Module 5: ISML

Learning Objectives

After completing this module, you will be able to:

- Use ISML tags in templates, including: `<isset>`, `<isinclude>`, `<isdecorate>`, and conditional tags.
- Use local and remote includes in ISML.

Introduction

Internet Store Markup Language (ISML) templates are files with an extension of `.isml`. They define how data, tags, and page markup are transformed into HTML that is sent to the browser, using Cascading Style Sheets (CSS) for page layout and styling.

The Demandware platform uses templates to generate dynamic HTML-based web pages for responses sent back to the client. Templates are created using ISML tags and expressions.

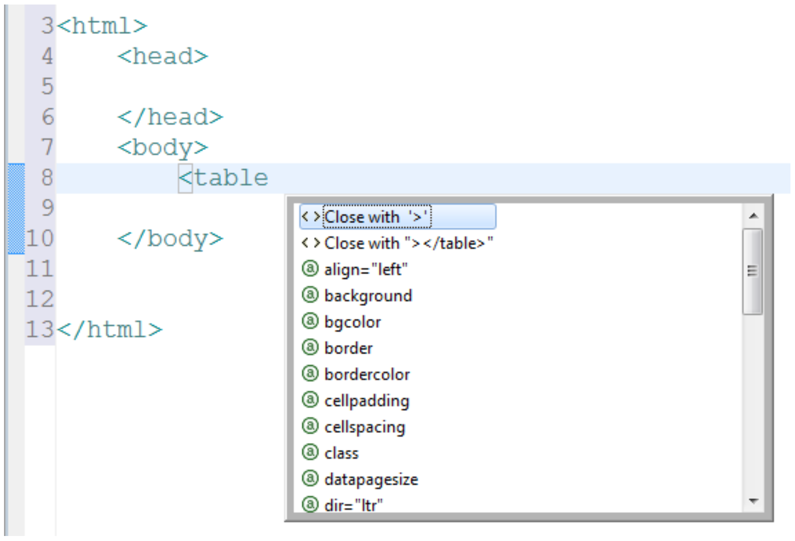
When describing a Demandware application using the Model-View-Controller (MVC) pattern, templates represent the view, pipelines represent the controller and the DW Script API represents the model.

Create an ISML Template

To create an ISML template, follow these steps:

1. In UX Studio, select a cartridge in Navigator View. Select **File > New > ISML Template.** The **Create Template** dialog displays.
2. In the parent folder field, enter the name of the folder where you want to store your template. If the folder does not exist it will be created.
3. In the Template name box, enter a name for your template. There is no need to type the .isml extension.
4. Click **Finish**.

5. Your new template opens in the ISML editor in UX Studio. This editor supports HTML and ISML system tag auto-completions as shown.



```
<html>
    <head>

    </head>
    <body>
        <table

    </body>


</html>
```

Lesson 5.1: ISML Tags and Expressions

ISML tags are Demandware proprietary extensions to HTML that developers use inside ISML templates. ISML tags and expressions cannot be written in any other file other than ISML templates. ISML tags are SGML-like extension tags that start with **is**, e.g. <isprint> and describe, together with regular HTML, how dynamic data will be embedded and formatted on the page.

Depending on their tasks, ISML tags can be divided into the following groups:

Group	Tags	Purpose
HTTP-related	`<iscookie>`	Sets cookies in the browser
	`<iscontent>`	Sets the MIME type
	`<isredirect>`	Redirects browsers to specific URLs
	`<isstatus>`	Define status codes
Flow Control	`<isif>`	Evaluates a condition
	`<iselse>` `<iselseif>`	Specifying alternative logic when an `<isif>` condition does not evaluate to true
	`<isloop>`	Creates a loop statement
	`<isnext>`	Jumps to the next iteration in a loop statement
	`<isbreak>`	Terminates loops
Variable-related	`<isset>`	Creates a variable
	`<isremove>`	Removes a variable
Include	`<isinclude>`	Includes the contents of one template on the current template
	`<ismodule>`	Declares a custom tag
	`<iscomponent>`	Includes the output of a pipeline on the current page
Scripting	`<isscript>`	Allows Demandware Script execution inside templates
Forms	`<isselect>`	Enhances the HTML `<select>` tag
Output	`<isprint>`	Formats and encodes strings for output
	`<isslot>`	Creates a content slot
Others	`<iscache>`	Caches a page
	`<iscomment>`	Adds comments
	`<isdecorate>`	Reuses a template for page layout
	`<isreplace>`	Replaces content inside a decorator template
Active Data	`<isactivedatahead>`	Allows collection of active data from pages with a `<head>` tag
	`<isactivecontenthead>`	Collects category context from a page for active data collection
	`<isobject>`	Collects specific object impressions/views dynamically

ISML Expressions

ISML Expressions are based on the Demandware Script language. Since Demandware Script implements the ECMAScript standard, access to variables, methods, and objects is the same as using JavaScript.

ISML expressions are embedded inside `${…}` to enable the ISML processor to interpret the expression prior to executing an ISML tag or the rest of the page. ISML expressions provide access to data by using dot notation. This example accesses a property of the `Product` object in the pipeline dictionary:

```
${pdict.myProduct.UUID}
```

The difference between this ISML expression and one used inside a pipeline node property (i.e. decision node) is that in ISML you must specify the `${pdict.`*`object.property`*`}` if you want to access a value in the pipeline dictionary, whereas inside pipeline node properties the access to the `pdict` is implicit and the `${}` not used: i.e. `Product.UUID.`

ISML expressions can also access Demandware Script classes and methods. Two packages are available implicitly in ISML, so classes do not need to be fully qualified:

1. `TopLevel` package: `session.getCustomer()`
2. `dw.web` package: `URLUtils.url(), URLUtils.webRoot()`

   `TopLevel` package has a class named `global` which is also implied so it never has to occur in the prefix.

Other access to classes and methods must be fully qualified:

```
${dw.system.Site.getCurrent().getName()}
```

Here are some more examples of ISML expressions:

```
${TopLevel.global.session.getCustomer().getProfile().getLastName()}
```

Since TopLevel package and global class is implicit, the above code is equivalent to code below.

```
${session.getCustomer().getProfile().getLastName()}
```

The getter method can be replaced with properties also. So the above code is equivalent to code below.

```
${session.customer.profile.lastName}
${pdict.CurrentSession.customer.profile.lastName}
${pdict.CurrentCustomer.profile.lastName}
${dw.system.Site.getCurrent().getName()}
${dw.system.Site.current.name}
```

ISML expressions can also allow complex arithmetical, boolean and string operations:

```
${pdict.myProduct.getLongDescription() != null}
```

In this module, we will cover the most frequently used tags: `<isset>`, `<isinclude>`, `<isdecorate>`, `<isloop>` and the conditional tags `<isif>`, `<iselseif>`, and `<iselse>`

Note: Although there are some ISML tags that do not need a corresponding closing `</>` tag (i.e.: the `<isslot>` tag), it is best practice to always use a closing tag.

<isredirect> tag

This tag can redirect the control to another pipeline and redirect can be permanent or temporary.

```
<isredirect location="${URLUtils.https('Account-Show')}"
permanent="true"/>
<isredirect location="${URLUtils.url('LoginPanel')}">
<isredirect location="${URLUtils.url('LoginPanel-Start')}"
permanent="false">
```

<iscomment> tag

This tag is used to write comments in the ISML. For example.

```
<iscomment> ....This is a comment....</iscomment>
```

<isprint> tag

This tag can print formatted output of a variable or an expression to the browser. In order to do so, it uses built in styles or formatters. You can see the documentation for formatters. Here are examples of using isprint with styles.

```
<isprint value="${myMoney}" style="MONEY_LONG"/>
<isprint value="${myMoney}" style="MONEY_SHORT"/>
<isprint value="${myNumber}" style="DECIMAL"/>
<isprint value="${myNumber}" style="INTEGER"/>
<isprint value="${myDate}" style="DATE_LONG"/>
<isprint value="${myDate}" style="DATE_SHORT"/>
<isprint value="${myString}" encoding="off"/>
```

`MONEY_LONG` prints money with currency symbol e.g. $3,333.00

`MONEY_SHORT` prints money without the symbol e.g. 3,333.00

`DECIMAL` prints the value with two decimal places e.g. 3,455.35

`INTEGER` rounds of and prints only the integer portion e.g. 3,455

`DATE_LONG` prints date in the long format like Jul 24, 2016

`DATE_SHORT` prints date in the long format like 07/24/2016

`encoding="off"` prints strings containing HTML, for example:

`<h1> Welcome to Demandware Class</h1>` will be printed as:

Welcome to Demandware Class

Lesson 5.2: Creating and Accessing Variables

You can create and access your own custom variables in an ISML template by using the `<isset>` tag.

When using the `<isset>` tag, name and value are required attributes that must be assigned. The default scope is `session`, so you must be careful to qualify your variables accordingly if you do not want them.

Example:

```
<isset
name = "<name>"
value = "<expression>"
scope = "session"|"request"|"page"
>
```

Here are some examples of using `isset` tag and retrieving the variables back from the scope

session Scope

```
<isset name = "x" value = "12343" scope="session"/>
<isset name = "x" value = "12343" />   (session is implied here)
<isset name = "x" value = "${12343}" scope="session"/>
```

Retrieving from session

```
${session.custom.x}
${pdict.CurrentSession.custom.x}
```

request Scope

```
<isset name="x" value="${12343}" scope="request"/>
${request.custom.x}
${pdict.CurrentRequest.custom.x}
```

pdict Scope

```
<isset name = "x" value = "${12343}" scope = "pdict"/>
```

Retrieving from pdict

```
${pdict.x}
```

Page Scope

```
<isset name = "x" value = "${12343}" scope = "page"/>
${page.custom.x}   does not work
```

Retrieving form page

```
${page.x} does not work
${x} works
```

Value Attribute

The value attribute can be a hardcoded string or number, or it can be an ISML expression accessing another variable or object.

Value Type	Example
`String`	`value="hardcoded text"`
`expression`	`value="${pdict.myProduct.name}"`

Scope Attribute

A variable's scope attribute refers to its accessibility level, such as `session`, `request`, and `page`. It is important to understand the scopes of a variable and which objects can access that variable at each level. Listed are the scopes from widest to narrowest access.

Scope	Description
Global Preferences	Available to any site within an organization. Accessible via the `dw.system.OrganizationPreferences` class.
Site Preferences	Available to any pipeline executing as part of a site. Accessible via the `dw.system.SitePreferences` class.
Session	Available through the whole customer session, even across multiple requests. Any variable added to the session scope becomes a custom attribute of the session object. Since it is not a standard attribute it must be accessed with the `session.custom` qualifier: `${session.custom.myVar}`
`pdict`	Available while a pipeline executes. It can encompass multiple requests, similar to **Interaction Continue Nodes**.
`request`	Available through a single browser request-response cycle; it does not persist in memory for a subsequent request. Typically it is the same as the pipeline scope. They are available via the `request` scope. Similar to session variables, you must prefix request variables with a qualifier `request.custom` when accessing them: `${request.custom.myRequestVar}`
`page`	Available only for a specific ISML page, and its locally included pages. Their scope is limited to the current template, and any locally included templates. They are accessed without a prefix: `${pageVar}`
`slotcontent`	Available only in the rendering template for a content slot.
`<isloop>` variable	Available only inside the loop.

Lesson 5.3: Reusing Code in Templates

Reusable code saves time in both code creation and update. It also reduces errors and helps to ensure a consistent look and feel.

You can use the following tags to reuse code in ISML templates:

Tag	Description
`<isinclude>`	Enables you to embed an ISML template inside an invoking template. There are two types: ▪ **Local Include** – include the code of one ISML template inside of another while generating the page. All variables from the including template are available in the included template, including page variables. SiteGenesis uses local includes extensively. ▪ **Remote Include** –include the output of another pipeline inside of an ISML template. This is used primarily for partial page caching. **Note**: Pipeline dictionary and page variables from invoking template are **not** available in the included template. The only variables available to a remotely included pipeline are session variables. **Note**: Includes from another server are not supported.
`<isdecorate>`	Enables you to decorate the enclosed content with the contents of the specified (decorator) template. A decorator is an ISML template that has HTML, CSS, and the overall page design.
`<ismodule>`	Enables you to define your own ISML tags which can be used like any standard tags.
`<iscomponent>`	Invokes a remote include. It enables you to pass as many attributes as you want without having to use the `URLUtils` methods.

Local Includes

Use the following syntax:

```
<isinclude template=”[directory/]templatename”/>
```

Note: You do not need to add the `.isml` extension when including a template.

Example

Template 1:

```
<h1>My Template</h1> <br/>
<isinclude template=”extras/calendar”/>
```

Template 2:

(`calendar.isml`)

```
<h1>Included template</h1>
```

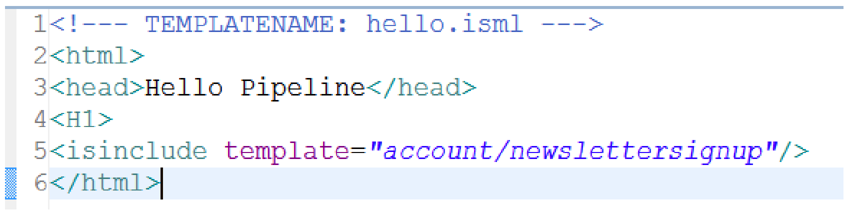
When the browser renders the template, the user will see:

My Template

Included template

To locally include one template into another using the `<isinclude>` tag, follow these steps:

1. Open any ISML template.
2. In the ISML code, determine where you want to embed the locally included template.
3. Add the `<isinclude>` tag to the template, using the following as an example:



```
<!--- TEMPLATENAME: hello.isml --->
<html>
<head>Hello Pipeline</head>
<H1>
<isinclude template="account/newslettersignup"/>
</html>
```

4. Save the template.
5. To test, use your template in a pipeline.

Remote Includes

The syntax is:

```
<isinclude url=”pipeline_url”/>
```

Using a remote include in a template will invoke another pipeline which returns HTML at runtime. The following examples show how to call a pipeline without passing URL parameters:

```
<isinclude url="${URLUtils.url('Product-IncludeLastVisited')}" />
```

In this example, the `dw.web.URLUtils url()` method builds a site-specific URL for the `Product-IncludeLastVisited` pipeline. This is a best practice since you should never hardcode a pipeline URL since it would contain a specific server in it. Use the `URLUtils` methods instead.

Here is an example of passing URL parameters:

```
<isinclude url="${URLUtils.https('Product-
GetLowATSThreshold','productid','ETOTE','typeOfTV','Wide-screen')}"/>
```

The page generated by the invoked pipeline can be dynamic or it may come from cache.

You can also implement a remote include, via the `<iscomponent>` tag. It supports passing multiple attributes.

```
<iscomponent
   pipeline = <string> | <expression>
   [locale = <string> | <expression> ]
   [any number of additional arbitrarily named parameters]
/>
```

Example

```
<iscomponent pipeline="Product-GetLowATSThreshold" productid="ETOTE"
typeOfTV="Wide-screen"/>
```

Using a Remote Include

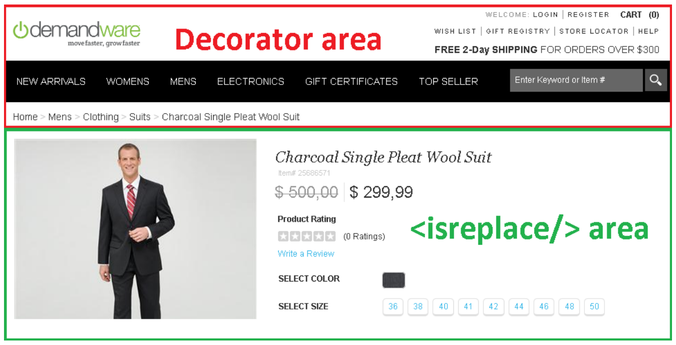
To remotely include a pipeline into a template, follow these steps:

1. Open an ISML template.
2. In the ISML code, determine where you want to embed the remotely included pipeline.
3. Add the `<isinclude>` tag to the template using the following as an example (`param` and `value` are optional):

   ```
   <isinclude url="${URLUtils.url(‘Pipeline-StartNode’, [‘param’,
   ‘value’, …])}"/>
   ```
4. Save the template.
5. To test, use your template in a pipeline, being called by an **interaction** node.

The `<isdecorate>` Tag

The decorator template uses `<isreplace/>` to identify where to include the decorated content. The following example shows a decorator and the area where the code is being replaced.



Typically, the decorator template only uses one tag, `<isreplace/>`. However, you can use multiple tags. If the decorator template uses multiple `<isreplace/>` tags, the content to be decorated will be included for each `<isreplace/>` tag.

A typical use case is to decorate the content body with a header and footer.

Example:

Template using a decorator

```
<isdecorate template="decoratorFolder/pt_myDecorator">
  ...My content...to be decorated
</isdecorate>
```

Decorator Template (`templates/default/decoratorFolder/pt_myDecorator.isml`)

```
<html>
    <head>…</head>
      <body>
        This contains Header/Banner/Menus etc.
      <isreplace/>
        This contains footer/Copyright notice etc.
      </body>
  <html>
```

Final generated page

```
<html>
    <head>…</head>
      <body>
        This contains Header/Banner/Menus etc.
      ...My content...to be decorated
        This contains footer/Copyright notice etc.
      </body>
 <html>
```

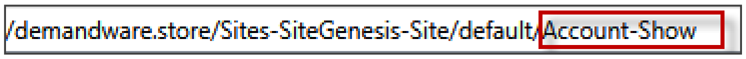
Using the <isdecorate> Tag

To use the `<isdecorate>` tag, follow these steps:

1. Open the ISML template that has the code you want to replace in a decorator. Add the `<isdecorate>` tag around the code to include in a decorator.

   ```
   <isdecorate template="[directory/]decoratorname">
       Your code goes here.
   </isdecorate>
   ```

2. Save the template.
3. Open the decorator template. If you are using a SiteGenesis template, the decorator templates names start with `pt_`.
4. Find the location in the code where you want to use the `<isreplace/>` tag. Add the tag to the template.
5. Test the page by calling the pipeline that uses the decorator template. For example, if the decorator template is used by the `Account-Show` pipeline/start node, type in the URL that will execute the `Account-Show` pipeline.

   
   /demandware.store/Sites-SiteGenesis-Site/default/Account-Show

Creating Custom Tags with <ismodule>

There are **three key ISML files required** for creating and using a custom tag:

1. The ISML file which sets the values of any attributes of the custom tag. This example is in `util/modules.isml`:

```
<ismodule template="components/breadcrumbs"
   name="breadcrumbs"
   attribute="bctext1"
   attribute="bcurl1"
   attribute="bctext2"
   attribute="bcurl2"
   attribute="bctext3"
   attribute="bcurl3"
/>
```

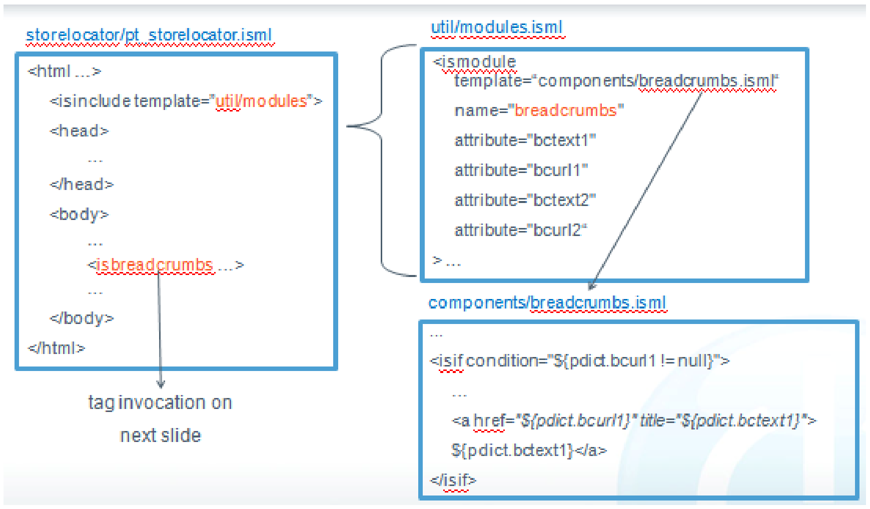
2. The ISML file which specifies what happens when the attributes are passed. See the code snippet from inside `breadcrumbs.isml`:

```
<isif condition="${pdict.bcurl1 != null}">
   …
   <a href="${pdict.bcurl1}" title="${pdict.bctext1}">
   ${pdict.bctext1}</a>
</isif>
```

3. Invoke the custom tag inside an ISML template:

```
<html …>
<isinclude template="util/modules"/>
<head>
…
</head>
<body>
…
<isbreadcrumbs bctext1="…" bcurl1="…"/>
</body>
</html>
```

Here is how it would be put together.



storelocator/pt_storelocator.isml

```
<html ...>
    <isinclude template="util/modules">
    <head>
        ...
    </head>
    <body>
        ...
        <isbreadcrumbs ...>
        ...
    </body>
</html>
```

util/modules.isml

```
<ismodule
    template="components/breadcrumbs.isml"
    name="breadcrumbs"
    attribute="bctext1"
    attribute="bcurl1"
    attribute="bctext2"
    attribute="bcurl2"
> ...
```

components/breadcrumbs.isml

```
...
<isif condition="${pdict.bcurl1 != null}">
    ...
    <a href="${pdict.bcurl1}" title="${pdict.bctext1}">
    ${pdict.bctext1}</a>
</isif>
```

Lesson 5.4: Conditional Statements and Loops

Every programming language provides the ability to evaluate a condition to determine what logical path the program should take. Most languages use the keywords *if*, *else if*, and *else*. Demandware uses similar keywords, but adds `is` to the beginning of the syntax:

```
<isif condition=”${ISML expression evaluated}”>
     Do something here if true.
<iselseif condition=”${check another condition}”>
     Do something if this one is true.
<iselse>
    If none of the above conditions are true, do this.
</isif>
```

Using Conditional Statements

To use a conditional statement in an ISML template, follow these steps:

1. Determine the location on your ISML page where you want to write your conditional statement.
2. Open your conditional statement with the `<isif condition=””>` tag.

Example:

```
<isif condition="${pdict.myProduct.online}">
  Product is online
<iselse>
 Product is offline
</isif>
```

Loops

With `<isloop>` you can loop through the elements of a specified collection or array. For example, you can list data such as: categories, products, shipping and payment methods. You can nest `<isloop>` statements.

You can use the following supporting tags with `<isloop>`:

- Use the `<isbreak>` tag within a loop to terminate a loop unconditionally. If used in a nested loop, it terminates only the inner loop.
- Use `<isnext>` to jump forward within a loop to the next list element of an iterator. This tag affects only the iterator of the inner loop. If an iterator has already reached its last element, or an iterator is empty when an `<isnext>` is processed, the loop is terminated instantly.

The full syntax for using the `<isloop>` tag is:

```
<isloop
iterator|items = "<expression>"
[ alias|var = "<var name>" ]
[ status = "<var name>" ]
[ begin = "<expression>" ]
[ end = "<expression>" ]
[ step = "<expression>" ]>
…do something in the loop using <var_name>…
</isloop>
```

The attributes have the following usage:

Attribute	Description
`items` (iterator)	Expression returning an object to iterate over. Attributes *iterator* and *items* can be used interchangeably.
`var` (alias)	Name of the variable referencing the object in the iterative collection referenced in the current iteration.
`status`	Name of the variable name referencing loop status object. The loop status is used to query information such as the counter or whether it is the first item.
`begin`	Expression specifying a begin index for the loop. If the begin is greater than 0, the `<isloop>` skips the first x items and starts looping at the begin index. If begin is smaller than 0, the `<isloop>` is skipped.
`end`	Expression specifying an end index (inclusive). If end is smaller than begin, the `<isloop>` is skipped.
`step`	Expression specifying the step used to increase the index. If step is smaller than 1, 1 is used as the step value.

For the `status` variable, the following properties are accessible:

Attribute	Description
`count`	The number of iterations, starting with 1.
`index`	The current index into the set of items, while iterating.
`first`	True, if this is the first item while iterating (count == 1).
`last`	True, if this is the last item while iterating.
`odd`	True, if count is an odd value.
`even`	True, if count is an even value.

For example, if the `<isloop>` tag declares a `status="loopstate"` variable, then it is possible to determine the first time the loop executes by using: `<isif condition="loopstate.first">`.

Another example of `<isloop>` tag is :

```
 <isloop items="${order.object.shipments}" var="Shipment"
status="loopState">
<isif condition="${loopState.count >= (pdict.OrderPagingModel.pageSize +
1)}">
     <isbreak/>
</isif>
     <isif condition="${loopState.count==0}">
          <isnext/>
     </isif>
     ${loopState.count}
     ${loopState.index}
     ${loopState.first}
     ${loopState.last}
     ${loopState.even}
     ${loopState.odd}
</isloop>
```