Dynamic Server Pages and Output Templates Developer's Guide

Innovation Release

Version 10.0

April 2017

Document ID: IS-DSP-DG-100-20170419

Table of Contents

About this Guide

This guide is for those developers who want to build browser-based clients using Dynamic Server Pages (DSPs) and who want to build output templates to format the results of services executed on webMethods Integration Server.

Note: This guide describes features and functionality that may or may not be available with your licensed version of webMethods Integration Server. For information about the licensed components for your installation, see the **Settings > License** page in the webMethods Integration Server Administrator.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.

Convention	Description
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Using Dynamic Server Pages (DSPs)

What Is a Dynamic Server Page?

A *dynamic server page* (DSP) is a document embedded with special codes (tags) that instruct webMethods Integration Server to perform certain actions when an HTTP (or HTTPS) client requests the document. DSPs are used to construct browser-based applications. Because they are HTML based, they can be used to build complex user interfaces that includes any valid construct (e.g., forms, cascading style sheets, JavaScript) recognized by the client's browser.

Important: When Integration Server returns a DSP, it always sets the value of the HTTP `content-type` header field to `text/html`. Therefore, a DSP should only contain HTML content and should only be used by clients that recognize and accept this content type.

What Does a DSP Look Like?

A DSP looks like an ordinary HTML document that contains additional tags enclosed in % symbols (e.g., `%loop%`). When a client requests a DSP, Integration Server executes the action specified by the tag and substitutes the result of that action (based on the rules of the tag) in the document it returns to the client.

Note: A DSP tag is never sent to the client; the client receives only the result of the tag.

The following is an example of a very basic DSP (tags are in bold). In this example, the DSP invokes a service by way of the `%invoke%` tag. The `%loop%` … `%endloop%` block loops over the list of documents (called *orders*) that the service returns and inserts the results into the HTML document.

```
<HTML>
<HEAD>
<title>Order Tracking System>/title>
</HEAD>
<BODY>
<h1>Current Orders</h1>
%invoke orders:showOrders%
%loop orders%
 <p>Date: %value orderDate% PO Number %value orderNum%</p>
%endloop%
%endinvoke%
</BODY>
```

When Do You Use DSPs?

DSPs are used to build browser-based clients (i.e., clients that use a web browser to retrieve documents). They allow you to construct a more secure and flexible user interface than can be built by directly invoking a service from a browser.

webMethods Integration Server Administrator is a good example of the type of user interface you can create with DSPs. The interface for this application is composed entirely of DSPs. You may want to refer to it for design ideas and examples of how to use particular tags. To examine the DSPs that make up the server's user interface, look at the DSP files in the *Integration Server_directory*\packages\WmRoot\pub directory on your Integration Server.

What Are the Advantages of Using DSPs?

In the example in "What Does a DSP Look Like?" on page 10, the DSP invokes a service and displays its results. You could accomplish the same thing by invoking the service directly from a browser and applying an output template to the result. However, DSPs have several advantages over directly invoking a service with a URL:

- They conceal the INVOKE mechanism and the name of the service from the user.
- They give you the flexibility to change the name of a service or replace one service with another without changing the way in which the end user invokes the service. (The user always invokes the same DSP, whose contents you can change as needed.)
- They can easily be updated and extended.
- They allow you to execute multiple services via a single request.
- They allow you to conditionally execute a service based on run-time input. For example, you might build a DSP that actually contains several different HTML pages, and use the `%switch%` tag to select among them.

Creating DSPs

To create a DSP, you must compose it with a text editor and then save it on Integration Server (see "Publishing DSPs" on page 12). Unlike output templates, you do not create DSPs with Software AG Designer.

When you build a DSP, do the following:

- Type literal text exactly as you want it to appear in the document that you want Integration Server to return to the client.
- Insert DSP tags at the points where you want their results to appear. For a summary of valid DSP tags and how to use them, see "Using the DSP Tags" on page 17.
- Make sure that your HTML file contains the proper encoding or META tag for the encoding you use, as this may affect the browser or parser performance outside the Integration Server.

Important: Make sure the document that you create resolves into a valid HTML document.

Note: While building your DSP, keep in mind that at run time Integration Server will process it once, from top to bottom.

File Encoding and Character Set Limitations

The file encoding you use limits the characters that you can use in your DSP (including the data inserted into your DSP using %VALUE% statements) to those in the character set of the encoding you choose. It also limits any translated versions of the DSPs to the same character set. Generally it is a good idea to use the UTF-8 (Unicode) encoding, since Unicode supports nearly all of the characters in all of the world's languages. You will not lose any data if you choose to save your DSP in UTF-8; any data entered into a form will be properly interpreted by Integration Server.

The encoding you choose also applies to any localized (translated) versions of the DSP that you create, so you should choose a character encoding that supports all of the languages for which you will create localized DSPs. For details, see "DSPs and Output Templates in Different Languages" on page 71.

Publishing DSPs

To run a DSP, you must publish it on an Integration Server. To do this, take the following general steps.

1. Save the DSP document in a text file that has a ".dsp" extension. For example: `showorders.dsp`.
2. Place the DSP file in the pub directory of the package in which you want the dsp to reside. For example:
   - To publish a DSP in the orders package, you would copy it to:

     Integration Server_directory\packages\orders\pub
   - To publish a DSP in the status subdirectory within the orders package, you would copy it to:

     Integration Server_directory\packages\orders\pub\status

For details about publishing DSPs in other languages, see "DSPs and Output Templates in Different Languages" on page 71.

Securing DSPs

The following sections describe how you can secure DSPs against unauthorized access, secure against cross site scripting (XSS) attacks, and limit the external URLs to which a page can be redirected.

Securing DSPs Against Unauthorized Access

When you publish a DSP, you need to configure the server's security mechanisms to protect the DSP from unauthorized access. DSPs have two levels of security protection you need to set.

- **Access to the DSP itself.** Access to a DSP is controlled by an Access Control List (ACL). An ACL specifies which users have permission to retrieve the DSP. An ACL allows you to make access to the DSP as liberal (e.g., allow access to anyone) or as restrictive (e.g., restrict access to only certain people) as you need.

  To assign an ACL to a DSP, you must update (or create) the .access file in the directory where the DSP resides. For procedures, see "Assigning ACLs to Files the Server Can Serve" in *webMethods Integration Server Administrator's Guide*.

  Note: Unlike a service, access to a DSP cannot be restricted to a particular port. Thus you do not specify port-level controls for a DSP.

- **Access to services invoked by the DSP.** When a user requests a DSP, the services invoked by the DSP are subject to a port-level check (against the port on which the DSP was requested) and an ACL check (against the user that requested the DSP). To ensure that the services in your DSP execute successfully, you must do the following:

  - Make sure that the services it invokes are allowed to execute on the port(s) where the DSP will be requested.
  - Make sure that users who are authorized to use the DSP are also authorized to execute the services that the DSP invokes. (For convenience, you might want to assign the same ACL to the DSP and to the services it invokes.)

  For information about configuring port-level security and assigning ACLs to services, see *webMethods Integration Server Administrator's Guide*.

  Note: A service that is internally invoked by a service in a DSP is not subject to security control unless **Enforce Execute ACL** is set (in the Properties panel) for the internally invoked service. When this option is set, the server performs an ACL check on the service using the user ID under which the DSP was initially requested.

Securing DSPs Against Cross Site Scripting Attacks

If you have custom DSPs that use the `%value` *`Variable`*`%` tag, the output from the tag might be vulnerable to cross site scripting (XSS) attacks. To prevent these cross site scripting attacks, set the watt.core.template.enableFilterHtml parameter to `true` (the default). When this parameter is true, the output from a `%value` *`Variable`*`%` tag, including XML and JavaScript, is HTML encoded.

When the watt.core.template.enableFilterHtml parameter is set to `true`, if you do not want Integration Server to HTML encode the output from a `%value` *`Variable`*`%`

tag, you can use the `encode(none)` option of the `%value` *`Variable`*`%` tag, (`%value` *`Variable`* `encode(none)%`).

If you do not want Integration Server to HTML encode the output from any `%value` *`Variable`*`%` tag in *all* DSPs, set the watt.core.template.enableFilterHtml parameter to `false`. Setting the watt.core.template.enableFilterHtml parameter to `false` does not override settings of the `%value` *`Variable`*`%` tag's `encode` option.

Important: If you use `encode(none)` so that the output from a `%value` *`Variable`*`%` tag is not HTML encoded, that value is vulnerable to cross site scripting attacks. If you set the watt.core.template.enableFilterHtml parameter to `false`, all DSPs that use the `%value` *`Variable`*`%` tag are vulnerable to cross site scripting attacks.

For more information about the `encode(none)` option, see "%value%" on page 66. For more information about the watt.core.template.enableFilterHtml parameter, see *webMethods Integration Server Administrator's Guide*.

Securing DSPs Against CSRF Attacks

Integration Server adds CSRF secure tokens in DSPs dynamically thereby ensuring that the custom DSPs are secured against CSRF attacks.

However, Integration Server does not insert CSRF secure tokens in custom DSPs that use the JavaScripts Location object such as document.location and window.location.href. You must update these pages manually.

For example, if you have the following code in your custom DSP:

```
document.location="ldap-settings.dsp";
```

You must replace it with the following code, enabling the GET request:

```
if(is_csrf_guard_enabled && needToInsertToken) {
   document.location="ldap-settings.dsp?"
   + _csrfTokenNm_ + "=" + _csrfTokenVal_;
} else {
   document.location="ldap-settings.dsp";
}
```

You do not have to define the JavaScript variables *_csrfTokenNm_*, *_csrfTokenVal_*, *is_csrf_guard_enabled*, and *needToInsertToken*. But, you must import *Integration Server_directory*/WmRoot/csrf-guard.js to your DSP before using these variables, if you have not already imported /WmRoot/webMethods.js.

In GET requests, Integration Server inserts the CSRF secure token in the URL, thus displaying the CSRF secure token. When the CSRF guard is enabled (**Security > CSRF Guard** page in the webMethods Integration Server Administrator), to further secure the DSPs, Software AG recommends that you replace the GET requests with POST requests. POST requests eliminate the risk of sending the CSRF secure tokens in URLs. To replace a GET request by a POST request, pass the values as HTML form properties. To create new HTML form and set the properties in the form, use the `createForm(<FORM_ID>`,

`<ACTION>, "POST", <PARENT_TAG>)` and `setFormProperty(<FORM_ID>, <PROPERTY_ID>, <PROPERTY_VALUE>)` methods defined in JavaScript `webMethods.js`.

For example, if the CSRF guard is enabled, to convert the above GET request code to POST, replace it with the following code:

Note: If the CSRF guard is disabled, continue to use the GET request.

```
  if(is_csrf_guard_enabled && needToInsertToken)
{
createForm("htmlForm_listeners ", 'ldap-settings.dsp', "POST", <PARENT_TAG>);
setFormProperty(“htmlForm_listeners”, _csrfTokenNm_, _csrfTokenVal_);
htmlForm_listeners.submit();
} else {
document.location="ldap-settings.dsp";
}
```

The `<PARENT_TAG>` can be head or body based on whether this code belongs to head or body of the DSP.

Integration Server inserts CSRF secure tokens in the links in DSPs only if these links point to a DSP. If these links do not point to a DSP, you must update these links manually to include the CSRF secure tokens. For example, if you have the following code in your DSP:

```
<a href="/invoke/wm.sap.Transaction/viewAs?type=xml</a>
```

If the CSRF guard is enabled, to convert it to POST request, create a new HTML form as shown below and change the link in the DSP:

Note: If the CSRF guard is disabled, continue to use the GET request.

```
if(is_csrf_guard_enabled && needToInsertToken)
{
createForm(“htmlform_transactionView”,
“="/invoke/wm.sap.Transaction/viewAs”, POST, <PARENT_TAG>);
setFormProperty(“htmlform_transactionView”, “type” “xml”);
setFormProperty(“htmlform_transactionView”, _csrfTokenNm_, _csrfTokenVal_);
<a href=”javascript:document.htmlform_transactionView.submit();></a>
} else {
<a href="/invoke/wm.sap.Transaction/viewAs?type=xml</a>
}
```

If the links in DSP point to another DSP, Integration Server automatically inserts CSRF secure token in the links. To further enhance the security, it is recommended that you convert the link in DSP as a POST request if it points to another DSP, provided the CSRF guard is enabled. For example, if you have the following code in your DSP:

```
<a href="security-ports-add.dsp">
```

After Integration Server inserts the CSRF secure token in the URL, the code is changed to the following:

```
<a href="security-ports-add.dsp?secureCSRFToken=<token_id>">
```

If the CSRF guard is enabled, to convert it to POST request, create a new HTML form as shown below and change the link in the DSP:

Note: If the CSRF guard is disabled, continue to use the GET request.

```
if(is_csrf_guard_enabled && needToInsertToken) {
createForm(“htmlform_security_ports”, “security-ports-add.dsp”, “POST”, <PARENT_TAG>);
setFormProperty(“htmlform_security_ports”, “action”, “add”);
<a href="javascript:document.htmlform_security_ports.submit();">Add Port</a>
} else {
<a href="security-ports-add.dsp?action=add">Add Port</a>
}
```

For more information about configuring CSRF guard in Integration Server, see *webMethods Integration Server Administrator’s Guide*.

Limiting the External URLs that Can Be Used for Redirection

Use the `%validConst%` tag in custom DSPs to specify a list of URLs to which a page can be redirected. By specifying the `%validConst%` tag, the page can only be redirected to the URLs you specify.

For more information about how to use this tag, see “Specifying a List of Permitted URLs for Redirection” on page 33.

Requesting DSPs

To process a DSP, you request it from a browser using the following URL format:

http://*hostName*:*portNum*/*packageName*/*fileName* .dsp

where:

hostName	Is the host name or IP address of the Integration Server on which the DSP resides.
portNum	Is the port number on which the Integration Server listens for HTTP requests.
packageName	Is the name of the package to which the DSP belongs. *packageName* must match the package directory in which the DSP resides within *Integration Server_directory*\packages on the server. If you do not specify a package name, the server looks for the named DSP in the Default package. **Note:** This parameter is case-sensitive.
fileName.dsp	Is the name of the file containing the DSP. This file name must have a “.dsp” extension, and it must reside within the pub directory (or a subdirectory beneath pub) under the package directory named in *packageName* . If the DSP resides in a

subdirectory, include the name of that subdirectory in the file name (see example below).

> **Note:** This parameter is case-sensitive.

Examples

The following URL retrieves showorders.dsp from a package named ORDER_TRAK on a server named rubicon:

```
http://rubicon:5555/ORDER_TRAK/showorders.dsp
```

The following URL retrieves showorders.dsp from a package named ORDER_TRAK on a server named rubicon:

```
http://rubicon:5555/ORDER_TRAK/showorders.dsp
```

The following URL retrieves showorders.dsp from the STATUS subdirectory in a package named ORDER_TRAK on a server named rubicon:

```
http://rubicon:5555/ORDER_TRAK/STATUS/showorders.dsp
```

The following URL retrieves showorders.dsp from the Default package on a server named rubicon:

```
http://rubicon:5555/showorders.dsp
```

Hyperlinks to DSPs

Typing the DSP's URL on the address line in your browser is one way to run a DSP. However, when you use DSPs to build a user interface, you will often invoke DSPs from HTML forms and links as shown in the following example.

```
<HTML>
<HEAD>
   <title>Order Tracking System</title>
</HEAD>
<BODY>
<A HREF=/ORDER_TRAK/showorders.dsp>Show
Orders</A>
</BODY>
</HTML>
```

Using the DSP Tags

To develop a DSP, you embed DSP tags where you want the results of the tags to appear. The following is a summary of tags that you can use to build DSPs. For a complete description of each tag, see "Tag Descriptions" on page 45.

> **Important:** DSP tags are case sensitive. In your DSP, you must type them exactly as shown below (e.g., type `%loop%`, not `%LOOP%`).

Use this tag...	To...
`%invoke%`	Invoke a service within a DSP.
`%value%` `%rename%` `%scope%`	Manipulate variables.
`%ifvar%` `%switch%`	Conditionally process a block of code within a DSP.
`%loop%`	Reiterate a block of code within a DSP.
`%include%`	Insert the content of a text file (which may contain additional DSP tags) into the DSP.
`%validConst %`	Limit the external URLs that can be accessed via redirection.
`%comment%`	Denote a comment within a DSP. Comments are neither processed by the DSP Processor nor returned to the requestor.

Note: Integration Server automatically resolves tags when a client requests that DSP via an HTTP or HTTPS request. If you want to resolve tags within a document at some arbitrary point in a service, you can explicitly run the DSP Processor against the document using the services in the pub.report folder. For information about using these services, see "Arbitrarily Processing DSP Tags" on page 34.

Begin...End Constructs

Many DSP tags have both beginning and ending elements. When you use the `%loop%` tag, for example, you enclose the code over which you want the DSP Processor to iterate, within a `%loop%`...`%end%` construct.

To make your DSP easier to read, you can append a suffix to the `%end%` element of any construct to visually associate it with its beginning element. For example, in the following DSP, the `%end%` element of the `%loop%` construct is named `%endloop%` and the `%end%` element for the `%ifvar%` construct is named `%endifvar%`.

```
.
.
.
%ifvar orders%
 %loop orders%
 <p>Date: %value orderDate%PO Number: %value orderNum%</p>
 %endloop%
```

```
%endifvar%
.
.
.
```

Be aware that only the first three characters of an %end% element are significant. The DSP Processor ignores any suffixes that you add and simply associates an `%end%` element with the most recent beginning element (in other words, the `%end%` element always ends the *current* construct). This means that you can nest one construct within another (as shown above), but you cannot overlap them.

For example, you cannot use the `%comment%`…`%endcomment%` construct to "remark out" an `%endloop%` element as shown in the following sample. Because the DSP Processor ignores suffixes, the first `%endloop%` tag would end the comment block, and the `%endcomment%` tag would end the loop.

```
  .
  .
  .
  %ifvar orders%
   %loop orders%
   <p>Date: %value orderDate% PO Number: %value orderNum%</p>
%comment%
   <p>Buyer: %value orderDate% PO Number: %value orderNum%</p>
   %endloop%
%endcomment%
   <p><b>Shipping Details:</b></p>
   <p>Date Shipped: %value shipDate%<br>
    Carrier: %value carrier% %value serviceLevel%<br>
    =====================================================</p>
   %endloop%
  %endifvar%
  .
  .
  .
```

Invoking Services Using the %invoke% Tag

You use the `%invoke%` tag to invoke a service in a DSP. When this tag is processed, Integration Server executes the specified service at the point where the tag appears and returns the results of the service to the DSP processor.

The basic format of the `%invoke%` tag is as follows, where *serviceName* is the fully qualified name of the service that you want to invoke:

```
%invoke serviceName%
Block of Code
[%onerror%
Block
of Code ]
%end%
```

Example

```
.
.
.
%invoke orders:getShipInfo%
  <p>
  Date Shipped: %value shipDate%<br>
  Carrier: %value carrier% %value serviceLevel%
  </p>
%onerror%
  %include standarderror.txt%
%endinvoke%
.
.
.
```

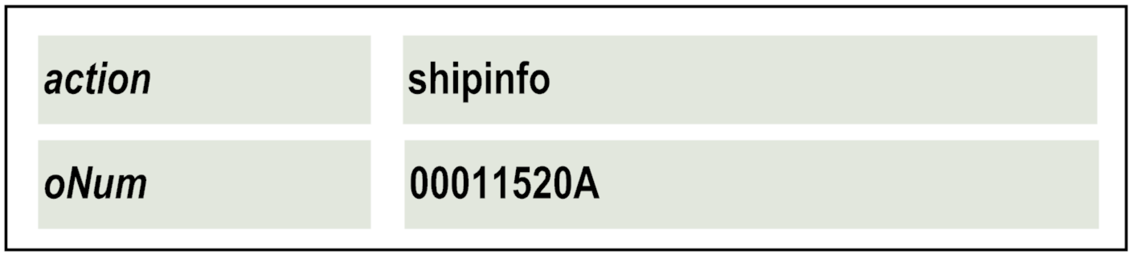
What Is Scope?

When you invoke a service in a DSP, notice that you do not specifically state which parameters you want to pass to it. Instead, the service automatically receives an IData object containing all the variables that are in the DSP's current *scope*.

Scope refers to the set of variables upon which a DSP can operate directly. When a DSP is initially invoked, its scope encompasses the set of *name=value* pairs it receives (via GET or POST) from the requestor. For example, if you were to request a DSP with the following URL:

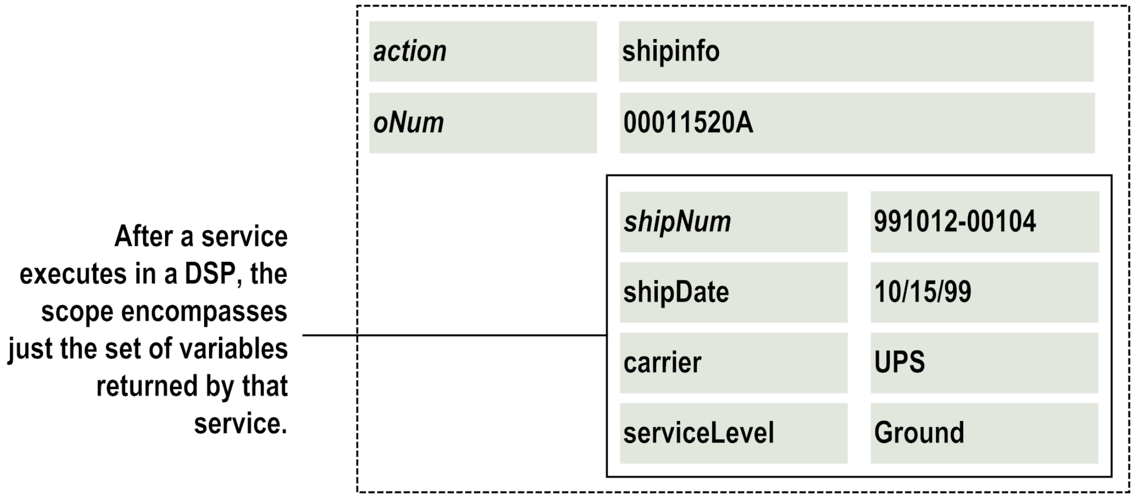
```
  http://rubicon:5555/ORDER_TRAK/getorderinfo.dsp?
action=shipinfo&oNum=00011520
```

Its initial scope would look like this:



action	shipinfo
oNum	00011520A

After you execute a service with the `%invoke%` tag, the scope automatically switches to encompass the set of variables returned by that service. For example, if the DSP in the preceding example invokes a service that returns shipping information, that DSP's collection of variables would look like this after the service executes. Note that just the variables returned by the service are within scope.


action
shipinfo
oNum
00011520A
shipNum
991012-00104
shipDate
10/15/99
carrier
UPS
serviceLevel
Ground
After a service executes in a DSP, the scope encompasses just the set of variables returned by that service.

Conceptually, you can think of a DSP as maintaining its run-time variables in a set of nested containers. It starts with a container that holds the variables submitted by the requestor. When a service is invoked, it puts the results from the service in a new container inside the initial container—the new container comprises the current scope. If another service is invoked while that container is open, the variables returned by the from the service are put into another container, which is placed inside the previous container, and so forth.

Note: Besides the `%invoke%` tag, certain other tags (e.g., `%loop%`) implicitly switch scope. This behavior is noted in the tag descriptions in "Tag Descriptions" on page 45.

Why Does Scope Matter?

Understanding (and controlling) the scope within a DSP is important for two reasons:

- Only those variables that are within the current scope are passed to a service that you invoke in a DSP. So, to ensure that a service receives all the variables it needs at run time, you must make sure that all those variables are within the scope. For more information about passing parameters to and within a DSP, see "Passing Parameters with a DSP" on page 24.
- Only those variables that are within scope can be addressed without qualifiers. (See "Referencing Variables In and Out of Scope" on page 23.) Moreover, except for the initial scope (whose variables exist for the entire life of a DSP), certain tags, such as the `%end%` tag, cause the current scope to close *and* discard the variables within it.

Ending the Scope of the Invoke Action

The `%end%` element in the `%invoke%`…`%end%` construct ends the scope for that invoke. It marks the point in the DSP where the variables associated with that construct are dropped and scope reverts to the previous level.

The following example shows a DSP that invokes two services in sequence. In this DSP, both services receive the variables from the *initial* scope as input. The two `%invoke%` blocks are highlighted in bold in the sample.

```
 <HTML>
 <HEAD>
 <title>Order Tracking System</title>
 </HEAD>
 <!--User passes in order number in param named oNum> -->
 <!--The initial scope contains only <oNum>     -->
%invoke orders:getShipInfo%
<!--Scope switches to results
of getShipInfo      -->
<!--You can reference variables
in the initial scope -->
<!--with a relative addressing
qualifier     -->
<p>
Order: %value /oNum%<br>
Date Shipped: %value shipDate%<br>
Carrier: %value carrier% %value
serviceLevel%
</p>
%endinvoke%
 <!--Scope reverts to the initial scope     -->
 <!--Results from getShipInfo have been discarded  -->
 <!--and cannot be accessed beyond this point    -->
%invoke orders:getCustInfo%
<!--Scope switches to results
of getCustInfo    -->
<!--You can reference variables
in the initial scope-->
<!--with a relative addressing
qualifier    -->
<table>
<tr><td>Company:</td>
  <td>%value companyName%</td></tr>
<tr><td>Phone:</td>
  <td>%value phoneNum%</td></tr>
<tr><td>Address:</td>
  <td>%value StreetAddr1%<br>
%value StreetAddr2%</td></tr>
<tr><td></td>     <td>%value
city%, %value state%</td></tr>
<tr><td></td>     <td>%value
postalCode%</td></tr>
</table>
%endinvoke%
 <!--Scope reverts to the initial scope     -->
 <!--Results from getCustInfo have been discarded  -->
 <!--and cannot be accessed beyond this point    -->
 </BODY>
 </HTML>
 <HTML>
 <HEAD>
 <title>Order Tracking System</title>
 </HEAD>
 <!--User passes in order number in param named <oNum> -->
 <!--The initial scope contains only <oNum>     -->
%invoke orders:getShipInfo%
<!--Scope switches to results
```

```
of getShipInfo      -->
<!--You can reference variables
in the initial scope -->
<!--with a relative addressing
qualifier     -->
<p>
Order: %value /oNum%<br>
Date Shipped: %value shipDate%<br>
Carrier: %value carrier% %value
serviceLevel%
</p>
%invoke orders:getCustInfo%
<!--Scope switches to results
of getCustInfo     -->
<!--You can reference variables
in the initial scope -->
<!--and prior scope with relative
addressing qualifiers -->
<table>
<tr><td>Company:</td>
 <td>%value companyName%</td></tr>
<tr><td>Phone:</td>
 <td>%value phoneNum%</td></tr><tr><td>Address:</td> <td>%valueStreetAddr1%<br>
%value StreetAddr2%</td></tr>
<tr><td></td>    <td>%value
city%, %value
state%</td></tr>
<tr><td></td>    <td>%value
postalCode%</td></tr>
</table>
%endinvoke%
<!--Scope reverts back to results
of getShipInfo     -->
<!--Results from getCustInfo
have been discarded     -->
<!--and cannot be accessed beyond
this point     -->
%endinvoke%
 <!--Scope reverts to the initial scope     -->
 <!--Results from getShipInfo have been discarded  -->
 <!--and cannot be accessed beyond this point     -->
 </BODY>
 </HTML>
```

Referencing Variables In and Out of Scope

You can refer to variables that are in the current scope directly—without any qualifiers. To reference a variable that is out of scope, you must use the following directory-like notation to describe its position relative to either the current scope or the initial scope.

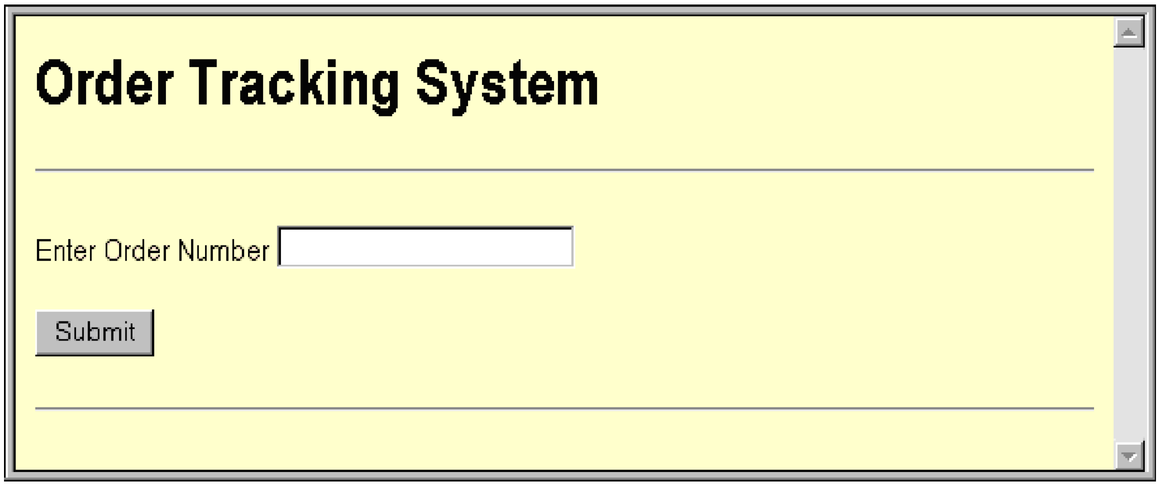
Use this notation...	To...
variableName	Reference a variable in the current scope. For example: `shipNum`.

Use this notation...	To...	
../variableName	Reference a variable one or more levels above the current scope. For example:	
	`../oNum`	One level above
	`../ ../oNum`	Two levels above
/variableName	Reference a variable in the initial scope. For example: `/oNum`.	
recName/ variableName	Reference a variable within a specific document. For example:	
	`buyerInfo/state`	Selects the *state* element from the document *buyerInfo* in the current scope
	`../buyerInfo/oNum`	Selects the *state* element from the document *buyerInfo* one level above current scope

Passing Parameters with a DSP

You use the standard HTTP "GET" and "POST" methods to pass input parameters to a DSP. In a browser-based client, you usually do this with an HTML form. For example, if you were creating an order-tracking application, you might create a form that prompts the user for an order number and invokes a DSP that returns the shipping status of that order.

You can use an HTML form to pass input to a DSP



Use an HTML <FORM> tag to invoke a DSP. Then, use HTML <INPUT> tags to pass input parameters to it.

```
 <HTML>
 <HEAD>
 <TITLE>Order Tracking System</TITLE>
 </HEAD>
 <BODY BGCOLOR="#FFFFCC">
 <H1>Shipping Information</H1>
 <HR>
<FORM ACTION="/ORDER_TRAK/getorderinfo.dsp"
METHOD="GET">
  <P>Enter Order Number  <INPUT TYPE="TEXT" NAME="oNum"> <BR>
  <INPUT TYPE="HIDDEN" NAME="action"
VALUE="shipinfo">  <BR>
  <INPUT TYPE="SUBMIT" VALUE="Submit">
  </FORM>
 <HR>
 </BODY>
```

When the DSP is invoked, its initial scope encompasses two parameters: *oNum* and *action* . Services that you invoke within this scope receive an IData object containing these two elements.

The following code shows the contents of the DSP (getorderinfo.dsp) invoked by the previous example. It uses the value in *action* to conditionally execute a section of the DSP that invokes the service. (For more information about using the `%switch%` tag to conditionally execute a section of code, see "Building Conditional Blocks with the %switch% Tag" on page 31.)

```
<HTML>
<HEAD>
<title>Order Tracking System</title>
</HEAD>
<BODY BGCOLOR="#FFFFCC">
<!--User passes in order number in param named <oNum> -->
<!--and requested action in <action>      -->
<H1>Order Tracking System</H1>
%switch action%
 %case 'shipinfo'%
```

```
  %invoke orders:getShipInfo%
  <p>
  Order: %value /oNum%<br>
  Date Shipped: %value shipDate%<br>
  Carrier: %value carrier% %value serviceLevel%
  </p>
  %endinvoke%
%case ‘orderinfo’%
  %invoke orders:getOrderInfo%
.
.
.
```

Passing Parameters Between DSPs

Because HTTP does not preserve variables from one request to another, to pass data from one DSP to another, you must explicitly set those values in the documents that you return to the requestor. For example, let’s say you want to allow your user to view or edit the shipment displayed by the order-tracking DSP above. To do this, you must return a document containing links to the DSPs that perform these tasks, and these DSPs will need the order number (*oNum*) that the user submitted on the original page. To pass the order number to these DSPs, you must put *oNum* in the page you return to the client.

The following example shows two ways in which you can build a DSP that will pass a variable to another DSP. The first portion of code in bold illustrates how you can pass a parameter in a hidden input element. The second portion of code in bold illustrates how you can pass a parameter as a name-value pair in a link to a DSP.

```
<HTML>
<HEAD>
<title>Order Tracking System</title>
</HEAD>
<BODY BGCOLOR="#FFFFCC">
<H1>Order Tracking System</H1>
<!--User passes in order number in param named <oNum> -->
<!--and requested action in <action>        -->
%switch action%
  %case ‘shipinfo’%
  %invoke orders:getShipInfo%
   <H2>Shipping Information for Order %value /oNum%</H2>
   <P>Date Shipped: %value shipDate%<BR>
   Carrier: %value carrier% %value serviceLevel%
   </P>
   <HR>
   %ifvar shipDate -isnotempty%
    <FORM ACTION="/ORDER_TRAK/editshipinfo.dsp" METHOD="get">
    <P><B>Change this Shipment:</B></P>
    <P><INPUT TYPE="RADIO" NAME="action" VALUE="edit">
     Edit Shipment Details</P>
    <P><INPUT TYPE="RADIO" NAME="action" VALUE="cancel">
     Cancel this shipment</P>
     <INPUT TYPE="SUBMIT" VALUE="Submit">
    <INPUT TYPE="HIDDEN" NAME="oNum" VALUE="%value /oNum">
    </FORM>
    <HR>
  %endifvar%
  <P><A HREF="/ORDER_TRAK/getorderinfo.dsp
    ?action=orderinfo&oNum=%value /oNum%">View Entire Order</A></P>
%endinvoke%
```

```
%case 'getorder'%
 %invoke orders:getOrderInfo%
 .
 .
 .
```

Passing Parameters Between Services within a DSP

To pass data between services within a DSP, you can use any of the following techniques:

- Invoke the service that needs the parameter within the scope of the service that produces the parameter. When a service is invoked in a DSP, it receives all of the variables that are "in scope" at the point where it is invoked. For an example of this, see the sample code on "Ending the Scope of the Invoke Action" on page 21.
- Use the `%rename%` tag to copy a variable that is out of scope into the current scope. For details and examples, see the `%rename%` tag description on "%rename%" on page 58.
- Use the `%scope%` tag to add a variable to the current scope and specify its value. For details and examples, see the `%scope%` tag description on "%scope%" on page 59.

Catching Errors

If you want your DSP to react in a specified way when the invoked service fails, include an `%onerror%` tag within your `%invoke%...%end%` construct. The code in the `%onerror %` block executes only if an exception occurs while the service executes or the service returns an error.

When the `%onerror%` block executes, the scope contains the following values:

Key...	Description
error	A String containing the Java class name of the exception that was thrown (e.g., com.wm.app.b2b.server.AccessException).
errorMessage	A String containing the exception message in English, regardless of the locale of the server or client.
localizedMessage	A String containing the exception message, translated into the language used by the client that invoked the DSP.
errorInput	The IData object that was passed to the invoked service.
errorOutput	The IData object returned by the invoked service. If the service returned an error, *errorOutput* will contain $error and any other variables that were in the pipeline when the service ended.

Key...	Description
	Important: If the service experiences an exception (i.e., the server is not able to execute it successfully), *errorOutput* will not be exist. This variable is only produced when the service returns an error.
errorService	The name of the invoked service.

The following example shows how you might use an `%onerror%` clause to return an error message to the user.

```
 .
 .
 .
 %invoke orders:getShipInfo%
  <H2>Shipping Details for Order %value /oNum%</H2>
  <P>Date Shipped: %value shipDate%<BR>
  .
  .
  .
 %onerror%
<HR>
<P><FONT COLOR="#FF0000">The
Server could not process your request
because the following error occurred.
Contact your server
administrator.</FONT></P>
<TABLE WIDTH="50%" BORDER="1">
<TR><TD><B>Service</B></TD><TD>%value
errorService%</TD></TR>
<TR><TD><B>Error</B></TD><TD>%value
Error% &nbsp;
%value errorMessage%</TD></TR>
</TABLE>
<HR>
 %endinvoke%
   .
   .
   .
```

Extracting Results from an Array Variable

If a service returns an array variable—such as a String list, a String table, or a document list—to a DSP, you use the `%loop%` tag with a *variableName* to extract values from the elements in the array.

When you use `%loop%` on a document variable, the scope within the loop block automatically changes to encompass just those elements within the specified document.

The following example shows a loop that extracts values from a document list (called *items*) that is returned by the service named orders:getOrderInfo.

```
 %invoke orders:getOrderInfo%
  <P>This shipment contains the following items</P>
  <TABLE WIDTH="90%" BORDER="1">
  <TR><TD>Number</TD><TD>Qty</TD><TD>Description</TD><TD>Status</TD></TR>
%loop items%
<TR>
<TD>%value stockNum%</TD>
<TD>%value qty%</TD>
<TD>%value description%</TD>
<TD>%value status%</TD>
</TR>
%endloop%
  </TABLE>
 %endinvoke%
  .
  .
  .
```

For additional information about the `%loop%` tag, see the `%loop%` tag description on "%loop%" on page 53.

Extracting Results from a Document

To extract results from a document (i.e., an IData object), you use the `%loop%` tag with the `-struct` option to execute a block of code once for each key in the structure.

The following example shows how you would extract values from each key in a document named *buyerInfo* .

```
 %invoke orders:getOrderInfo%
  <P>Buyer:</P><P>
%loop -struct buyerInfo%
%value%<BR>
%endloop%
  </TABLE>
   .
   .
   .
```

Using the %loop% Tag to Examine the Current Scope

If you use the `-struct` option without specifying the name of a document, the loop executes once for each element in the current scope. During testing and debugging, you may want to use this technique to examine the variables and their values at a particular point in the DSP. The following example shows the code you would use to display the name of each key and its contents in the current scope.

```
   .
   .
   .
<P>
%loop -struct%
%value $key% %value%<BR>
%endloop%
 </P>
   .
   .
   .
```

Conditionally Executing Blocks of Code

There are two tags you can use to conditionally execute code in a DSP: `%ifvar%` and `%switch%`. Both tags selectively execute a block of code based on the existence or value of a variable at run time.

Building Conditional Blocks With the %ifvar% Tag

The `%ifvar%` tag is similar to an "if…then…else" expression in other programming languages. You use it to denote a block of code that is to be executed only when a specified variable exists or contains a value that you specify.

The basic format of the `%ifvar%` tag is as follows, where *variableName* specifies the name of the variable that will be evaluated at run time:

```
%ifvar variableName%
Block of Code
[%else%
Block of Code]
%end%
```

Example

```
   .
   .
   .
 <!--Check for presence of backordered items in the order -->
 <!--and display if they exist                 -->
%ifvar backItems%
<p>Backordered Items
%loop backItems%
%value%<BR>
%endloop%
%endifvar%
   .
   .
   .
```

Testing for a Particular Value

In the preceding example, the enclosed block of code executes if a variable named *backItems* exists in the current scope. To test for the content of a variable, you can apply the following options to the `%ifvar%` tag.

Use this option...	To...
`-isnull`	Test whether the specified variable exists and is null.
`-notempty`	Test whether the specified variable contains a value (i.e., the value is not null or empty).

Use this option...	To...
`equals('anyString')`	Test whether the specified variable contains a specific value. (*variableName* must be a String variable to use this option.)
`vequals(refVariable)`	Test whether the value of the specified variable matches the contents of another variable in the pipeline.

The following example shows an `%ifvar%...%else%...%end%` construct that executes one of two blocks, depending on the contents of a variable named *clubMember*.

```
  .
  .
  .
 %invoke orders:getShipInfo%
  <H2>Shipping Details for Order %value /oNum%</H2>
  <P>Date Shipped: %value shipDate%<BR>
   .
   .
   .
%ifvar clubMember equals('Y')%
%include membershipterms.txt%
%else%
%include nonmembershipterms.txt%
%endifvar%
 %endinvoke%
   .
   .
   .
```

For additional information about the `%ifvar%` tag, see the `%ifvar%` description on "%ifvar%" on page 49.

Building Conditional Blocks with the %switch% Tag

You can use the `%switch%` tag to construct a conditional expression based on the value of a specified variable. The `%switch%` tag allows you to define a separate block of code (a case) for each value that a variable can take at run time. You use this tag instead of `%ifvar%` when you have more than two possible paths of execution. (You can also handle multiple paths of execution by building multiple `%ifvar%` expressions, however, the `%switch%` tag is much easier to use and maintain for this purpose.)

The basic format of the `%switch%` tag is as follows, where *variableName* specifies the name of the variable you want to evaluate at run time and *switchValue* is the value that will cause a case to execute:

```
%switch variableName%
 %case 'switchValue'%
Block of Code
 %case 'switchValue'%
  Block of Code
  .
  .
  .
[%case%
Default Block
of Code ]
%end%
```

At run time, the DSP Processor evaluates each `%case%` block in order, and executes the first block whose *switchValue* matches the value in *variableName*. After processing the code within that block, the DSP Processor skips the remaining cases in the `%switch%` construct.

The following example shows an `%switch%` construct that executes one of two cases, depending on the contents of a variable named *action*.

```
   .
   .
   .
%swtich action%
 %case 'shipinfo'%
  %invoke orders:getShipInfo%
    .
    .
    .
   %endinvoke%
  %case 'vieworder'%
   %invoke orders:getOrderInfo%
    .
    .
    .
  %endinvoke%
%endswitch%
```

Specifying a Default Case

If you want to specify a block of code that executes when all other cases are not true, include a case without a *switchValue*. The following example illustrates how to create a default case by omitting *switchValue* and putting the case at the end of the `%switch%` construct.

```
   .
   .
   .
 %switch acctType%
  %case 'Platinum'%
   %include platorderform.html%
  %case 'Gold'%
%include goldform.html%
%case%
   %include regorderform.html%
```

If you include a default case, you must put it at the end of the switch construct. For additional information about using the `%switch%` tag, see the `%switch%` tag description on "%switch%" on page 62.

Inserting Text Files in a DSP

The `%include%` tag allows you to insert a text file in a DSP. When you use the `%include %` tag, the DSP Processor inserts the specified file and evaluates its contents (and processes any tags that it contains) from top to bottom at run time.

The basic format of the `%include%` tag is as follows, where *fileName* specifies the name of the file that you want to insert into the DSP. (If *fileName* is not in the same directory as the DSP, you must specify its path relative to the DSP as shown by the example.)

```
%include fileName%
```

Example

```
  .
  .
  .
%switch acctType%
 %case 'Platinum'%
%include forms\platorderform.txt%
 %case 'Gold'%
%include forms\goldorderform.txt%
 %case%
%include forms\regorderform.txt%
%endswitch%
```

When you insert a file, it inherits the scope that is current at the point where you call it. For additional information about the `%include%` tag, see the `%include%` tag description on "%include%" on page 50.

Specifying a List of Permitted URLs for Redirection

Use the `%validConst%` tag to specify external URLs that Integration Server will allow for redirection from the page. The following example shows how to use the `%validConst%` tag to specify the URL "http://example.com" as a valid URL for redirection.

Example

```
  .
  .
  .
%validConst url(http://example.com)%
%endvalidConst%
  .
  .
  .
```

If you want to allow more than one URL for redirection, you can specify a comma-separated list of URLs. The following example shows how to use the `%validConst %` tag to specify the URLs "http://example.com", "http://example.org", and "http:// example.net" are valid for redirection.

Example

```
  .
  .
  .
%validConst url(http://example.com,http://example.org,http://example.net)%
%endvalidConst%
  .
  .
  .
```

Arbitrarily Processing DSP Tags

Integration Server automatically processes DSP tags when a DSP is requested via HTTP. You can also process the tags in a string of text at any arbitrary point during a service using the services in the pub.report folder. When you use these services, tags are processed against the variables that are in the pipeline at run time.

For information about using the services in the pub.report folder, see *webMethods Integration Server Built-In Services Reference*.

2 Using Output Templates to Format Service Output

What is an Output Template?

Output templates allow you to insert output values from a service into a document that you define. They work much like server-side includes (SSIs) in that they contain special “tags” that webMethods Integration Server processes before passing the document back to the client.

Output templates are similar to DSPs. In fact, the tags you use to compose DSPs are the same ones you use to compose output templates. However, DSPs have one additional tag—the `%invoke%` tag—that distinguishes them from output templates. This tag allows a DSP to invoke a service.

Output templates are used most frequently to customize the HTML page that a service returns to a browser-based application. However, you can also use them to generate other types of documents. For example, if you have a service that retrieves a record from a relational database, you might use an output template to format your results as an XML document or a comma-delimited record before returning it to the requestor.

You can also use output templates to return WML or HDML content to wireless devices, such as an Internet-enabled wireless phone or an Internet-enabled personal digital assistant.

What Does an Output Template Look Like?

A template is simply a String containing text and one or more *tags*. Tags are special commands, enclosed in % symbols, that cause webMethods Integration Server to perform a specified action—typically to insert the value of a variable—at a specified point in the String.

For example, if you have the following values in the pipeline:

Table 1. Output in the Pipeline

Variable Name	Value
CompanyName	Bitterroot Boards
AccountNum	BTB-9590651
ContactName	Lauren Cheung
PhoneNum	406-721-5000

You might create a template that looks as follows to return the values in an HTML document to a browser-based client. The strings in bold are template tags instructing the server to insert specified values from the pipeline at run time.

```
 <!--Output Template for an HTML Document-->
============================================
 <!DOCTYPE HTML PUBLIC>
 <HTML><HEAD><META HTTP-EQUIV="Content-Type" VALUE="text/html;chaset=UTF-
 8"><TITLE></TITLE></HEAD>
 <BODY><P>Contact information for account %value AccountNum% is:</P>
    <TABLE>
       <TR><TD><B>Account Name</B></TD><TD>%value CompanyName%</TD></TR>
       <TR><TD><B>Contact Name</B></TD><TD>%value ContactName%</TD></TR>
       <TR><TD><B>Phone Number</B></TD><TD>%value PhoneNum%</TD></TR>
    </TABLE>
 </BODY></HTML>
```

Output templates are not limited to HTML. You could create an output template that looks like this to return values in an XML document:

```
 <!--Output Template for an XML Document-->
==========================================
 <?xml version="1.0"?>
 <ACCOUNT_INFO>
     <ACCOUNT_NAME>%value AccountNum%</ACCOUNT_NAME>
     <COMPANY_NAME>%value CompanyName%</COMPANY_NAME>
     <CONTACT_NAME>%value ContactName%</CONTACT_NAME>
     <PHONE_NUM>%value PhoneNum%</PHONE_NUM>
 </ACCOUNT_INFO>
```

Or an output template that looks like this to return values as a comma-separated record:

```
 <!--Output Template for a Comma-Separated
Values Record-->
==========================================================
%value AccountNum%,%value CompanyName%,%value
ContactName%,%value PhoneNum%
```

When Does the Server Use an Output Template?

The server applies output templates to the results of services that are invoked by HTTP, FTP, or SMTP clients (i.e., requests that come through the HTTP, HTTPS, FTP, or SMTP listeners). You can also arbitrarily apply output templates to the pipeline using the built-in services in the pub.report folder.

Using Output Templates to Return Output to HTTP Clients

If a service has an output template assigned to it, the server automatically applies the template to the results of the service (i.e., the contents of the pipeline) any time that service is externally invoked by an HTTP client. (If a service does not have an output template, the server simply returns the results of the service in the body of an HTML document, formatted as a two-column table.)

Guidelines for Using HTML-Based Output Templates with HTTP Clients

If you want to use an HTML-based output template to return output to an HTTP client, keep the following points in mind:

- Make sure the output template produces a valid HTML document.
- If the client checks the Content-Type value in the HTTP response header, make sure the template contains a <meta> tag that sets the Content Type to “text/html”.

Guidelines for Using XML-Based Output Templates with HTTP Clients

If you want to use an XML-based output template to return output to an HTTP client, keep the following points in mind:

- Make sure the output template generates a valid and well-formed XML document.
- Make sure the template contains a <meta> tag that sets the Content Type to “text/xml”.
- If your client is a browser, make sure it can accept and display XML (for example, Microsoft Internet Explorer 5.0 can display XML) or that the client machine has a MIME definition for Content-Type “text/xml”.

Using Output Templates to Return Output to SMTP and FTP Clients

Besides HTTP clients, the server also applies output templates to the results it returns to FTP and email clients. However, be aware that:

- For e-mail clients, the server can apply either XML- or HTML-based output templates.
- For FTP clients, the server will only apply XML-based output templates. If an HTML-based output template is assigned to the service, it is ignored.

Using Output Templates to Return Output to Wireless Devices

Integration Server can use WML (Wireless Markup Language) output templates or HDML (Handheld Device Markup Language) output templates to return output to Internet-enabled wireless devices. You might want to return service output in WML and HDML if you allow the use of wireless devices to invoke services on Integration Server. For example, if you allow clients to submit purchase orders using a wireless device, you would want to be able to send their order confirmation to the wireless device.

For information about how Integration Server communicates with a wireless device, see *webMethods Integration Server Administrator’s Guide*.

Guidelines for Creating WML Output Templates

If you want to use a WML output template to return output to an Internet-enabled wireless device, such as a wireless phone, keep the following points in mind:

- The output template needs to produce a valid and well-formed WML document.
- Make sure the template contains a <meta> tag that sets the Content Type to "text/vnd.wap.wml".
- Wireless devices have small screen dimensions.
- The browser on the wireless device needs to support WML 1.1 or higher to receive a WML output template from Integration Server.
- Different types of web browsers on wireless devices may display WML pages differently.
- Some web browsers for wireless devices place a limitation on the length of a URL: name in WML pages. Make sure that you create WML pages that are compliant with the browser requirements.

For more information about WML, see "http://www.oasis-open.org/cover/".

Guidelines for Creating HDML Output Templates

If you want to use an HDML output template to return output to an Internet-enabled wireless device, such as a personal digital assistant, keep the following points in mind:

- The output template needs to produce a valid HDML document.
- Make sure the template contains a <meta> tag that sets the Content Type to "text/x-hdml".
- Wireless devices have small screen dimensions.
- The browser on the wireless device needs to support HDML 3.0 or higher to receive an HDML output template from Integration Server.

For more information about HDML, see "http://www.Phone.com".

Applying Output Templates Arbitrarily

You can arbitrarily apply output templates to the contents of the pipeline using the built-in services in the pub.report folder. For information about using these services, see *webMethods Integration Server Built-In Services Reference*.

Creating an Output Template

You can use Designer to create an output template, or you can create an output template file using an ordinary text editor.

When you create an output template, keep the following points in mind:

- You must give the output template file a name that is unique within the package in which it resides.
- If you plan to assign the output template to a service, the template file must reside in the \templates directory of the package where the service resides (i.e., *Integration Server_directory*\packages*packageName* \templates).
- If you specify a file encoding other than UTF-8 in the <meta> tag of your template's content, the characters that you use in your template (including the data inserted into your template using %VALUE% statements) are limited to those in the character set of the encoding you choose.

To create the contents of an output template file, type all literal text exactly as you want it to appear in the result and then embed any of the following tags where you want the server to execute them at run time. For a complete description of each tag, see "Tag Descriptions" on page 45.

Important: These tags are case sensitive and must be typed exactly as shown in this document. Additionally, all text between %…% symbols in a tag must appear on one line (i.e., a tag cannot contain line breaks).

Use this tag...	To...
`%value` *`VarName`*`%`	Insert the value of the specified variable into the output string.
`%scope` *`DocumentName`*`%` . . . `%end%`	Limit the scope for the enclosed block of code to those elements in the specified document.
`%loop` *`VarName`*`%` . . . `%end%`	Repeat the enclosed block of code once for each element in the a specified array variable.
`%ifvar` *`VarName`*`%` . . . `%end%`	Conditionally include the enclosed block of code if a variable exists or matches a specified value.
`%switch` *`VarName`*`%` `%case` *`Value1`*`%` `%case`*`Value2`*`%` `%case` *`Value3`*`%` . . . `%end%`	Conditionally include a block of code depending on the value of a specified variable.

Use this tag...	To...
`%include`*`TemplateName`*`%`	Insert and execute a specified output template.

`%validConst url`*`url1,url2,...,urln)`*` %`
`%endvalidConst%`
	Limit the external URLs that can be accessed via redirection.

`%nl%`	Insert new line in the output string.

`%comment%`
`.`
`.`
`.`
`%end%`
	Omit the enclosed block of text from the output.

`%rename `*`VarName NewVarName`*`%`	Change the name of a variable for the purpose of referencing it in the output template. This may be necessary if you include predefined output templates that use variable names that are different from those in the pipeline.

For steps for creating an output template using Designer, see *webMethods Service Development Help*.

Assigning an Output Template to a Service

Use of an output template to format service output is optional. You assign an output template to a service using the Properties view in Designer.

When using output templates, keep the following points in mind:

- A service can have at most one output template assigned to it at a time. You can dynamically change the output template assignment at run time, however. For more information, see "Applying Output Templates Arbitrarily" on page 39.
- You can assign the same output template to more than one service.
- If you assign an existing output template to a service, the output template must reside in the *Integration Server_directory*\packages\packageName \templates directory, where *packageName* is the package in which the service is located.

  Note: If you assign an output template to a service and later copy that service to a different package, you must copy the output template file to the *Integration Server_directory*\packages\packageName \templates directory of the new package. (If you copy an entire package, any output templates will be included automatically.) If the template file has a file extension other than .html, rename the file extension as ".html" so that Designer will recognize its contents.

- The server treats the case of the file name differently depending on which operating system you are using. For example, on a case-insensitive system such as Windows, the server would see the names "template" and "TEMPLATE" as the same name. However, on a case-sensitive system such as UNIX, the server would see these as two different names. If you are trying to assign an existing output template and you enter a file name in the wrong case on a UNIX system, the wrong file name could be assigned as the output template for your service.

For steps for assigning an output template to a service, see *webMethods Service Development Help*.

Securing Pages and Documents Created from Output Templates

You can secure the documents and pages created from output templates against cross site scripting (XSS) attacks. You can also limit the external URLs to which HTML pages created from an output template can be redirected.

Securing Documents and HTML Pages Against Cross Scripting Attacks

If you use the `%value` *`Variable`*`%` tag in output templates, the output from the tag in the resulting documents or HTML pages created from the output templates might be vulnerable to cross site scripting (XSS) attacks. To prevent these cross site scripting attacks, set the watt.core.template.enableFilterHtml parameter to `true` (the default). When this parameter is `true`, the output from a `%value` *`Variable`*`%` tag, including XML and JavaScript, is HTML encoded.

When the watt.core.template.enableFilterHtml parameter is set to `true`, if you do not want Integration Server to HTML encode the output from a `%value` *`Variable`*`%` tag, you can use the `encode(none)` option of the `%value` *`Variable`*`%` tag, (`%value` *`Variable`* `encode(none)%`).

If you do not want Integration Server to HTML encode the output from any `%value` *`Variable`*`%` tag in all documents and/or HTML pages resulting from output templates, set the watt.core.template.enableFilterHtml parameter to `false`. Setting the watt.core.template.enableFilterHtml parameter to `false` does not override settings of the `%value` *`Variable`*`%` tag's `encode` option.

Important: If you use `encode(none)` so that the output from a `%value` *`Variable`*`%` tag is not HTML encoded, that value is vulnerable to cross site scripting attacks. If you set the watt.core.template.enableFilterHtml parameter to `false`, all documents and pages resulting from output templates that use the `%value` *`Variable`*`%` tag are vulnerable to cross site scripting attacks.

For more information about the encode(none) option, see "%value%" on page 66.
For more information about the watt.core.template.enableFilterHtml parameter, see *webMethods Integration Server Administrator's Guide*.

Limiting the External URLs that Can Be Used for Redirection

Use the `%validConst%` tag in output templates to specify a list of URLs to which an HTML page resulting from the output template can be redirected. By specifying the `%validConst%` tag, the page can only be redirected to the URLs you specify.

For more information about how to use this tag, see "Specifying a List of Permitted URLs for Redirection" on page 33.

A Tag Descriptions

Overview

This appendix describes the tags you use to construct Dynamic Server Pages (DSPs) and output templates.

Important: Tags are case sensitive and must be typed into a template or DSP exactly as shown in this appendix (e.g., %loop%, not %LOOP% or %Loop%).

Important: All text between %…% symbols in a tag must appear on one line (i.e., no line breaks).

The examples shown in this appendix assume a pipeline that looks as follows:

Table 2. Contents of the Pipeline

Key	Value	
submittor	`Mark Asante`	
shipNum	`991015-00104`	
shipDate	`10/15/99`	
carrier	`UPS`	
serviceLevel	`Ground`	
arrivalDate	`10/18/99`	
items	**Key**	**Value**
	qty	`10`
	stockNum	`BK-XS160`
	description	`Extreme Spline 160 Snowboard-Black`
	orderNum	`GSG-99401088`
	status	`Partial Order`
	qty	`15`

Key	**Value**	
	stockNum	`WT-XS160`
	description	`Extreme Spline 160 Snowboard-White`
	orderNum	`GSG-99401088`
	status	`Complete`
supplierInfo	**Key**	**Value**
	companyName	`Bitterroot Boards, LLC`
	streetAddr1	`1290 Antelope Drive`
	streetAddr2	
	city	`Missoula`
	state	`MT`
	postalCode	`59801`
	supplierID	`BRB-950817-001`
	phoneNum	`406-721-5000`
	faxNum	`406-721-5001`
	email	`Shipping@BitterrootBoards.com`
buyerInfo	**Key**	**Value**
	companyName	`Global Sporting Goods, Inc.`
	accountNum	
	phoneNum	`(216)741-7566`
	faxNum	`(216)741-7533`

Key	**Value**	
	streetAddr1	`10211 Brookpark Road`
	streetAddr2	
	city	`Cleveland`
	state	`OH`
	postalCode	`22130`
	email	`Receiving@GSG.com`
backItems	`SL-XS170 Extreme Spline 170 Snowboard- Silver`	
	`BL-KZ111 Kazoo 111 Junior Board- Blue`	
	`BL-KZ121 Kazoo 121 Junior Board-Blue`	

%comment%

You use the `%comment%` tag to include remarks in your code. At run time, the server ignores all text (and tags) between the`%comment%` and `%end%` tag.

Syntax

```
%comment%
Block of Code
%end%
```

Effect on Scope

None

Examples

The following example contains a section of explanatory information.

```
%comment%
Use this template to generate an order list from any document containing
a purchased item number, quantity, description, and PO number
%end%
<tr>
<td>%value stockNum%</td>
<td>%value qty%</td>
<td>%value description%</td>
<td>%value orderNum%</td>
</tr>
```

%ifvar%

You use the `%ifvar%` tag to conditionally include or exclude a block of code based on the existence or value of a specified variable.

Syntax

```
%ifvar Variable [option option
option...] %
Block of Code
[%else%
Block of Code ]
%end%
```

Arguments

Variable specifies the name of the variable that determines whether or not the enclosed block of code is processed.

Options

You can use any of the following options with this tag. To specify multiple options, separate them with spaces.

Options	Description
`-isnull`	Includes the enclosed block of code only if *Variable* is null. For example: `%ifvar backItems -isnull%`.
`-notempty`	Includes the enclosed block of code only if *Variable* contains one or more characters (for string variables only). For example: `%ifvar supplierInfo/email -notempty%`.
`equals('`*AnyString*`')`	Includes the enclosed block of code only if the value of *Variable* matches the string you specify in *AnyString* . (*AnyString* is case sensitive. “FedEx” does not match “Fedex” or “FEDEX”). For example: `%ifvar carrier equals ('FedEx')%`.
`vequals(`*RefVar*`)`	Includes the enclosed block of code only if the value of *Variable* matches the value of the variable you specify in *RefVar* . For example: `%ifvar supplierInfo/state vequals(buyerInfo/state)%`.

Options	Description
`matches('regular_exp')`	Specifies that the condition is true if the value of *Variable* matches the regular expression *regular_exp* . For example: `%ifvar carrier matches('UPS*')%`.

Effect on Scope

None

Notes

For readability, you can optionally use `%endif%` or `%endifvar%` instead of `%end%` to denote the end of the `%ifvar%` block.

Examples

The following example inserts a paragraph if a variable named *AuthCode* exists.

```
.
.
.
%ifvar AuthCode%
  <p>Authorization Code Received %value $date%: %value AuthCode</p>
%endif%
.
.
.
```

The following example generates a line for each element in the *backItems* String list only if *backItems* exists; otherwise, it prints a standard message.

```
.
.
.
%ifvar backitems%
  <p>The following items are backordered</p>
  <p>
  %loop backItems%
   %value%<BR>
  %endloop%
</p>
%else%
  <p>There are no backordered items pending for your account</p>
%end%
.
.
.
```

%include%

You use the `%include%` tag to reference a text file. When you `%include%` a text file, the server inserts the contents of the specified file (processing any tags it contains) at run time. If you use template and/or DSPs extensively, you may want to build a library of standard “code fragments” that you reference using `%include%` tags as needed.

Syntax

`%include` *`FileName`*`%`

Arguments

FileName specifies the name of the text file that you want to insert. If the text file is not in the same directory as the template or DSP that references it, *FileName* must specify its path relative to the template or DSP file.

Effect on Scope

None. If the inserted file contains tags, they inherit the scope that is in effect at the point where the `%include%` tag appears.

Examples

The following example inserts a file called “TMPL_ShipToBlock.html” into the code. Because path information is not provided, the server expects to find this file in the same directory as the file containing the template or DSP.

```
.
.
.
%scope buyerInfo%
  <p>Shipped To:<br>
   %include TMPL_ShipToBlock.html%</p>
%end%
.
.
.
```

The following example inserts a file called “TMPL_ShipToBlock.html” into the code. At run time, the server expects to find this file in a subdirectory called “StandardDSPs” in the directory where the template or DSP resides.

```
.
.
.
%scope buyerInfo%
  <p>Shipped To:<br>
   %include StandardDSPs/TMPL_ShipToBlock.html%</p>
%end%
.
.
.
```

The following example inserts a file called “TMPL_ShipToBlock.html” into the code. At run time, the server expects to find this file in the template or DSPs parent directory.

```
.
.
.
%scope buyerInfo%
  <p>Shipped To:<br>
   %include ../TMPL_ShipToBlock.html%</p>
```

```
%end%
.
.
.
```

%invoke%

You use the `%invoke%` tag to execute a service from a DSP. When you use this tag, the server executes the specified service at run time and returns the results of the service to the DSP. You may optionally include the `%onerror%` tag within the `%invoke%` block to define a block of code that executes if an exception occurs while the service executes or the service returns an error.

This tag can only be used in DSPs. It cannot be used in an output template.

Syntax

```
%invoke serviceName %
Block of Code
[%onerror%
Block of Code ]
%end%
```

Arguments

serviceName specifies the fully qualified name of the service that you want to invoke.

Effect on Scope

Within an `%invoke%` block, the scope switches to the results of the service. If the service fails, the scope within the `%onerror%` block contains the following:

Key	Description
error	A String containing the Java class name of the exception that was thrown (e.g., com.wm.app.b2b.server.AccessException).
errorMessage	A String containing the exception message.
errorInput	The IData object that was passed to the invoked service.
errorOutput	The IData object containing the output returned by the invoked service. If the service returned an error, *errorOutput* will contain *$error* . If the service experienced an exception, *errorOutput* will not exist.
errorService	The name of the invoked service.

Examples

The following example invokes a service that returns shipping information and a form allowing the user to optionally edit or cancel an order. This example also includes a `%onerror%` block that displays error information if the service fails.

```
%invoke orders:getShipInfo%
  <H2>Shipping Details for Order %value /oNum%</H2>
  <P>Date Shipped: %value shipDate%<BR>
  Carrier: %value carrier% %value serviceLevel%
  </P>
  <HR>
  %ifvar shipDate -isnotempty%
    <FORM ACTION="http://rubicon:5555/orders/editShipInfo.dsp" METHOD="get">
    <P><B>Change this Shipment:</B></P>
    <P><INPUT TYPE="RADIO" NAME="action" VALUE="edit">
       Edit Shipment Details</P>
    <P><INPUT TYPE="RADIO" NAME=action" VALUE="cancel">
       Cancel this shipment</P>
       <INPUT TYPE="SUBMIT" VALUE="Submit">
    <INPUT TYPE="HIDDEN" NAME="oNum" VALUE="%value /oNum">
    </FORM>
    <HR>
  %endifvar%
  <P><A HREF="http://rubicon:5555/orders/getorder.dsp
      ?action=showorder&oNum=%value /oNum%">View Entire Order</A></P>
%onerror%
  <HR>
  <P><FONT COLOR="#FF0000">The Server could not process your request
  because the following error occurred. Contact your server
  administrator.</FONT></P>
  <TABLE WIDTH="50%" BORDER="1">
  <TR><TD><B>Service</B></TD><TD>%value errorService%</TD></TR>
  <TR><TD><B>Error</B></TD><TD>%value error% &nbsp;
              %value errorMessage%</TD></TR>
  </TABLE>
  <HR>
%endinvoke%
    .
    .
    .
```

%loop%

You use the `%loop%` tag to repeat a block of code once for each element in a specified array (String list or document list) or for each key in a document.

Syntax

```
%loop [Variable] [option option
option...]%
Block of Code
%end%
```

Arguments

Variable specifies the name of the array variable over which you want the enclosed section of code to iterate.

- You may optionally omit *Variable* and specify the `-struct` option to loop over each element in the current scope.
- When looping against a set of complex elements (e.g., documents, document lists, String lists) in a document, you can optionally use the #$key keyword to specify *Variable* instead of explicitly specifying a key name. When you use #$key in place of an explicit key name, it indicates that you want to apply the body of the loop to (i.e., switch the scope to) the current key. This allows you to process the contents of a document whose key names are not known. It is most often used with the `-struct` option to process a set of documents contained within another document. For an example of how this option is used, see the examples, below.

If *Variable* is a...	The loop is applied to...
String list	Each String in the list.
Document list	Each document in the document list.
Document	Each key in the document. When you use `%loop%` to process the elements of a document, you must also use the `-struct` option in the `%loop%` tag.

Options

You can use any of the following options with this tag. To specify multiple options, separate the options with spaces.

Options	Description
`-struct`	Specifies that *Variable* is a document and instructs the server to apply the loop once to each key in that document. When you use the –struct option, you can use the *$key* variable to retrieve the name of each element in the document. See examples, below.
`-eol`	Ends the body of the loop at the next end-of-line (EOL) character in the code. When you use –eol, you can omit the `%end%` tag.
`-$index`	Returns the current index number in an array. You can use it within a loop to obtain the index number upon which the loop is acting during each iteration.

Effect on Scope

If *Variable* is a document list, scope switches to the document on which the loop is executing.

Notes

Omit the variable name from the `%value%` tag if it is used in the body of a loop for a String table or a document. When variable name is omitted, the server inserts the value of the current element at run time.

For readability, you can optionally use `%endloop%` instead of `%end%` to denote the end of a `%loop%` block.

Examples

The following example generates a paragraph for each document in the *items* document list.

```
.
.
.
<p>This shipment contains the following items:</p>
%loop items%
  <p>%value qty% %value stockNum% %value description% %value status%</p>
%end%
.
.
.
```

The following example generates a line for each element in the *backItems* String list.

```
.
.
.
<p>The following items are backordered</p>
<p>
%loop backItems%
  %value%<BR>
%end%
</p>
.
.
.
```

The following example shows how you can nest `%loop%` tags to process the individual document elements in a document list.

```
.
.
.
<This shipment contains the following items:</p>
<table>
%loop items%
<tr>
  %loop -struct%
  <td>%value%</td>
  %end%
</tr>
%end%
```

```
.
.
.
```

The following example shows how you can use the `%loop%` tag to dump the contents (key names and values) of the current scope.

```
.
.
.
%loop -struct%
  %value $key% %value%<BR>
%end%
.
.
.
```

The following example shows how you use the #$key option to loop over the individual elements in a collection of documents contained within another document.

```
This example assumes that the service returns a document named MatchingAddress
that holds a set of documents (of unknown name and quantity) containing
address information.
.
.
.
<p>The following addresses were returned:</p>
%loop -struct MatchingAddresses%
 %loop -struct #$key%
   <p>
   %value streetAddr1%
   %value streetAddr2%
   %value city%, %value state% %value postalCode%
  </p>
 %end%
%end%
.
.
.
```

The following example shows how to obtain the iteration number of the loop execution and insert it into the output:

```
  indices = {%loop arrayA%%value $index%%loopsep ','%%endloop%}
```

When `ArrayA` has three elements, the output will be:

```
  indices = {0,1,2}
```

%loopsep%

You use the `%loopsep%` tag to insert a specified character sequence between the results from a `%loop%` block.

Syntax

`%loopsep 'sepString'%`

Arguments

sepString is a string that you want to insert between each result.

Effect on Scope

None

Notes

`%loopsep%` does not insert *sepString* after the result produced by the last iteration of the loop.

Examples

The following example inserts a comma between each value produced by the loop.

```
.
.
.
%loop items%
   %loop -struct%
     %value%
     %loopsep ‘,’%
   %endloop%
%endloop%
.
.
.
```

%nl%

You use the `%nl%` tag to generate a new line character in the code. The tag is useful when you want to preserve the ending of a line that ends in a tag. If you do not explicitly insert a `%nl%` tag on such lines, the server drops the new line character following that tag. (Note that this tag does not insert the HTML line break <BR> code. It merely inserts a line break character, which is treated as white space.) The main reason you use this tag is to preserve the format of the underlying code in a DSP, which can make it easier to read during debugging.

Syntax

`%nl%`

Effect on Scope

None

Examples

The following example shows how the `%nl%` tag is used to preserve the line endings on lines occupied by the three `%value%` tags. If the `%nl%` tag did not appear in this code, the three lines would be concatenated in the HTML document generated by the server.

```
.
.
.
<hr>
<p>Shipping Info:
%value carrier%%nl%
%value serviceLevel%%nl%
%value arrivalDate%%nl%</p>
<hr>
.
.
.
```

%rename%

You use the `%rename%` tag to move or copy a variable in the pipeline.

Syntax

`%rename` *`SourceVar TargetVar [option option option...]`* `%`

Arguments

SourceVar is the name of the variable that you want to move or copy. *SourceVar* can reside in any existing scope or document.

TargetVar specifies the name of the variable to which you want *SourceVar* moved or copied. *TargetVar* must be in the current scope. If *TargetVar* does not exist, it will be created. If *TargetVar* already exists, it will be overwritten.

Options

You can use the following option with this tag.

Option	Description
`-copy`	Copies *SourceVar* to *TargetVar* instead of moving it to *TargetVar*. If you do not use -copy, *SourceVar* is deleted after its contents are copied to *TargetVar*.

Effect on Scope

Does not cause scope to switch to a different level, but, depending on how you use this tag, it may alter the contents of the current scope.

Examples

The following example renames the *state* variable in the current scope.

```
.
.
.
%scope buyerInfo%
%rename state ST%
   %invoke TMPL_ShipToBlock.html%</p>
%end%
.
.
.
```

The following example copies the variable named *oNum* from the previous scope into the current scope.

```
.
.
.
%invoke orders:getCustInfo%</p>
 %rename ../oNum oNum -copy%
 %invoke orders:getOrderDetails%</p>
.
.
.
```

%scope%

You use the `%scope%` tag to restrict a specified block of code to a particular document in the pipeline. You can also use the `%scope%` tag to define a completely new document and switch the scope to that document.

Syntax

```
%scope [DocumentName] [option
option option...]%
Block of Code
%end%
```

Arguments

DocumentName specifies the name of a document within the current scope. If you do not specify *DocumentName*, the param and rparam options will extend the current scope. If you specify a new *DocumentName*, the scope switches to that document.

Options

You can use any of the following options to add elements to the scope specified by *DocumentName*. When you specify multiple options, separate them with spaces.

Important: If you set the value of an existing variable with these options, the value you specify will replace the variable's current value.

Note: For space reasons, the `%scope%` tag is shown on multiple lines in some of the examples below. Be aware that when you use the `%scope%` tag in a template or DSP, *the entire tag must appear on one line.*

Option	Description

`param(`*`Name='Value'`*`)`	Defines a new String or String array with the name you specify in *Name* and assigns to it the string you specify in *Value*.
	
	If *Name* is a String, specify one *Value* and enclose it in single quotes. For example:
	
	`%scope param(buyerClass='Gold')%`
	
	If *Name* is a String array:
	
	■ Include a set of empty brackets at the end of the name to indicate that you are defining an array.
	■ Enclose each element *Value* in single quotes.
	■ Separate elements with commas.
	
	For example:
	
	`%scope param(shipPoints[]='BWI','LAX',`
	`'ORD','MSP','DFW')%`

`rparam(`*`Name={Key='Value';`*
`Key='Value;Key='Value'}``)`
	Defines a new document or document list with the name you specify in *Name*, and assigns to it, values that you provide in a list of *Key*=*Value* pairs.
	
	If *Name* is a document:
	
	■ Enclose its list of elements in braces { }.
	■ Separate the elements with semicolons.
	■ Enclose *Value* strings in single quotes.
	
	For example:
	
	`%scope rparam(custServiceInfo=`
	`{csClass='Gold';csPhone='800-444-2300';`
	`csRep='Lauren Cheung'})%`
	
	If *Name* is a document list:
	
	■ Enclose each document in the list with braces { }.
	■ Separate documents with vertical bars `|`.
	■ Separate elements within each document with semicolons.

Option	Description
	■ Enclose *Value* strings in single quotes. For example: `%scope rparam(custServiceCtrs[]= {csName='Memphis';csPhone='800-444-2300';}| {csName='Troy';csPhone='800-444-3300';}| {csName='Austin';csPhone='800-444-4300';})%`

Effect on Scope

Switches scope to the specified document.

Notes

The specified scope remains in effect until the next `%scope%` tag is encountered (which declares a new scope) or the next, non-nested `%end%` tag is encountered (which reverts to the prior scope).

For readability, you may use `%endscope%` instead of `%end%` to denote the end of the `%scope%` structure.

Examples

The following example sets the scope to the document named *buyerInfo* .

```
.
.
.
%scope buyerInfo%
  <p>Shipped To:<br>
    %value companyName%<br>
    %value streetAddr1%<br>
    %value streetAddr2%<br>
    %value city%, %value state% %value postalCode%
%end%
.
.
.
```

The following example sets the scope to document *buyerInfo* and then uses the `param` option to add variables called *buyerClass* and *shipPoint* to that document.

```
.
.
.
%scope buyerInfo param(buyerClass=’Gold’) param(shipPoint=’BWI Hub’)%
  <p>Shipped To:<br>
    %value companyName%<br>
    %value streetAddr1%<br>
    %value streetAddr2%<br>
    %value city%, %value state% %value postalCode%</p>
<hr>
    <p>Point of Departure: %value shipPoint%<br>
    Customer Class: %value buyerClass%</p>
%end%
```

```
.
.
.
```

The following example sets the scope to document *buyerInfo* and then uses the `rparam` option to add a String variable named *req* to that scope before invoking a service.

```
.
.
.
<p>Open Orders:</p>
%scope buyerInfo rparam(req=openorders)%
%invoke orders:getOrderInfo%<br>
  %loop orders%
   Date: %value oDate%
   Number: <A HREF="showOrder.dsp&oNum=%value oNum%">%value oNum%</A>
   <br>
  %endloop%
<p>Click Order Number to View Details:</p>
%endscope%
.
.
.
```

%switch%

You use the `%switch%` tag to process one block from a series of predefined alternatives based on the value of a specified variable at run time.

Syntax

```
%switch Variable%
  %case 'SwitchValue'%
Block of Code
%case 'SwtichValue'%
Block of Code
        .
        .
        .
  [%case%
Default Block
of Code]
%end%
```

Arguments

Variable specifies the name of the variable whose value will determine which case is processed at run time.

SwitchValue is a string that specifies the value that will cause the associated case to be processed at run time.

Effect on Scope

None

Notes

To select a case, *SwitchValue* must match the value of *Variable* exactly. *SwitchValue* is case sensitive—"FedEx" does not match "Fedex" or "FEDEX".

The server evaluates `%case%` tags in the order that they appear in your code. When it finds a "true" case, it processes the associated block of code and then exits the `%switch%…%end%` structure.

You can specify a default case using the `%case%` tag without a *SwitchValue* . This case will be processed if *Variable* does not exist or if none of the other cases are true. The default case must appear as the last `%case%` in the `%switch%…%end%` structure.

For readability, you can optionally use `%endswitch%` instead of `%end%` to denote the end of the `%switch%` structure.

Examples

The following example inserts a specified paragraph, depending on the value in *carrier.*

```
.
.
.
%switch carrier%
 %case ‘FedEx’%
  <p>Shipped via Federal Express %value serviceLevel% %value trackNum%
 %case ‘UPS’%
  <p>Shipped via UPS %value serviceLevel%
 %case ‘Freight’%
  <p>Shipped via %transCompany%<br>
   FOB: %value buyerInfo/streetAddr1%<br>
   %value buyerInfo/streetAddr2%<br>
   %value city%, %value state% %postalCode%
%end%
</p>
.
.
.
```

The following example invokes different services based on the value of a variable named *action* .

```
<HTML>
<HEAD>
<title>Order Tracking System</title>
</HEAD>
<BODY BGCOLOR="#FFFFCC">
<H1>Order Tracking System</H1>
<HR>
%switch action%
 %case ‘shipinfo’%
  %invoke orders:getShipInfo%
    .
    .
 %case ‘showorder’%
  %invoke orders:getOrderInfo%
    .
    .
 %case ‘showinvoice’%
  %invoke orders:getInvoices%
```

```
   .
   .
%end%
<HR>
%include stdFooter.txt%
</BODY>
</HTML>
```

%sysvar%

You use the `%sysvar%` tag to insert the value of a special variable or server property into the document.

Syntax

`%sysvar` *`SystemVariable`*`%`

Arguments

SystemVariable is one of the following values, indicating which system variable or property you want to insert.

Value	Description
`host`	Inserts the name of the server that processed the DSP or template.
`date`	Inserts the current date. The date will appear in “Weekday Month Day HH:MM:SS Locale Year” format—e.g., `Fri Aug 12 04:15:30 Pacific 2007`.
`lastmod`	Inserts the date and time that this file was last modified. The date will appear in “Weekday Month Day HH:MM:SS Locale Year” format—e.g., `Fri Aug 12 04:15:30 Pacific 2007`.
`property` *`(propertyName)`*	Inserts the current value of the server property specified by *propertyName* (e.g., watt.server.port). See *webMethods Integration Server Administrator’s Guide* for a list of server properties.

Effect on Scope

None

Examples

The following example inserts the name of the server processing the DSP or template and the date on which it was processed:

```
.
```

```
.
.
Response generated on %sysvar date% by host %sysvar host%
.
.
.
```

The following example includes the value of the “watt.server.port” property, which identifies the server’s main HTTP listening port:

```
.
.
.
<p>
%sysvar host% was listening on %sysvar property(watt.server.port)%
<p>
.
.
.
```

%validConst%

You use the `%validConst%` tag to specify a list of external URLs that will be permitted for redirection from the page.

Syntax

```
%validConst url(url1,url2,...,urln)%
%endvalidConst%
```

Arguments

`url(`*`url1,url2,...urln`*`)%` is one or more URLs to which the page can be redirected. Use a comma-separated list to specify more than one URL.

Effect on Scope

None

Examples

The following example shows how to allow the page to redirect to the URLs http://example.com, http://example.org, and http://example.net:

```
.
.
.
%validConst url(http://example.com,http://example.org,http://example.net)
%endvalidConst
.
.
.
```

%value%

You use the `%value%` tag to insert the value of a specified variable in a document.

Syntax

```
%value [Variable] [option option option...]%
```

Arguments

Variable specifies the name of the variable whose value you want to insert into the code. You may specify the name of a variable that exists in the pipeline or the following keyword/reserved words.

Variable	Description
`$key`	Inserts the name of the current key. You can use this keyword when looping over a document (i.e., using the `%loop -struct%` tag) to retrieve the name of each key in the current scope or a specified document.

`url` *`url`*
 	
`returnurl` *`url`*
	Instructs Integration Server to redirect to the specified URL. Software AG recommends that you only use the reserved words `url` or `returnurl` when you want to redirect a DSP page to a location internal to Integration Server, by using a relative path (e.g., `%value url ../redirectedurl.dsp %`), or to redirect to an external URL that you have allowed using the `%validConst%` tag.
	
	If you attempt to use the reserved words `url` or `returnurl` to redirect to an external URL (e.g., http://example.com) that you have *not* allowed using the `%validConst%` tag, the setting of the watt.core.template.enableSecureUrlRedirection server configuration parameter governs the server's behavior. When the watt.core.template.enableSecureUrlRedirection parameter is:
	
	■ `true`, Integration Server uses secure URL redirection and prepends the value "error.dsp?data=" to the output of the `%value%` tag (e.g., error.dsp?data=http://example.com). This is the default setting for the watt.core.template.enableSecureUrlRedirection parameter.
	
	■ `false`, Integration Server does not use secure URL redirection. That is, Integration Server does *not* prepend "error.dsp?data=" to the external URL and, as a result,

Variable	Description
	redirection to the external URL is possible, which might put your application at risk.

When you specify *Variable*, the server retrieves the variable from the current scope. To select a variable outside the current scope, you use the following symbols to address it.

You would use...	To...
/Variable	Insert Variable from the initial scope.
../Variable	Insert Variable from the parent of the current scope.
DocName/Variable	Insert Variable from a document named DocName.

If you don't specify *Variable*, the current element within the current scope is assumed. (See example of this usage, below.)

Options

You can use any of the following options with this tag. To specify multiple options, separate the options with spaces.

Option	Description
`null='`*`AnyString`*`'`	Specifies the string that you want the server to insert when *Variable* is null. You specify the string in *AnyString*. For example: `%value carrier null='No Carrier Assigned'%`.
`empty='`*`AnyString`*`'`	Specifies the string that you want the server to insert when *Variable* contains an empty string. You specify the string in *AnyString*. For example: `%value description empty='Description Not Found'%`.
`index=`*`IndexNum`*	Specifies the index of an element that you want to insert. You can use this option to extract an element from an array variable. Specify an integer that represents the element's position in the array. (Arrays are zero based.) For example: `%value backItems index=1%`.
`$index`	Returns the current index number in an array. This option is usually used with the `%loop%` tag to print the loop's iteration. See "%loop%" on page 53 for more information and an example.

Option	Description
`encode (Code)`	Encodes the contents of *Variable* prior to inserting it, where *Code* specifies the encoding system you want the server to apply to the string.

Set Code to...	To encode the value using...
`xml`	XML encoding
`b64`	Base-64 encoding
`url`	URL encoding
`none`	No encoding

Note: When the watt.core.template.enableFilterHtml parameter is set to `true` (the default), the output from a `%value Variable%` tag, including XML and JavaScript, is HTML encoded to prevent cross site scripting attacks. To display the value without HTML encoding, use the `encode` option with *Code* set to `none` (`%value Variable encode(none)%`). For more information about the watt.core.template.enableFilterHtml parameter, see *webMethods Integration Server Administrator's Guide*.

Option	Description
`decode (Code)`	Decodes the contents of *Variable* prior to inserting it, where *Code* specifies the way in which *Variable* is currently encoded.

Set Code to...	To encode the value using...
`b64`	Base-64 encoding
`url`	URL encoding

Option	Description
`+nl`	Adds a new line character. This option is used within the `%value%` tag so that the lines in the HTML document generated by the server would not be concatenated. The `+nl` is similar to the `%nl%` tag. For more information on `%nl%`, see "%nl%" on page 57. For example:

```
%value +nl /carrier/Name%
%value +nl /arrival/Date%
```

Option	Description

	The result of the service will be:
	`carrierName`
	`arrivalDate`

Effect on Scope

None

Examples

The following example invokes a service and then uses the `%value%` tag to insert the results of the service into the DSP

```
.
.
.
%invoke orders:getOrderInfo%<br>
<p>%value buyerInfo/companyName%<br>
  %value buyerInfo/acctNum%
  <P>This shipment contains the following items</P>
  <TABLE WIDTH="90%" BORDER="1">
  <TR><TD>Number</TD><TD>Qty</TD><TD>Description</TD><TD>Status</TD></TR>
  %loop items%
   <TD>%value stockNum%</TD>
   <TD>%value qty%</TD>
   <TD>%value description%</TD>
   <TD>%value status%</TD>
   </TR>
  %endLOOP%
  </TABLE>
%endinvoke%
.
.
.
```

The following example dumps the content of the current scope.

```
.
.
.
<p>
  %loop -struct%
    %value $key% %value%<br>
  %endloop%
<p>
.
.
.
```

The following example inserts the contents of *carrier* . If carrier is null, the string “UPS” is inserted. If carrier is empty, the string “UPS” is also inserted.

```
%value carrier null=UPS empty=UPS%
```

B DSPs and Output Templates in Different Languages

Overview

This appendix contains procedures and guidelines for using DSPs and output templates in different languages (or for translating into other languages) with webMethods Integration Server.

Creating DSPs and Templates in Other Languages

Creating Localized DSPs and Templates

A *localized* DSP or template simply means that it is a DSP or template tailored to display on a client machine that uses a particular language. The term "locale" is used to represent the language and the country that the language is used in, given that language usage is different depending on the country (for example, English in the United States versus English in the United Kingdom).

You create a localized DSP or template just like any other DSP or template. For details, see "Using Dynamic Server Pages (DSPs)" on page 9 or "Using Output Templates to Format Service Output" on page 35. Keep the following points in mind:

- You should create a default DSP or template file in case of error, if one does not already exist. For details, see "Creating a Default DSP or Template" on page 72.
- You must create the localized file using the same encoding as the default file, if it exists. Using UTF-8 encoding will reduce difficulty in localizing later into other languages.

Creating a Default DSP or Template

Before publishing your localized DSPs or templates to Integration Server, it is recommended that you create a default DSP or template. This prevents errors from being displayed when a client machine's locale does not match any localized DSPs or templates within the package on the server. If this occurs, an error appears to the user unless a default DSP or template is present. If it is present, that file appears instead of the error.

You create a default DSP or template like any other DSP or template; however, it must have the same filename and encoding as the localized DSP or template. Create the default file with UTF-8 encoding to reduce difficulty in localizing later into other languages. In addition, keep the contents of this file generic, because it is displayed for all languages and locales except for the ones for which you provided localized files.

You can create the default file in any language. Select the language for the file carefully, because the default file will be displayed to all users for who you did not supply a specific file for their locale (or to those users who did not set their browser to request a language).

The default file is located at the root directory of published DSPs and templates in a package. For details on creating DSPs or templates, see "Using Dynamic Server Pages (DSPs)" on page 9 or "Using Output Templates to Format Service Output" on page 35. For details on publishing the default DSP or template to the server, see the next section.

Points to Remember

- You must create the default DSP or output template file with the same filename and encoding as its corresponding localized files on the server.
- You should create the default file with UTF-8 encoding to reduce difficulty in localizing later into other languages.
- Keep the contents of the default file generic.

Note: If you already created or imported an output template in Designer for a service in a package, then you already have a default template for that package.

Using Localized DSPs and Templates

Setting Up Your Browser

The first step in using a localized DSP or template is to make sure that your Internet browser will load and display it in the appropriate language. Integration Server stores this setting as your user locale.

If you do not set a language preference in your browser and you run Windows, Integration Server uses the default language that Windows uses. (This information is inferred from the HTTP Accept-Language header sent by your browser.)

For instructions for setting language preferences, see your browser's help.

Publishing Localized DSPs and Templates

Before you publish your localized DSPs and templates to Integration Server, make sure that you saved the files with the same encoding as the default language file. It is strongly recommended to use UTF-8 encoding for all DSPs and templates, particularly in multilingual environments.

Publishing localized DSPs and templates to Integration Server is relatively simple. After you create your localized DSP or template file (and a default file), you place the DSP or template file in a directory on the server. See the following procedures.

Publishing a Localized DSP to the Server

To publish a localized DSP to the server

1. Save the DSP document in a text file that has a ".dsp" extension. Make sure that the default DSP file has the same name and encoding as the localized DSP file. For example: `showorders.dsp`.
2. Place the *default* DSP file in the pub directory of the package in which you want the DSP to reside. For example, to publish a default DSP in the orders package, you would copy it to:

   Integration Server_directory\packages\orders\pub
3. Place the *localized* DSP file in the pub\localeID directory of the package in which you want the DSP to reside. You may have to create this directory. The *localeID* corresponds to a language code as defined in RFC 3066 by the W3C. For details on *localeID* values, see "The localeID Value" on page 76. For example:
   - To publish a Japanese DSP in the orders package, you would copy it to:

     \packages\orders\pub\ja\
   - To publish an English DSP in the orders package, you would copy it to:

     \packages\orders\pub\en\
   - To publish a United Kingdom English DSP in the orders package, you would copy it to:

     \packages\orders\pub\en-GB\
   - To publish a United Kingdom English DSP to the status subdirectory within the orders package, you would copy it to:

     \packages\orders\pub\en-GB\status\

Publishing a Localized Template to the Server

To publish a localized template to the server

1. Save the template document to a file. Make sure that the default template file has the same name as the localized template file. For example: `showorders.html`.
2. Place the *default* template file in the templates directory of the package in which you want the template to reside. For example, to publish a default template in the orders package, you would copy it to:

   Integration Server_directory\packages\orders\templates\
3. Place the *localized* template file in the templates\localeID directory of the package in which you want the template to reside. You may have to create this directory. The *localeID* corresponds to a language and country code as defined in RFC 3066 by the

W3C. For details on localeID values, see "The localeID Value" on page 76. For example:

- To publish a Japanese template in the orders package, you would copy it to:

  Integration Server_directory\packages\orders\templates\ja\

- To publish an English template in the orders package, you would copy it to:

  Integration Server_directory\packages\orders\templates\en\

- To publish a United Kingdom English template in the orders package, you would copy it to:

  Integration Server_directory\packages\orders\templates\en-GB\

- To publish a United Kingdom English template in the status subdirectory within the orders package, you would copy it to:

  Integration Server_directory\packages\orders\templates\en-GB\status\

Note: If you have references to other files within your DSP or template (for example, CSS and GIF files), the same localization rules apply. Make sure to publish those files in the appropriate directory for locale.

Requesting a Localized DSP

To process a localized DSP, you request it from a browser using the following URL format:

```
http://hostName:portNum/packageName/fileName.dsp
```

For details on the values in the URL, see "Requesting DSPs" on page 16.

Examples

The following URL retrieves showorders.dsp from a package named ORDER_TRAK on a server named rubicon:

```
http://rubicon:5555/ORDER_TRAK/showorders.dsp
```

If the client machine has a Japanese locale:

- Integration Server looks for:

  Integration Server_directory\packages\ORDER_TRAK\ja-JP\showorders.dsp

- If not found, then it looks for just the language code of the *localeID* as follows:

  Integration Server_directory\packages\ORDER_TRAK\ja\showorders.dsp

- If not found, then it looks for the next language listed in the browser language settings. If another language is not listed, then it looks for a default DSP at:

  Integration Server_directory\packages\ORDER_TRAK\showorders.dsp

- If a default DSP does not exist, then an error appears in the browser.

The following URL retrieves showorders.dsp from the STATUS subdirectory in a package named ORDER_TRAK on a server named rubicon:

```
http://rubicon:5555/ORDER_TRAK/STATUS/showorders.dsp
```

If the client machine has a United States English locale:

- Integration Server looks for:

  Integration Server_directory\packages\ORDER_TRAK\STATUS\ en-US \showorders.dsp
- If not found, then it looks for just the language code of the *localeID* as follows:

  Integration Server_directory\packages\ORDER_TRAK\STATUS\en\ showorders.dsp
- If not found, then it looks for the next language listed in the browser language settings. If another language is not listed, then it looks for a default DSP at:

  Integration Server_directory\packages\ORDER_TRAK\STATUS\showorders.dsp
- If a default DSP does not exist, then an error appears in the browser.

The *localeID* Value

When you publish localized DSPs and templates to Integration Server, they must reside in directories that correspond to the appropriate locales. The *localeID* value represents the locale in the following directory path:

Integration Server_directory\packages\orders\pub\localeID\

The *localeID* value corresponds to the language code and country code of the locale as defined in RFC 3066 by the W3C. The syntax of the localeID is as follows:

languagecode-COUNTRYCODE

where:

languagecode	Is the language code as defined by ISO-639. This value must be lowercase.
COUNTRYCODE	Is the country code as defined by ISO-3166. This value must be uppercase. If this value is not supplied at run time, Integration Server uses the language code only. If the language code is not supplied, then the default DSP or template (if present) is displayed in the browser.

You can also have a *localeID* that is less specific by using the language code only. For examples, see the following table.

This localeID...	Represents...
en-US	English as used in the United States
en-GB	English as used in the United Kingdom
en	English, non-country specific
ja-JP	Japanese as used in Japan
ja	Japanese, non-country specific
fr-CA	French as used in Canada

For More Information

For a full listing of language codes as defined by ISO-639, see:

"http://www.loc.gov/standards/iso639-2/langcodes.html"

For a full listing of country codes as defined by ISO-3166, see:

"http://www.iso.org/iso/country_codes"

For details on RFC 3066, which established the *localeID* format, see:

"http://www.ietf.org/rfc/rfc3066.txt"

Index