**Important:** The ATG Control Center shows you the configuration layers used by the modules you are running. Additional layers may exist within your ATG installation depending on the complexity of your system, such as the directories included in your CONFIGPATH, the number of ATG solutions you have installed and whether you're using the internal Serverina server or an external HTTP server.

### ***Opening a Component in a Different Configuration Layer***

By default, you open and edit components in ATG's Local configuration layer. You can open a component in any layer, but you cannot edit the component's properties if the layer is locked. Locked configuration layers, such as Dynamo Base, are marked with a padlock icon (🔒).

To open a component in a different configuration layer:

1. Open the Components window and select the component you want to edit.
2. Right-click on the component and select **Open Component in Layer** from the pop-up menu. The **Select a Configuration Layer** dialog box opens, listing the name and path of each configuration layer. Check marks identify the layers currently in use.
3. Select the layer you want to open and click **OK**. The component opens in a separate Component Editor window.

### ***Changing the Default Configuration Layer***

By default, you open and edit components in Dynamo's Local configuration layer. You can set another configuration layer to open by default as long as it is not a locked layer. After changing the default configuration layer, you can still select a different configuration layer when you open a component. Any components you create, duplicate, or paste will be placed in the default configuration layer.

**Important:** When you use the ATG Control Center to change the default configuration layer, the new default affects only the client machine on which the change is applied and it remains in effect until you shut down that machine. If you want to make a persistent change that applies to the ATG platform and all client machines, you must manually edit the `<ATG10dir>/home/Local config/CONFIG.properties` file and set the `defaultForUpdates` property to false. See the [ATG Installation and Configuration Guide](#) for more information.

To change the default configuration layer:

1. In the Components window, select **Tools > Set Update Layer**.
2. When the **Set a Default Configuration Layer** dialog box opens, select the configuration layer you want to open by default.
3. Click **OK**.

## **Starting and Stopping Nucleus Components**

"Live" components - those that are currently running - are identified with a red dot icon in the Component Editor and in the path view of the Component Browser.

You can start and stop a component from the Component Editor by:



- Clicking the **Start/Stop** button on the toolbar or
- Selecting Start/Stop Component from the Component menu.

You can start and stop a component from the Components window by:

- Selecting Start Component or Stop Component from the File menu.
- Right-clicking on the component and selecting Start Component or Stop Component from the pop-up menu.

**Important:** Stopping or restarting components referenced by other components may leave the ATG platform in an unstable state.

## Rebuilding the Component Index

ATG Control Center creates and maintains indexes of components. Any components you create, either through the ATG Control Center or by creating `.properties` files by hand, appear in the Local module of the Component Browser once you rebuild the component index or restart the ATG platform. By rebuilding a component index, you free up the ACC to display components at a quicker rate. To rebuild the component index while the ATG platform is running, select **Tools > Rebuild Component Index** in the Components task area.


For sites with large numbers of components, indexing can take substantial time and CPU resources. Once your site is deployed and relatively stable, you may want to limit or eliminate component indexing.

You can selectively exclude portions of the hierarchy from indexing by adding pathname prefixes to the `excludeDirectories` property of the `/atg/devtools/ComponentIndex` component. Including `/"` in this list inhibits all indexing, but still leaves the ATG Control Center able to connect.


The component index is maintained incrementally once built, and is rebuilt completely once a day at 1 AM by default. An index is rebuilt at startup only if it does not exist at all. If you'd like to change the frequency of the index building, modify the `updateSchedule` property of the `atg/devtools/IndexBuilder` component. For information on the format used by `updateSchedule`, see the *Scheduler* section in the *Core ATG Services* chapter of the [ATG Programming Guide](#).

## Managing Site Pages







You can see all JSPs in ATG products through the Pages and Components task area of the ATG Control Center. Open the J2EE Pages document browser to view a listing of all J2EE projects and the applications they contain:

Icon	Item Type	Description
	J2EE project	ATG platform module that holds one or more J2EE applications.



Icon	Item Type	Description
	J2EE application	ATG application that's part of a J2EE project. To use a particular J2EE application, you start up the J2EE project that contains it.

After you select an application, you can see its folders and documents. Each document type has its own icon:

Icon	Document Type	File extensions
	JSP document	. j s p and . j s p f
	HTML document	. html or . htm
	XML document	. xml
	Plain text file	. t x t
	Image file	. g i f, . j p g, or . j p e g
	File of any other type	other

When you select a folder or document, you can see its path relative to the Web application root directory in the information panel at the bottom of the screen.

/test.j s p

Using the toolbar and menus at the top of the screen, you can take care of common file management tasks like copying, pasting, deleting, etc. You can open documents in the Document Editor, and preview them in your Web browser. Double-click on the document you want to edit or click the **Open Document** button. See [Appendix E: ATG Document Editor](#) for more information.

## Creating Document Folders

To create a folder in the document root directory:

1. Click the **New Folder** button on the toolbar.
2. When the Specify New Folder Name dialog box opens, specify where you want to place the new folder. (If you selected a folder in the document browser, that folder will already be highlighted in the dialog box.)
3. Type the name of the new folder and click **OK**.

## Deleting Folders

To delete a folder and its contents:



1. Select the folder you want to delete and click the **Delete** button in the toolbar.
2. When the confirmation dialog box opens, click **Yes**.

## Searching for Documents

If you can't find a particular document, you can search for it by page name. You can also search for documents that reference a particular Nucleus component.

To search for a document:

1. Click the **Find Document** button in the toolbar.
2. When the Find Document dialog box opens, select the search method you want to use. You can search by page name (the default).
3. Specify the document you want to find in the Search For field. If you want your search to be case-sensitive, select the **Match case** checkbox.  
**Note:** You can search for partial names by using the asterisk (\*) or question mark (?) wildcard symbols. For example, a search for new\* would retrieve news.jsp and newFeatures.xml. If you don't include the file's extension, be sure to use a wildcard (index.\*, for example).
4. If you want to find documents that reference a particular component, select the **That references this component** checkbox and click the "... " button to select the component.
5. Type the location you want to search in the Look In field, or click **Browse** to select a directory from the document hierarchy. If you want to search all folders within this directory, make sure the **Include subfolders** checkbox is selected.
6. Click **Find**. The search results appear at the bottom of the Find Document dialog box.  
Double-click a document returned from your search to open it in the Document Editor.

## Creating Documents

To create a document in the ATG Document Editor:

1. Select a folder for the new document and click **New Document** in the toolbar.
2. When the Select Document Template dialog box opens, select the module and template to use. To see all available templates, select **all modules**. Click **OK**.
3. Choose the appropriate template:
  - **Change Password Form (JSP)**: Creates a JSP version of the Change Password form template.
  - **Login (JSP)**: Creates a JSP version of the Login template.
  - **Logout (JSP)**: Creates a JSP version of the Logout template.
  - **Page (JSP)**: Creates a simple JSP with a few basic HTML and DSP tag library tags.
  - **Page (JWML)**: Creates a basic page with some JWML tags.





**Note:** ATG applications might include additional JSP document templates.

4. When the New JavaServer Page dialog box opens, enter the new document's title and click **Finish**. The title is used as the HTML title and also appears as an <H1> heading in the new document.
5. In the Input New Document Name dialog, enter the document's file name and extension and click **OK**. The document name must be different from the folder name in which the file resides.

A separate Document Editor window opens, displaying the new document's path name, structure, and full text. [Appendix E: ATG Document Editor](#) shows how to edit a document with the ATG Document Editor.

### ***Rendering New JSPs***

The first time a JSP is requested, it might take longer to display than you expect. Each time a page is requested, the servlet-version of the page is translated into HTML and rendered. New pages are not yet converted into servlets, so they must first be translated into raw Java code and then compiled into servlets before undergoing the final translation into HTML. You can minimize the delay by precompiling all pages at startup; see your application installation guide for instructions.

You can precompile individual pages at server startup by specifying them in your Web application deployment descriptor. For example, you might specify the JSP MyPage. j sp as follows:

---

```
<servlet>
  <servlet-name>MyPage.jsp</servlet-name>

  <jsp-file>/path/to/jsp/MyPage.jsp</jsp-file>

  <load-on-startup>1</load-on-startup>
</servlet>
```

---

where:

- <servlet-name> identifies the servlet to compile.
- <jsp-file> identifies the path to the servlet.
- <load-on-startup> identifies the order in which to compile this servlet.





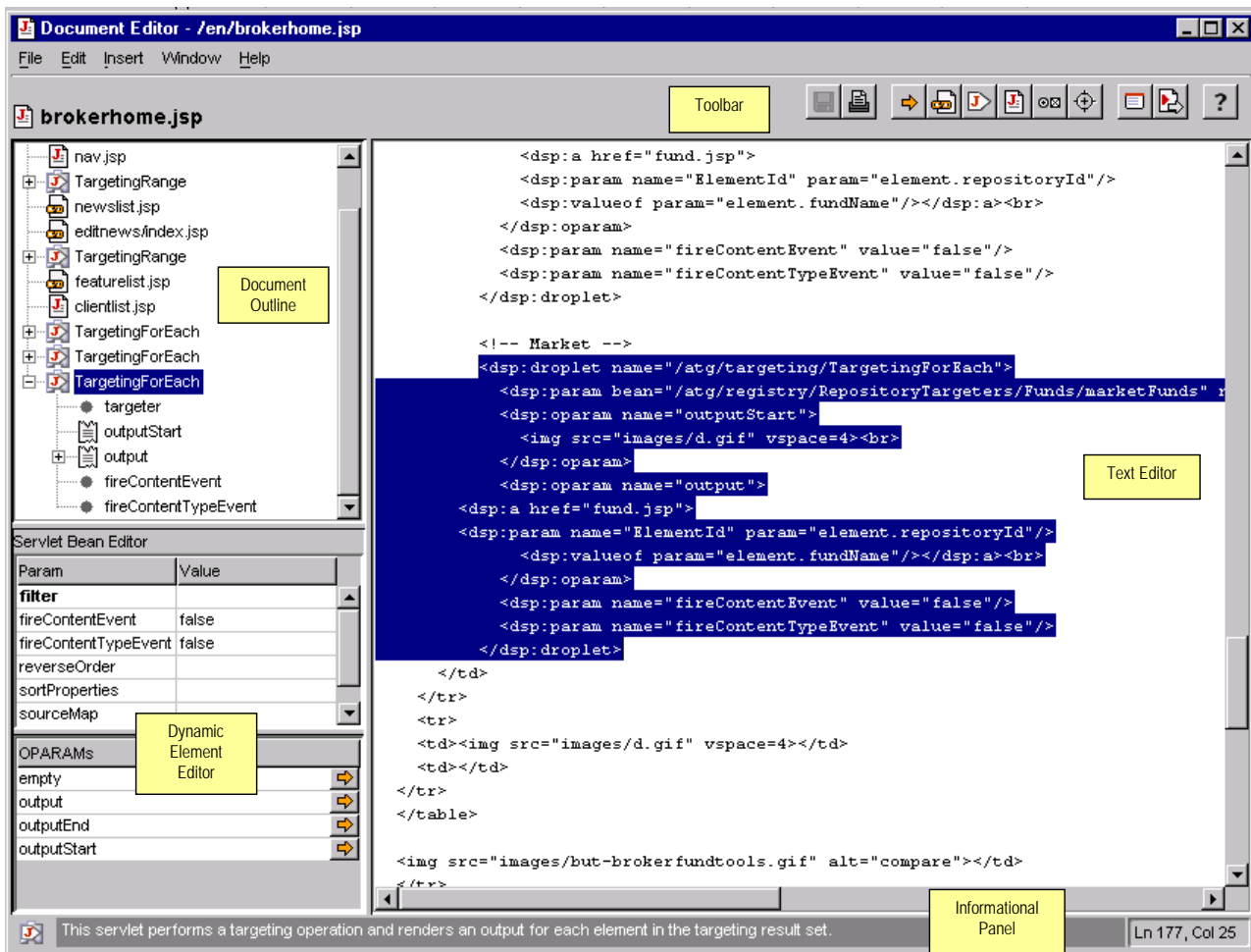
# Appendix E: ATG Document Editor

This appendix shows how to use ATG's proprietary Document Editor to edit JSPs. The following topics are covered::

- [Document Editor Window](#)
- [Creating a JSP](#)
- [Displaying Property Values](#)
- [Embedding Documents](#)
- [Inserting ATG Servlet Beans](#)
- [Inserting a Targeting Servlet Bean](#)
- [Creating Profile Form Pages](#)

## Document Editor Window

ATG Control Center opens a new Document Editor window whenever you open an existing document or create one. It's divided into three synchronized panels, each of which gives you a different view of the open document:



- The **Document Outline** panel at top left, shows the full hierarchy of dynamic elements in the document, including all ATG Servlet Beans, anchors, etc. This lets you quickly find and edit dynamic elements.
- The **Text Editor** panel on the right displays the complete code in the document.
- The **Dynamic Element Editor** panel in the lower left enables you to set the variables in dynamic elements.

The Document Outline, the Text Editor, and the Dynamic Element Editor are synchronized. When you click on one of the elements in the Document Outline, the Text Editor highlights the code that expresses that element, and the Dynamic Element Editor displays the appropriate editor. Similarly, if you move the insertion point to a dynamic element in the Text Editor, the corresponding element is highlighted in the Document Outline.










The full path of the active document appears at the top of the Document Editor window along with the File, Edit, Insert, Window, and Help menus and toolbar. You can use the toolbar to quickly insert dynamic elements in your JSP at the location of the insertion point in the Text Editor. When you click one of these



toolbar buttons, the Document Editor launches a wizard that prompts you for the information needed to complete the dynamic element. The **Preview** button opens the page in your Web browser. Other buttons let you save the document, print it, or view the help system.

## Document Outline

When you open a document in the Document Editor, the Document Outline shows a fully opened hierarchy of all the dynamic elements in the document. Dynamic elements that appear in this view include all ATG Servlet Beans, forms, inputs, and anchors. Other elements, including basic HTML elements, are also displayed. The elements of your document are displayed in the Document Outline with an icon and a text identifier, as follows:

Icon	Element	Text
	Open a JSP or JSPF	pathname of page
	XML page	pathname of page
	anchor	URL of destination
	ATG Servlet Bean	name of component
	imported component (see <a href="#">Viewing Components in Nucleus</a> for a list of the component types and icons used)	name of component
	form	FORM
	<dsp: val ueof>	name of parameter, bean, or bean property
	Java code segment	JAVA
	error	(error)

## Text Editor

The Text Editor displays the text of the active document. You can use the panel as a text editor to define your documents or other elements of your Web application. When you move the insertion point to a dynamic element in the Text Editor, the corresponding Dynamic Element Editor opens in the lower left panel.

## Dynamic Element Editor

The Dynamic Element Editor displays the parameters and values that pertain to any dynamic elements that are highlighted in the Text Editor. The specific editor that appears here depends on the selection in the Text Editor. The Document Editor uses four different types of Dynamic Element Editors:

- [Available Page Values Editor](#)
- [ATG Servlet Bean Editor](#)



- [Anchor Editor](#)
- [Insert Document Editor](#)

### **Available Page Values Editor**

This element editor appears when the insertion point is inside an `<dsp: oparam>` tag or anywhere in the Text Editor **except** within a dynamic element tag. This editor has two tabs: Page Params and Imported Beans:

#### **Page Params**

The Page Params tab lists all parameters that have been declared in the active JSP. Select a parameter and click the **Insert** button to insert that parameter into the page, using the `<dsp: val ueof param=. . . >` tag.

#### **Imported Beans**

The Imported Beans tab lists all JavaBeans that have been imported into the page. You can expand each bean to show its properties. Use the **Insert** button to insert the value of a property into the page and the **Import** button to import additional beans into the page.

### **ATG Servlet Bean Editor**

The ATG Servlet Bean Editor appears when the cursor is anywhere inside or between `<dsp: dropl et></dsp: dropl et>` tags, except when it is between `<dsp: oparam></dsp: oparam>` tags.

The servlet bean editor displays each input parameter and open parameter (OPARAM) defined by the servlet bean. The input parameters are listed in a table that shows these parameters and their values. The value for any input parameter that is not defined in the page is shown as empty. You can enter a value for one of these parameters, and the parameter and its value will be automatically inserted in the page. You can also change the value for a parameter that is defined on the page, and the new value will replace the old value. You can insert an OPARAM by clicking on the arrow to the right of the parameter's name.

### **Anchor Editor**

This element editor appears when the insertion point is anywhere inside or between `<dsp: a href=. . . ></dsp: a>` tags. The Anchor Editor has two parts: Anchor Destination and Open Document button.

#### **Anchor Destination**

This field displays the URL to the anchor link destination. To select a file as the destination document:

1. Click ... to open the Select an Anchor URL dialog. This dialog box displays the directory structure of your document tree.
2. From the top pane, select the document root that holds the destination file. The active document root is currently selected.  
**Note:** All document roots available to your ATG instance display in this tree menu, including those that are part of applications which are not currently running.
3. From the bottom pane, select the file that will be the link destination and then click **OK**.

**Open Document button**

This button is active if the destination is a file within the active application. Click this button to open another Document Editor window for the destination document.

**Insert Document Editor**

This element editor appears when the insertion point is anywhere inside or between `<dsp: include></dsp: include>` tags. The Insert Document Editor has three parts: Inserted Document, Open Document button, and Passed Parameters.

**Inserted Document**

This field displays the URL (relative pathname or absolute URL) of the file being inserted at this point. To select a file as the document to be inserted:

1. Click **...** to open the Select a Document to Insert dialog box. This dialog box displays the directory structure of your Web application document tree.
2. Select the local file to be the destination of the anchor link and click **OK**.

**Note:** The wizard assumes the file to embed is local. For destinations that are not local to the current page or Web application, skip the wizard and instead replace the page attribute with the `src` attribute and enter the entire path and file name. When an application references files in another Web application, make sure `web.xml` contains the context path to the secondary application. See [Absolute URLs](#) for instructions.

**Open Document button**

Click this button to open another Document Editor window for the inserted document.

**Passed Parameters**

Displays a list of parameters and values that are passed to the inserted document.

## Creating a JSP

To create a JSP.

1. In the Pages and Components task area, find or create the folder to store the new JSP.
2. Highlight the destination folder and click **New Document**.
3. In the Select Document Template dialog, choose **Page (JSP)**.
4. In the New JavaServer Page dialog, enter the document's title and click **Finish**.
5. In the Input New Document Name dialog, enter the file name and click **OK**.

**Note:** The names of the document and its folder should be different.

A Document Editor window opens and displays the new document.

**Anatomy of a JSP**

A new JSP looks like this:



---

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>

<dsp: page>

<html >
<head>
  <title>A Simple Test Page</title>
</head>

<body bgcolor="#ffffff">
  <h1>A Simple Test Page</h1>

</body>
</html >

</dsp: page>
```

---

The Document Editor initially supplies these markup tags:

- Title, background color, and heading tags.
- A standard JSP page directive that specifies the DSP tag library and the prefix dsp for the tags in that library:  

```
<%@ taglib uri="http://www.atg.com/taglibs/daf/dspjspTaglib1_0" prefix="dsp" %>
```
- dsp: page tags that enclose the body of the JSP. These tags facilitate ATG page processing and must be included in each page as they are in this example.

### ***Other Items to import***

To comply with the JSP specification, you must import any class you reference in a JSP to make it available to the page. For example, in order to embed Java in a tag and reference ATG specific classes, you must use an `@import` directive to make `atg.servlet` package available to the page. For example:

```
<%@ page import="atg.servlet.*"%>
```

### ***Changing the tag library directive***

A tag library can be deployed in two ways. By default, the DSP tag libraries tld files are kept in the tag library jar file inside `WEB-INF/lib`. The URI used in JSP page directives are defined in the tld itself.

The URI in the page directive must match the URI specified in either the tag library's tld file (as is the case with the DSP tag libraries) or `web.xml`. If you want to define a different URI for the DSP tag libraries, specify the new URI and tld location in `web.xml`. Then, you can use either the default or the new URI in your JSPs.

For instructions on making your J2EE application deployment-ready, see the J2EE information provided by your application server vendor.





If you prefer a prefix other than the default and wish to use the Document Editor, you must use one of the following URIs:

- `http://www.atg.com/taglib/dspjspTaglib1_0`
- `dspTaglib`
- `dsp`

Then you need to modify the tag libraries `web.xml` file and the JSPs that use it.

Although you can set the URI to any value you want, a URI other than these three causes the Document Editor to disregard the prefix setting in the page directive and instead use the default (`dsp`).

When you change the prefix in the DSP tag libraries declaration statement, the Document Editor adjusts to the new prefix and provides it to any new tags you insert. Existing tags must be updated manually.

## Displaying Property Values

JSPs can display the property values of Nucleus components. This gives your application a simple way to display dynamic information based on the states of Java objects.

### Importing Components

In order to display a component's properties in a JSP, you first must import it so it is visible in the Document Editor. This is accomplished by an `import` construct modeled after Java class imports:

```
<dsp:importbean bean="full-component-name" />
```

For example, if you import the `/samples/Student_01` component, the following line is added to your file:

```
<dsp:importbean bean="/samples/Student_01" />
```

To add a `dsp:importbean` tag to a page:

1. Navigate to the Imported Beans tab of the Available Page Values Editor and click **Import**.
2. From the Select Component to Import dialog, pick a component by module or by path, and click **OK**.

After you import a component, the component and its properties become available in the Dynamic Element Editor. Also, importing a component into a page lets the page omit a component's full path name in references to it. You're not required to import any Nucleus components.



## Inserting a dsp:valueof tag

You can specify to display a component property with the dsp: valueof tag. To insert the <dsp: valueof> tag:

1. In the Available Page Values Editor (visible in the Dynamic Element Editor panel), go to the Imported Beans tab and expand the desired component to list its properties.
2. Select the property to display and click **Insert**.

The dsp: valueof tag appears in the Text Editor.

3. To include a default value, remove the final back slash, add an end tag (</dsp: valueof>), and insert your default value between that start tag and end tag.

For example:

---

```
<%@ taglib uri=http://www.atg.com/taglib/daf/dspjspTaglib1_0 prefix="dsp" %>

<dsp: page>
<dsp: importbean bean="/samples/Student_01"/>

<html >
<body>

Name: <dsp: valueof bean="Student_01.name"/><p>
Age: <dsp: valueof bean="Student_01.age"/><p>

</body>
</html >

</dsp: page>
```

---

Note: Because the component /samples/Student\_01 is imported, the inserted dsp: valueof tag includes only the component and property names and excludes the pathname.

## Previewing JSPs

If an open JSP is in a running application, you can preview a JSP in the Document Editor if it is in a running application: simply click the **Preview** button. This opens the page in your Web browser.

You can also open a JSP directly in your browser. For instructions, see the [ATG Installation and Configuration Guide](#).



## Embedding Documents

`dsp:include` lets you embed one page in another. For example, to embed `content.jsp` in `foyer.jsp`:

1. Open `foyer.jsp` in the Document Editor.
2. Click **Insert Document** in the toolbar.

The Select a Document dialog box appears, displaying all files in the document root.

3. Find `content.jsp` click **OK**.

The Document Editor inserts the `dsp:include` tag, using the page attribute to identify the page you are embedding:

```
<dsp:include page="content.jsp"/></dsp:include>
```

**Note:** The Document Editor assumes the included page is in the current Web application. For information about embedding pages from other Web applications, see [dsp:include](#).

## Inserting ATG Servlet Beans

You can use the Document Editor to insert ATG servlet beans. For example, you can insert a `ForEach` servlet bean as follows:

1. Place the cursor in the body of the document where you wish to insert the servlet bean.
2. From the Insert menu, select Insert Servlet Bean. The Insert Servlet Bean dialog box opens.
3. Select Servlet Beans from the Categories list, then select `ForEach` from the Components list. Click **OK**.
4. In the Insert `ForEach` Servlet Bean wizard, click the "..." button.
5. In the Choose a Dynamic Value dialog box, enter the array property whose elements you wish to retrieve. For example:

```
bean="/samples/Student_01.subjects"
```

6. Click **OK**, and then click Next in the wizard.
7. In the Specify output text box, specify the output to display for each property element. For example:

```
<li><dsp:valueof param="element"/></dsp:valueof>
```

8. Click **Next**.
9. In the Specify outputStart text box, enter the text to display before loop execution begins. For example:

```
<p>The student is registered for these courses:
```



10. Click **Next**.
11. Leave the Specify outputEnd text box blank, and click **Next**.
12. Leave the Specify empty text box blank, and click **Finish**.

The document should now contain these tags:

---

```
<dsp: droplet name="/atg/dynamo/droplet/ForEach">
  <dsp: param bean="/samples/Student_01.subjects" name="array"/>
  <dsp: oparam name="outputStart">
    <p>The student is registered for these courses:
  </dsp: oparam>
  <dsp: oparam name="output">
    <li><dsp: valueof param="element"></dsp: valueof>
  </dsp: oparam>
</dsp: droplet>
```

---

### **Importing servlet beans**

To make a servlet bean visible in the Dynamic Element Editor, import the component into the page with `dsp: importbean`. For example:

```
<dsp: importbean bean="/atg/dynamo/droplet/ForEach"/>
```


Importing the servlet bean lets you refer to it without including the entire path, as in this line:

```
<dsp: droplet name="ForEach">
```

By importing a Nucleus component into the pageContext with the `dsp: importbean` tag, other (non-EL) references to that component can exclude the Nucleus address.

## Inserting a Targeting Servlet Bean

The Document Editor in the Control Center includes wizards for embedding targeting servlet beans in your page. To embed a targeting servlet bean in a JSP:

1. Open a Document Editor for the page in which you want the targeted content to appear.
2. Place the cursor in the body of the document, immediately below the two lines you just added.
3. Click the **Insert Targeting Servlet** button, which looks like this:  The Insert Targeting Servlet dialog box opens.
4. Select Personalization from the Modules list, then select Servlet Beans from the Categories list.



5. From the Components list, select the targeting servlet bean you want to use, and click **OK**.
6. In the Specify Targeting Service dialog, select the targeter to use with the servlet bean.
7. Depending on which targeting servlet you selected, the Document Editor presents a series of dialogs that prompt you for the values of the parameters that configure the behavior of the servlet. You can also use the ATG Servlet Bean Editor to set the parameters of the targeting servlet bean.

## Adding Slots

Slots let you display dynamic content on your Web site pages. To add the slot where you wish it to appear in the JSP:

1. Open the JSP.
2. Click **Insert Targeting Servlet Bean** and add an appropriate servlet bean—for example TargetingForEach.
3. In the wizard dialog Specify Targeting Service, specify the slot component. The system enters the slot component as the Targeter parameter of the servlet bean.

## Inserting Forms

You can use the Insert Form wizard to create a form that uses a form handler:

1. Open a document in the Document editor.
2. Place the insertion point within the body of the document, and from the Insert menu, select Insert Form.
3. In the Insert Form Wizard, click the "... " button.
4. In the Select Form Handler Component dialog box, select [all modules] from the Modules list, then select Form Handlers from the Categories list.

The Components list shows you all of the form handlers available.

5. Select a form handler and click **OK**.

The wizard then guides you through the process of creating the form. The dialogs are tailored to the type of form handler you have selected.

## Creating Profile Form Pages

The Control Center provides two shortcuts for creating JSPs that contain profile form:

- Start with a blank page template, and use the Insert Form wizard to insert a profile form.
- Start with a profile form template that contains the type of profile form you want.

## Using the Insert Form Wizard

To create a profile form with the Insert Form wizard:

1. Create a document from the Page (JSP) template.
2. Open the document and place the insertion point within the body,
3. From the Insert menu, select Insert Form.
4. In the Insert Form Wizard, click ...
5. In the Select Form Handler Component dialog box, click the **By Path** radio button. Navigate to /atg/userprofiling, and select ProfileFormHandler. Click **OK**.
6. In the Insert Form Wizard, click **Next**.  
The Configure Input Fields dialog appears, displaying all of the available profile properties.
7. For each profile property to include in the profile form, specify the following:
  - Check **Include** to include the property.
  - In the Input Field Format column, select an input field format from the choices provided. (When you click on the Input Field Format cell for the property, a drop-down list of field formats appears.)
  - In the Default Value column, select whether the current value of the property should appear in the field when the user accesses the page, or that the field should be empty. (In most cases, you should select the current value of the property as the default. If the default is empty, and the user does not fill in the field, then the property is set to null when the user submits the form. Also, note that if the `extractDefaultValuesFromProfile` property of the Profile Form Handler is false, all fields will be empty in the form, regardless of the defaults you select.)
  - Check **Required** if the property is required to have a value. (If a user submits a form without completing a required field, an error results.)
8. Click **Next**. The Configure Submit Buttons and URLs dialog box appears.
9. In the lower part of the dialog box, enter text in the Button Text column for each submit operation you want to include on your page. (The text you enter appears on the button that invokes the operation.
10. To specify pages where you wish to redirect the after the operation succeeds or fails, click ... and select the Web application and file that serves as the success or failure destination. (The cancel operation cannot have an Failure URL associated with it, so that value cannot be specified.)
11. Click **Finish**.



The complete form appears in the Document Editor. Note that the profile property input fields appear on the page in alphabetical order, which might not be the order you desire. Edit the form in the Document Editor until the fields appear in the desired order.

## Using a Form Template

You can create a page containing a profile form by starting from a profile form template. The Personalization module includes the following profile form templates:

- Login Form (JSP)
- Logout Form (JSP)
- Change Password Form (JSP)

Each of these templates is a complete JSP containing a profile form, which you can then edit to match the content and format you want. To create a document from one of these templates:

1. In the Control Center, select the Pages and Components » Pages window.
2. From the File menu, choose **New Document**.
3. In the Select a Document Template dialog box, select **Personalization** from the Modules list, and then select the template you want from the Templates list. Click **OK**.
4. In the Document Title dialog box, enter a title for the document, and click **Finish**.
5. In the Input New Document Name dialog box, select where in the hierarchy you want to place the document, enter a name for the document, and then click **OK**.

The document is created, and opens in the Document Editor.

**Note:** Each of these forms uses the properties in the standard profile template that comes with ATG Personalization. You might need to edit the profile forms to match the profile properties of your specific profile template.







# Index

## A

- AddBusinessProcessStage servlet bean, 255
- AddMarkerToProfile servlet bean, 258
- advanced search, 113
  - combined with other search types, 116
  - handler properties, 113
  - value ranges, 116
- anchor tag
  - pass page parameter, 36
- application scope
  - set for attribute, 19
- array property
  - iterate over elements, 44, 53
  - set with checkboxes, 94
  - set with list box, 95
- attribute. See EL variable

## B

- BeanProperty servlet bean, 259
- boolean property
  - set in form, 91
- business process tracking
  - add stages, 255
  - check stages, 295, 309
  - remove stages, 339

## C

- Cache servlet bean, 262, 263
- cancelURL, 102
- CanonicalItemLink servlet bean, 266
- checkboxes, 91
  - group, 94
  - set array property, 94
- Clicks a Link scenario element, 69
- CollectionFilter servlet bean, 268
- combination search, 116
- Compare servlet bean, 270
- component. See Nucleus component, component property values
- component property values
  - dictionary, 25
  - display, 20, 442
  - display nested property values, 21
  - edit, 426
  - hashtable, 25



- link, 426
- set in form, 90
- set with
  - dsp:a, 22
  - dsp:setvalue, 22
  - hidden input tag, 97
  - submit, 98
- unlink, 427
- Components By Module view, 421
- Components By Path view, 423
- configuration layers, 428
- context path
  - configure multiple, 42
  - specify in dsp:include, 42
- credit card converter, 28
- CurrencyFormatter servlet bean, 278

## D

- date converter, 31
  - mindate and maxdate, 32
- dictionary, property values in, 25
- document
  - create, 432
  - edit. See Document Editor
  - embed, 443
  - initial rendering, 433
  - manage, 431
  - outline, 437
  - search, 432
  - templates, 432
- Document Editor, 435
  - Anchor Editor, 438
  - ATG Servlet Bean Editor, 438
  - Available Page Values Editor, 438
  - Dynamic Element Editor, 437
  - Insert Document Editor, 439
  - Text Editor, 437
- Document Outline, 437
- DOM
  - cache documents, 78
  - process nodes, 74
  - transformed from XML, 73
- drop-down list, 92
  - control initial selection, 92
  - nodefault attribute, 92
- DSP tag library, 14
- dsp:a, 22
- dsp:form, 87
  - synchronized attribute, 99
- dsp:getvalueof, 15
  - compared to dsp:tomap, 18
  - set attribute from JavaBean component, 16
  - set attribute from page parameter, 16
  - set attribute from static value, 16
  - set scripting variable, 19
- dsp:importbean, 18
- dsp:include
  - versus jsp:include, 39

- dsp:input, 93
- dsp:option, 92
- dsp:select, 92
- dsp:setvalue, 22
- dsp:textarea, 93
- dsp:tomap
  - compared to dsp:getvalueof, 18
  - set attribute from component property, 17
  - set attribute from Dynamic Bean, 17
  - set scripting variable, 19
- dsp:tomap, 17
- dsp:valueof, 20
  - default value, 20
  - valueishtml attribute, 21
- dsp:valueof:valueishtml attribute, 38
- Dynamic Bean
  - set EL variable, 17

## E

- EL variable
  - default scope, 19
  - set, 15
  - set from
    - Dynamic Bean, 17
    - JavaBean, 16
    - page parameter, 16
    - static value, 16
  - set with
    - dsp:importbean, 18
- embedded files
  - frame tag, 196
  - iframe tag, 199
  - img tag, 200
  - include tag, 203
  - link tag, 211
  - pass parameters to, 42
- EndTransactionDroplet servlet bean, 280
- error handling
  - form handlers, 102
  - multi-profile form handler, 149
  - ProfileFormHandler, 142
- ErrorMessageForEach servlet bean, 282
- escape html converter. See valueishtml converter

## F

- folder
  - create, 431
  - delete, 431
- For servlet bean, 285
- ForEach servlet bean, 288
  - iterate over array property elements, 44
- form
  - array property, 94
  - cancel, 102
  - checkboxes, 91
  - create, 87
  - drop-down list, 92
  - DSP tag conventions, 88



- embed JSP in, 89
- hidden input tag, 97
- order of tag processing, 99
- redirect, 102
- reset, 102
- set component properties, 90
- set nonarray (scalar) properties, 91
- submit, 97
- synchronize submissions, 99
- text entry field, 93
- form handler, 101
  - classes, 101
  - display exceptions, 103
  - error alerts, 103
  - error detection, 103
  - error handling, 102
  - MultiProfileAddFormHandler, 145
  - MultiProfileUpdateFormHandler, 147
  - ProfileFormHandler, 135
  - RepositoryFormHandler, 123
  - SearchFormHandler, 105
  - SimpleSQLFormHandler, 415
- form input
  - name attribute, 88
- Format servlet bean, 290
- FormSubmission events
  - filtered by scenario, 69
  - listened for by scenario, 69

## G

- GetDirectoryPrincipal servlet bean, 292

## H

- HasBusinessProcessStage servlet bean, 295
- HasEffectivePrincipal servlet bean, 297
- HasFunction servlet bean, 299
- hashtable, property values, 25
- hidden input tag, 97
- hierarchical search, 112
  - handler properties, 113
- HTML table
  - configure TableInfo component, 161
  - display repository data, 161
  - localize column headings, 163
  - manage sorting, 164
  - set default sort order, 165
  - sort column data, 162
  - sorted by users, 169
- HTML tables
  - sort indicators, 172
- hyperlink
  - set query string, 36

## I

- icons, 431, 437
- image
  - specify for submit control, 98
- include directive, 39
- indexed property
  - access, 23
  - specify index with page parameter, 24
  - specify index with property, 24
- indexes of components, 430
- input parameters, 45
- Insert Form Wizard, 446
- IsEmpty servlet bean, 301
- IsNull servlet bean, 303
- ItemLink servlet bean, 304
- ItemLookupDroplet servlet bean, 307

## J

- JavaBean component
  - set EL variable, 16
- JSP
  - embed another JSP, 39
  - embed JSP fragment, 39
  - forms in, 87
  - import servlets, 440
  - invoke from another Web application, 41
  - manage, 431
  - render on wireless device, 43
  - standard page layout, 439
- JSTL tag library, 14

## K

- keyword search, 108
  - combined with other search types, 116
  - handler properties, 108
  - logical operators, 109
  - multi-value property, 110
  - quick searching, 110
  - single-value property, 110

## L

- list box
  - set array property, 95
- listbox
  - multiple selection, 95
- logical operators
  - keyword search, 109

**M**

- methods in a component, 427
- MIME types, 83
- module, 422
- MostRecentBusinessProcessStage servlet bean, 309
- multi-profile form handler, 143
  - code example, create mutiple profiles, 154
  - code example, update mutiple profiles, 157
  - create multiple profiles, 145
  - error handling, 149
  - iterate over user profiles, 144
  - MultiProfileAddFormHandler properties, 145
  - MultiProfileUpdateFormHandler properties, 147
  - navigation properties, 148
  - scope, 150
  - set values, 144
  - submit operations, 143
  - update multiple profiles, 147
- MultiProfileAddFormHandler. See also multi-profile form handler
- MultiProfileUpdateFormHandler. See also multi-profile form handler
- multisite
  - change site context, 60, 355
  - code JSPs, 55
  - empty site context, 60
  - generate link to another site, 61, 360
  - get site configuration, 58, 293
  - get site for product, 55, 356
  - site priority used by SiteIdForItem, 56
  - test site sharing, 59, 353, 362

**N**

- NavHistoryCollector servlet bean, 312
- navigation
  - tracked by scenario, 69
- NodeForEach servlet bean, 74, 315
- NodeMatch servlet bean, 316
- Nucleus component
  - configuration layers, 428
  - configure, 425
  - create, 424
  - delete, 423
  - EL access to, 18
  - import, 441
  - import to page, 18
  - index, 430
  - pass name via page parameter, 48
  - properties, 426
  - search, 424
  - slots, 445
  - start, 429
  - stop, 429
  - view by modue, 421
  - view by path, 423

- nullable converter, 34
- number converter, 34

**O**

- open parameters, 46
  - oparam tag, 212
- output parameters, 45

**P**

- page directive
  - change tag library URI, 15
  - import tag library, 15
  - WML output, 43
- page parameters, 36
  - display HTML value, 38
  - inheritance, 37
  - obtain query string, 37
  - override, 37
  - pass component name, 48
  - pass in anchor tag, 36
  - pass to a page, 36
  - pass to embedded files, 42
  - precedence, 38
  - scope, 37
  - set, 37
  - set EL variable, 16
  - set with dsp:setvalue, 22
  - use as array index, 24
- page scope
  - set for EL variable, 19
- PageEventTriggerDroplet servlet bean, 317
- parameters. See servlet bean parameters
  - page. See page parameter
  - servlet beans, 212
- parametric search. See advanced search
- PipelineChainInvocation servlet bean, 318
- PossibleValues servlet bean, 118, 321
- profile form
  - create, 445
  - create from template, 447
  - create with wizard, 446
- ProfileFormHandler, 135
  - code example, 151
  - navigation properties, 141, 142
  - properties, 137
  - scope, 142
  - set profile properties, 136
  - submit operations, 126, 135
  - update multi-valued properties, 139
  - update repository item properties, 126



ProfileHasLastMarker servlet bean, 325  
 ProfileHasLastMarkerWithKey servlet bean, 327  
 ProfileHasMarker servlet bean, 330  
 profiles. See user profiles  
 ProtocolChange servlet bean, 331

## R

radio button  
   checked attribute, 92  
   default attribute, 92  
 radio buttons  
   checked attribute, 92  
   group, 91  
 Range servlet bean, 53, 335  
 Redirect servlet bean, 337  
 RemoveAllMarkersFromProfile servlet bean, 338  
 RemoveBusinessProcessStage  
   servlet bean, 339  
 RemoveMarkersFromProfile servlet bean, 342  
 repository  
   display data in HTML table, 161  
   update, 123  
 RepositoryFormHandler, 123  
   Map properties, 130  
   navigation properties, 127  
   properties, 124  
   update hierarchical properties, 126  
   update multi-valued properties, 128  
 RepositoryLookup servlet bean, 344  
 request scope  
   set for EL variable, 19  
 required converter, 35  
 RQLQueryForEach servlet bean, 311, 347  
 RQLQueryRange servlet bean, 351  
 RuleBasedRepositoryGroupFilter servlet bean, 352

## S

scalar property  
   set in form, 91  
 scenario, 69  
   Clicks a Link element, 69  
   create for slot, 68  
   filter FormSubmission events, 69  
   listen for FormSubmission events, 69  
   track user navigation, 69  
 scenarios  
   slots. See slot  
 scope  
   default, 19  
   set for attribute, 19  
 scripting variables, 19  
 search form, 105, See also advanced search; combination  
   search; hierarchical search; keyword search; text search  
   basic handler properties, 106  
   clearQuery operation, 117  
   controls to display search values, 118  
   create handler, 106  
   search operation, 117

  submit operations, 117  
 search results  
   display, 119  
   link to, 120  
   organize, 120  
   SearchFormHandler properties, 119  
 SearchFormHandler, 106  
   advanced search properties, 113  
   basic properties, 106  
   hierarchical search properties, 113  
   keyword search properties, 108  
   search results properties, 119  
   text search properties, 111  
 servlet bean, 44  
   AddBusinessProcessStage, 255  
   business process tracking, 254  
   database and repository access, 246  
   display nested property values, 49  
   editor, 438  
   embed in page, 44  
   import into JSP, 48, 444  
   InventoryFilterDroplet, 268  
   nest, 50  
   personalization, 250  
   process XMLdocuments, 249  
   reference outer bean parameters, 52  
   targeted content, 65  
   transaction, 250  
   use to display property values, 44  
 servlet bean parameters, 45  
   access with EL, 47  
   input, 45  
   open, 46  
   output, 45  
   reference in outer bean, 52  
   scope, 46, 47  
 session confirmation  
   require, 100, 183, 194  
 session scope  
   set for EL variable, 19  
 SimpleSQLFormHandler, 415  
   navigation properties, 417  
   properties, 415  
   submit handler methods, 417  
 site. See multisite  
 slot, 68  
   create scenario, 68  
   types of items in, 68  
 sortable tables  
   sort direction types, 169  
 SQL database  
   manage with SimpleSQLFormHandler, 415  
 SQLQueryForEach servlet bean, 367  
 SQLQueryRange servlet bean, 371  
 static value  
   set EL variable, 16  
 submit input tag, 97  
   set component property, 98  
   specify image, 98



Switch servlet bean, 53, 374

## T

table. See also HTML table

TableForEach servlet bean, 53

TableForEach servlet bean, 377

TableInfo, 161

- code example, 174

- column object properties, 164

- configure, 161

- localize column headings, 163

- sort column data, 162

TableRange servlet bean, 53, 381

tag converters

- anchor tag, 183

- credit card converter, 28

- date converter, 31

- dsp:textarea, 236

- input tag, 206

- nullable converter, 34

- number converter, 34

- required converter, 35

- select tag, 224

- valueishtml converter, 35

tag libraries

- deploy, 15

- DSP, 14

- import, 15

- JSTL, 14

- page directive URI, 15

targeting servlet bean, 65

- create slots, 445

- embed in page, 444

- event logging, 67

- limit result set, 68

- sort output, 66

- use slots, 445

- use with XML transformation, 79

TargetingArray servlet bean, 384, 387

TargetingForEach servlet bean, 390

TargetingRandom servlet bean, 392

TargetingRange servlet bean, 396

TargetPrincipalsDroplet servlet bean, 398

TemplateMap component, 78

Text Editor, 437

text entry field, 93

- override property value, 93

text search, 110

- combined with other search types, 116

- enable full-text searching, 112

- handler properties, 111

- simulated, 112

TransactionDroplet servlet bean, 400

## U

URL

- absolute, 41

- relative, 40

user profiles

- createmultiple, 145

- form handlers, 135, See also ProfileFormHandler, multi-profile form handlers

- iterate over, 144

- set properties, 136

- set properties in multiple, 144

- update multiple, 147

UserListDroplet servlet bean, 401

## V

valueishtml attribute, 21, 38

valueishtml converter, 35

ViewPrincipalsDroplet servlet bean, 403

## W

Wireless Application Protocol, 43

wireless device

- render JSP on, 43

WML

- specify for JSP output, 43

WML support

- go tag, 198

- postfield tag, 218

WorkflowInstanceQueryDroplet servlet bean, 406

WorkflowTaskQueryDroplet servlet bean, 409

## X

XML transformation, 71

- document cache, 78

- DOM cache, 78

- DOM node processing, 74

- in JSP, 73

- nested servlet beans, 84

- optimize performance, 80

- pass parameters to XSL stylesheet, 80

- serialized servlet beans, 85

- servlet beans, 72, 249

- template location, 78

- template use, 76

- use with targeting servlet bean, 79

- XMLtoDOM, 73

XMLtoDOM servlet bean, 73

XMLToDOM servlet bean, 411

XMLTransform servlet bean, 414

XSL stylesheet

- encode output stream, 83

- invoke with page parameters, 82

- invoke with URL query parameters, 82

- output attributes, 82

- output method, 83

- pass parameters to, 80

- set MIME type, 83

XSLT, 71