demandware®
A Salesforce Company

DEV 101: DEVELOPING IN DEMANDWARE

Student Guide

demandware
A Salesforce Company

Table of Contents

Introduction 7
Module 1: Getting Started 11
Lesson 1.1: Platform Review 11
Lesson 1.2: Platform Tools – Business Manager 12
Exercise: Business Manager Organization 13
Exercise: Create an Empty Site 14
Lesson 1.3: SiteGenesis Foundation Architecture 15
Exercise: Import SiteGenesis from a SiteGenesis Package 16
Lesson 1.4: Site Configuration 17
Exercise: Disable caching for an Empty Site or Training Site 18
Exercise: Re-Index Search for SiteGenesis 18
Sharing Data between Sites 19
Exercise: Share a Catalog between Sites and Set a Product Offline 20
Knowledge Check 21
Module 2: UX Studio 22
Lesson 2.1: Creating a Workspace 22
Exercise: Install the UX Studio Plugin and Create a Workspace 22
Lesson 2.2: Creating a Server Connection 23
Exercise: Create a New Server Connection 24
Exercise: Run the View Demandware Help 24
Lesson 2.3: Importing Projects (Cartridges) 25
Exercise: Import Projects in UX Studio 26
Lesson 2.4: Demandware Views 27
Exercise: Search for Text in Files 28
Knowledge Check 29
Module 3: Cartridges 30
Lesson 3.1: Cartridge Path 30

Exercise: Add a Cartridge to a Cartridge Path 31
Lesson 3.2: Cartridge Types 32
Exercise: View WebDAV cartridge Directory in Business Manager 33
Exercise: Create a New Version on the Server 33
Lesson 3.3: Create a New Cartridge 33
Exercise: Create a New Empty Cartridge 34
Lesson 3.4: Creating a Demandware Storefront Cartridge 35
Exercise: Create a New Storefront Cartridge 36
Knowledge Check 37
Module 4: Pipelines and JavaScript Controllers 38
Lesson 4.1: Creating a Pipeline 40
Exercise: Create a Pipeline 42
Lesson 4.2: Executing a Pipeline 43
Exercise: Execute a Pipeline 44
Exercise: Create a JHelloWorld JavaScript Controller 47
Troubleshooting with the Request Log Tool 48
Exercise: Run the Request Log 49
Exercise: Using a Call Node in a Pipeline 52
Exercise: Create a JavaScript Controller named JCall.js with Your Instructor 55
Lesson 4.3: The Pipeline Dictionary 56
Lesson 4.4: Troubleshooting with the Pipeline Debugger 59
Exercise: Create a Debug Configuration for Pipeline 60
Exercise: Use the Debugger for Pipeline 62
Lesson 4.5: Pipelets 63
Exercise: Use a Demandware Pipelet in a ShowProduct Pipeline 66
Exercise: Create JavaScript Controller JShowProduct 68
Knowledge Check 69

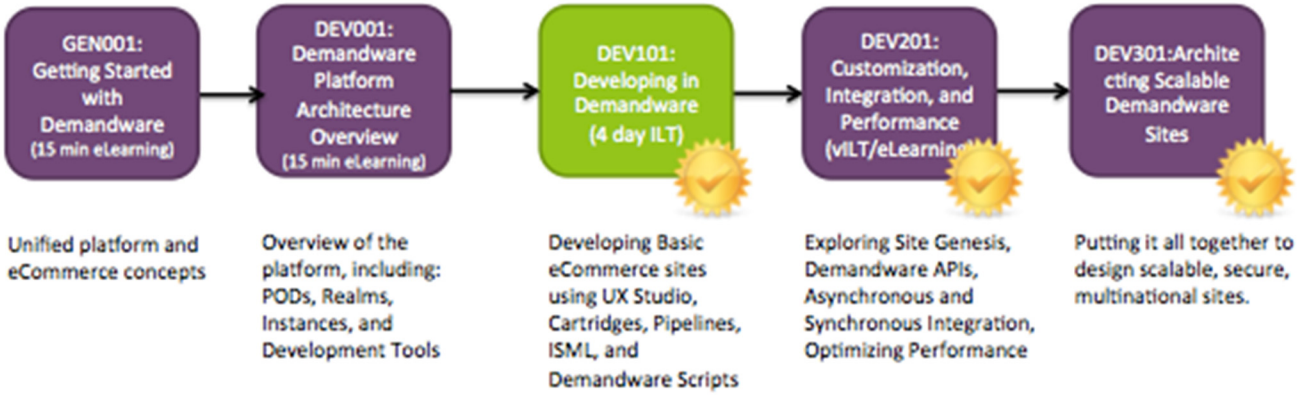
Exercise: Test Your Knowledge of Pipelines 70
Module 5: ISML 71
Lesson 5.1: ISML Tags and Expressions 73
Lesson 5.2: Creating and Accessing Variables 76
Exercise: Set and Retrieve Variables 78
Lesson 5.3: Reusing Code in Templates 79
Exercise: Use Local Includes in ShowProduct Pipeline 81
Exercise: Use Local Includes in JShowProduct JavaScript Controller 82
Exercise: Use Remote Include in ShowProduct Pipeline 84
Exercise: Use Remote Include in JShowProduct JavaScript controller 85
Exercise: Use a Decorator in ShowProduct Pipeline or JShowProduct JavaScript controller 89
Exercise: Use a Custom Tag in ShowProduct pipeline 92
Exercise: Use a Custom Tag in JShowProduct controller 93
Lesson 5.4: Conditional Statements and Loops 94
Exercise: Use Loops in Basket Pipeline 97
Exercise: Creating JBasket JavaScript Controller and Using Loop in ISML 98
Knowledge Check 99
Exercise: Complete Basket Pipeline (Optional) 99
Module 6: Content Slots 100
Lesson 6.1: Creating & Configuring Content Slots 102
Exercise: Create a Slot 104
Lesson 6.2: Using Content Link Functions 106
Exercise: Create a Slot Configuration 107
Knowledge Check 108
Demo: Create a Slot with a Rendering Template for a Vertical Carousel of Products 108
Module 7: Demandware Script 111
Lesson 7.1: Demandware Script API 112

Exercise A: Use Demandware Script in ISML in DScript Pipeline 115
Exercise B: Use Demandware Script JDScript Controller 115
Lesson 7.2: Script Pipelets 116
Exercise: Create a Script Pipelet in ShowProduct Pipeline 121
Exercise: Calling a Script Pipelet in JShowProduct JavaScript Controller 122
Lesson 7.3: Script and JavaScript Controller Debugging 124
Exercise: Create a Script Debug Configuration 125
Lesson 7.4: Resource API and Resource Bundles 127
Knowledge Check 129
Exercise: Use a Resource Bundle in the Demandware Script in ShowProduct Pipeline 130
Exercise: Use a Resource Bundle in the Demandware Script in JavaScript Controller 132
Module 8: Forms Framework 134
Lesson 8.1: XML Metadata File 137
Lesson 8.2: ISML Form Template 139
Lesson 8.3: Form Pipeline Elements 140
Exercise: Use the Forms Framework using Pipeline 144
Exercise: Use the Forms Framework Using JavaScript Controller 147
Knowledge Check 149
Exercise: Create Form Metadata on Your Own 150
Module 9: Custom Objects 151
Exercise: Create a Custom Object Definition 154
Lesson 9.1: Using Demandware Script to Create Custom Object Instances 154
Exercise: Create a Custom Object Instance Programmatically Using Pipeline 157
Example: Implicit and Explicit Transactions in JavaScript Controllers or Scripts 159
Exercise: Create a Custom Object using JavaScript Controller 160
Lesson 9.2: Custom Logging 162
Exercise: Custom Logging in Pipeline 164

Exercise: (This may be a Demo based on Time) Custom Logging in JavaScript Controller 165
Knowledge Check 166
Module 10: Data Binding and Explicit Transactions 167
Lesson 10.1: Data Binding with Forms and Objects 167
Exercise: Use `UpdateObjectWithForm` Pipelet 168
Exercise: Use `UpdateFormWithObject` Pipelet 171
Exercise: Create a Custom Object Using JavaScript Controller 172
Lesson 10.2: Explicit Database Transaction Handling 173
Exercise: Explicit Transaction Handling 174
Knowledge Check 176
Exercise: Retrieve Information from a Custom Attribute and Store it in EditPreferences Pipeline 177
Exercise: Store and Retrieve the Preferences using JavaScript Controller JEditPreferences.js 180
Module 11: Site Maintenance 183
Lesson 11.1: Site and Page Caching 183
Exercise: Page-Level Caching 186
Exercise: Partial Page Caching 188
Exercise: Use the Cache Information Tool 190
Lesson 11.2: Site Performance 191
Lesson 11.3: Code Replication 192
Lesson 11.4: Data Replication 195
Knowledge Check 197
Appendix A: Data Integration: Simple Feeds 198
Exercise: Simple Feed Integration 202
Congratulations 205

demandware
A Salesforce Company

Introduction



About the Course

Description	Participants modify and customize the Demandware reference application, SiteGenesis, through the use of core Demandware programming concepts, files, and scripting language.
Audience	Developers who have the following background and experience: ▪ Java or Javascript programming (at least 2 years) ▪ Working with XML files (data imports and exports) ▪ Familiarity with jQuery library and JSON syntax ▪ Use of Firebug or Web Developer toolkits
Duration	4 days
Prerequisites	To successfully participate in this course, students must complete the following prior to attending class: ▪ GEN001: Getting Started with the Demandware Platform (30 minute eLearning) ▪ DEV001: Demandware Platform Architecture Overview (20 minute eLearning) ▪ Install and test the Demandware UX studio plug-in to the Eclipse IDE on the student's laptop that will be used during class.

System Requirements	▪ A laptop computer (with the appropriate administrative system rights) with the Eclipse IDE and UX studio installed. A high-speed Internet connection.

Course Objectives

Welcome to *Developing in Demandware.* After completing this course, you will be able to:

- Create cartridges to add reusable functionality to a site
- Use pipelines to add business logic to a site
- Use JS controllers to add business logic to a site
- Create reusable code using ISML templates
- Use Demandware Script in ISML templates and script files
- Use content slots to improve the appearance and flexibility of a site
- Use the Forms Framework to control the validation, rendering, and storing of consumer-entered values

Module Objectives

The following table describes the objectives for each module:

Module	Objectives
Getting Started	▪ Create a new empty site. ▪ Import a copy of SiteGenesis into a site and configure its settings.
UX Studio Overview	▪ Use UX Studio to create a new workspace. ▪ Set up a server connection.
Cartridges	▪ Describe what a cartridge is, its directory structure, and its path in Business Manager. ▪ Create an empty cartridge. ▪ Create a new storefront cartridge.
Pipelines	▪ Describe what a pipeline is, the pipeline dictionary, and the elements in a pipeline. ▪ Create a pipeline that includes: start, interaction, call, and jump. ▪ Use pipelets within pipelines.

	▪ Execute and troubleshoot pipelines.
Internet Store Markup Language (ISML)	▪ Use ISML tags and conditional tags in templates. ▪ Use local and remote includes in ISML.
Content slots	▪ Create content slots for products and images. ▪ Use rendering templates with content slots. ▪ Configure content slots.
Demandware Script	▪ Describe the Demandware Script syntax. ▪ Describe the Demandware Script API packages. ▪ Use Demandware Script in ISML. ▪ Write custom Demandware Script to create a new script pipelet. ▪ Debug Demandware Script in UX Studio. ▪ Use the Resource API and resource bundles.
Forms Framework	▪ Describe the concepts and usage of the Demandware Forms Framework. ▪ Create a new form and implement it in a pipeline.
Custom Objects	▪ Define custom objects and create instances programmatically. ▪ Use a transactional pipelet to save the custom object in the database. ▪ Implement custom logging to allow debugging and error messages to be written to logs.
Data Binding and Explicit Transactions	▪ Use data binding to pre-fill forms and update persistent data from the form ▪ Use an explicit transaction to commit changes to the database
Site Maintenance	▪ Use the Pipeline Profiler and implement page caching. ▪ Replicate code and data in the Primary Instance Group (PIG).
JS Controllers	▪ Describe the purpose and benefits of JS controllers ▪ Describe the cartridge path and folder structure for JS controllers ▪ Explain the alternative to the pipeline dictionary when using JS controllers

	▪ Describe the use of the require statement ▪ Differentiate between exporting a function and using guards ▪ Handle transactions both implicitly and explicitly ▪ Call a model from a controller ▪ Differentiate between rendering templates and forms ▪ Explain the different methods of caching using JS controllers

demandware move faster, grow faster.

Module 1: Demandware Architecture

Learning Objectives

After completing this module, you will be able to

- Describe the key components of the Demandware platform including: Realms, Primary Instance Groups, and the SiteGenesis reference storefront application.
- Create a new empty site.
- Import a copy of SiteGenesis into a site and configure its settings.

Introduction

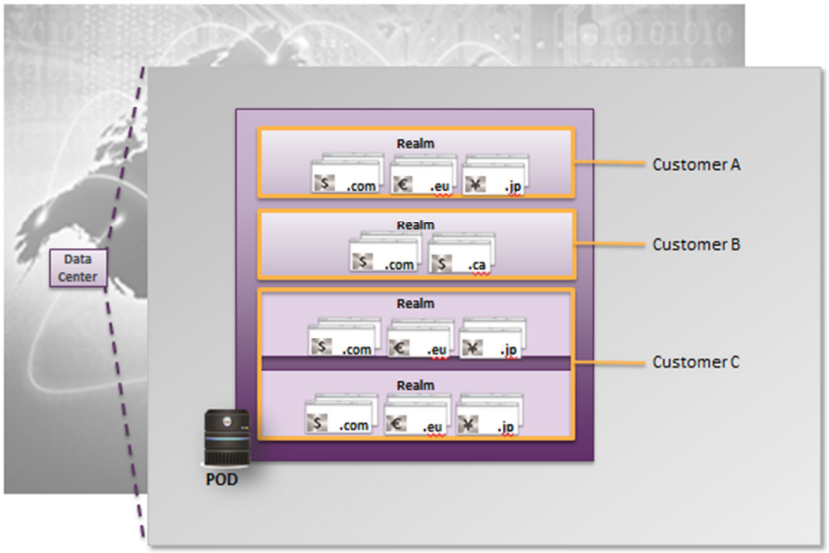
Demandware is a Software as a Service (SaaS) platform specializing in eCommerce where customers pay for applications hosted on the Internet. Software updates occur regularly on the hosted platform without customer intervention required. Using cloud technology, Demandware provides an on-demand platform architecture that is designed to handle traffic spikes by automatically allocating additional capacity as needed.

Lesson 1.1: Demandware Platform Concepts

Point of Delivery (POD) and Realms

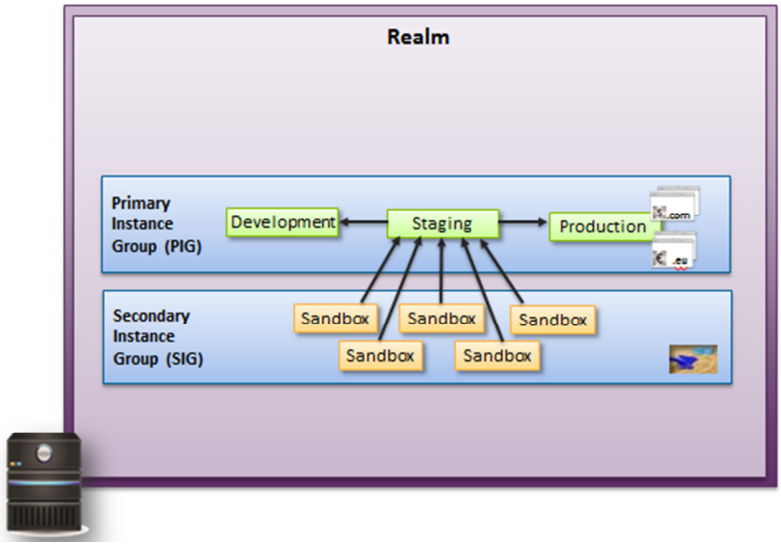
A Point Of Delivery (POD) refers to a self-contained Demandware hardware cluster that spawns a large number of virtual customer instances in a multi-tier service environment. It contains multiple application servers, database servers and the clustering and backup infrastructure. Each POD is hosted in a third-party data center.

Demandware partitions PODs into realms, which are associated with a given customer. A realm can contain one or more eCommerce sites (a.k.a, storefronts). A typical merchant using the Demandware platform will operate in a single realm. Within that one realm, the merchant can develop, stage, and deploy multiple sites. Each Demandware customer is allocated one or more realms. A realm is a collection of resources which are used for developing, testing, and hosting one or more Demandware eCommerce sites.



Instances and Instance Groups

A realm contains segmentation for development, staging, and production for one or more storefronts. Every realm contains instances located either in the Primary Instance Group (PIG) or a Secondary Instance Group (SIG).



Primary Instance Group (PIG)

Every Realm includes a Primary Instance Group (PIG) which includes three Demandware instances:

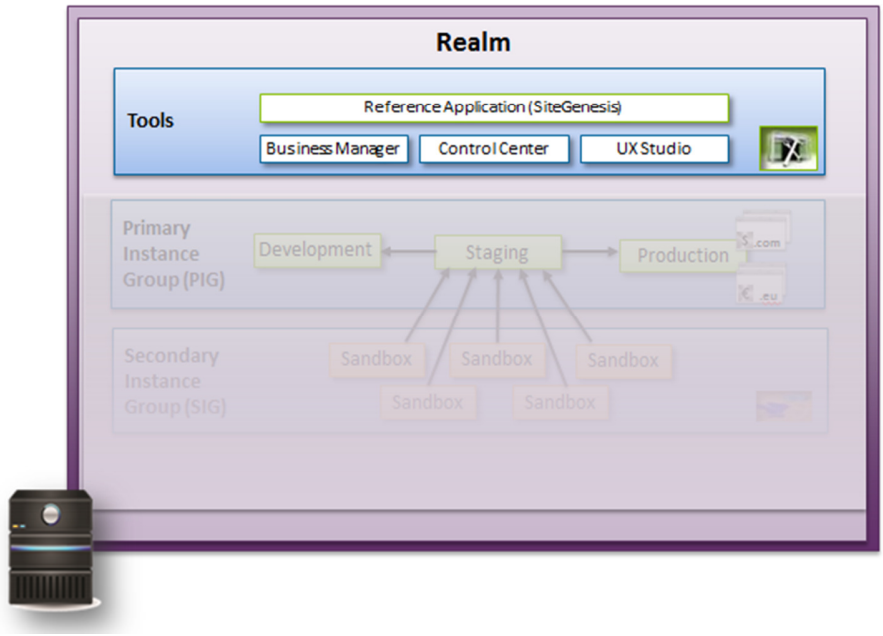
- **Production** – this is the live instance used as the actual eCommerce storefront.
- **Staging** – use this instance for configuration, data enrichment, data import, and uploading of code to prepare it for testing in the Development instance. Through data replication you can move data from the staging instance to either the development or the production instance.
- **Development** – this instance is the location where developers can test processes without impacting the production storefront (i.e. Product import feed)

Secondary Instance Group (SIG)

Every Realm includes a Secondary Instance Group (SIG) which includes five Demandware Sandboxes (but can accommodate more). Developers use sandboxes to develop and test code. They are not as powerful as PIG instances in terms of performance, memory and storage. However, they have a smaller footprint.

Lesson 1.2: Platform Tools – Business Manager

The Demandware platform contains several tools to assist in the development, configuration, and administration of sites.



Business Manager is a tool used by both merchants and developers to manage administrative tasks on the platform. Every Demandware instance has a Business Manager portal. For example, a merchandiser would log into the Business Manager portal in the Staging instance.

Merchandisers use Business Manager to manage…	Developers use Business Manager to manage…
▪ Products & Catalogs ▪ Content ▪ Marketing campaigns ▪ Search settings ▪ Customers ▪ Site Analytics ▪ Site URLs ▪ Site Preferences	▪ Code & Data Replication ▪ Code Versioning ▪ Site Development ▪ Data Import/Export ▪ Global Preferences for all sites /organization

demandware move faster, grow faster.

Exercise: Business Manager Organization

1. In Business Manager, click on each of the Merchant menu items in SiteGenesis to determine the main tasks in Business Manager. These are located on the left side under **Site – SiteGenesis**.
2. Click on the **Administration** menu items to determine the main tasks in Business Manager.

Deploying a Storefront

When you first log into Business Manager for a given instance, by default no storefront has been deployed. You must either:

- Create a new empty site (which contains no data).
- Import an existing site, such as SiteGenesis.

Exercise: Create an Empty Site

1. In Business Manager, go to **Administration > Sites > Manage Sites**.
2. Click **New**.
3. Enter a site **ID**. In this case, use “Training”. The ID is required and should not include spaces.
4. Enter a **Name.** In this case, use “Training”. The Name is required and can be any text string.
5. Click Currency drop-down and select the site’s default currency. You can only have one default currency per site. This is a required field.
6. Click **Apply**.
7. At this point, you can view or configure your new site.
   a. To select the site, click the site name, **Training**, located on the site list.
   b. Select **Site > Storefront**. A browser window opens, displaying the storefront URL.

Lesson 1.3: SiteGenesis Foundation Architecture

Demandware provides a sample reference site, called SiteGenesis, which you can use as the basis of your own custom sites. It is a full featured demonstration eCommerce site, which you can use to explore the Demandware platform and its capabilities. SiteGenesis is a resource for both developers and merchants:

- For developers, it provides sample code—pipelines, scripts, and ISML templates—which you can leverage.
- For merchants, it provides sample configurations for catalogs, categories, products, and so on.

Note: To get the latest version of SiteGenesis, import the read-only SiteGenesis package as a sample site in every Sandbox instance.

Caution: Never Import SiteGenesis into a Primary Instance Group (PIG) Instance!

You can **import SiteGenesis into each instance in your SIG**.

- If you import SiteGenesis into a sandbox that contains other customized sites, you may overwrite existing attributes and lose data. Therefore, you should not import SiteGenesis in this case.
- It is safe to **import SiteGenesis into an empty sandbox**.
- If you also want to import custom sites into the empty sandbox, import SiteGenesis **before importing the custom sites**. This ensures that you retain your site's custom attributes if there are conflicts because these custom attributes will overwrite those imported for SiteGenesis. If this occurs, the SiteGenesis site might not function properly, but your custom attribute data is kept intact. After importing SiteGenesis, you can validate its behavior by comparing it to the site running on the dedicated instance.

demandware move faster, grow faster.

Exercise: Import SiteGenesis from a SiteGenesis Package

Import the latest version of SiteGenesis

1. Log into Business Manager.
2. Select **Administration** > **Site Development** > **Site Import & Export**.
3. Determine if you want to import the site from a local or a remote instance and select the corresponding radio button.

Import a site from a local copy

1. Select the **SiteGenesis Demo Site** (alternatively click **Browse** to retrieve another local file; then click **Upload)**.
2. A confirmation message displays. Click **OK**.
3. You can view the status of the import in the **Status** section of the page.
4. When the import has completed, Business Manager lists the new site. You will also receive an email that the job has completed.

Import a site remotely

1. Enter all required data for accessing the remote server account, including the Hostname, Login, and Password. Then click **Connect**.
2. You can view the importable files from the remote server. Select the radio button next to the name of the import file you want to use.
3. Click **Import**.
4. A confirmation message displays. Click **OK**.
5. You can view the status of the import in the **Status** section of the page.
6. When your import has completed, Business Manager lists the new site. You will also receive an email that the job is complete.

Lesson 1.4: Site Configuration

After creating an empty site or training site, you need to disable site caching in order to see your code changes immediately in the site. Developers do this in their sandboxes to avoid having the page cache take effect and prevent pages from displaying differently after they have made code changes. In production instances the cache is on by default.

You will also need to index the site in order to be able to search for products from the storefront.

Exercise: Disable caching for an Empty Site or Training Site

1. In Business Manager, select **Administration > Sites > Manage Sites.** Select your site name.
2. Select the **Cache** tab.
3. Set the **Time to Live** value to 0 and uncheck the **Enable Page Caching** setting.
4. Click **Apply**.
5. It is also recommended that you to invalidate the cache at this stage. Below you can **Invalidate Page Cache** to invalidate fully or partially.
6. Check the status of the **Training** site.

Exercise: Re-Index Search for SiteGenesis

1. In Business Manager, select the site you wish to index **(SiteGenesis)** from your site list.
2. Select **Site > Search**.
3. Click **Search Indexes**.
4. Select the top checkbox **Index Type** to select all the indexes:
5. Click **Reindex**.
6. In **Site > SiteGenesis > Site Preferences > Storefront URLs** uncheck **Enable Storefront URLs**. This enables you to see the pipeline calls, rather than just the categories.

The indexes will begin rebuilding. When complete, the status changes to *Online*.

Sharing Data between Sites

The SiteGenesis import contains data specific to that site, but some data is also shared among all sites in the organization. The SiteGenesis catalogs are available to the empty site you created earlier.

The sharing of catalogs enables you to share master catalogs (containing all products) at the organization level, and for specific site catalogs to contain categories and products to control navigation for each site.

Site catalogs can have different categories and products assigned to those categories. In short, while master catalogs define all shared products for an organization, a site catalog provides the specific category navigation and products for a site. Therefore, a site must have only one site catalog, but may have one or many master catalogs.

Even when a master catalog is shared between two sites, there are site-specific attributes such as **OnlineFlag** that allow for one product to be online in one site, but offline on another. To achieve this, the product must be assigned to one site catalog and its **OnlineFlag** turned on, while on the second site catalog the same assigned product will have the **OnlineFlag** turned off.

demandware movefaster, growfaster.

Exercise: Share a Catalog between Sites and Set a Product Offline

1. In Business Manager, click **SiteGenesis > Products and Catalogs > Catalogs**.
2. Open **Storefront Catalog – EN**. Click **Edit**.
3. To share this catalog with the Training site:
   a. Select the **Site Assignments** tab.
   b. Check the **Training** site.
   c. Click **Apply.**
4. Select **Training > Search > Search Indexes**.
5. Check the top index box to select all indexes.
6. Click **Reindex**. This ensures that indexing occurs automatically after any changes that require it.
7. Select **Training > Products and Catalogs > Products.** Find the P0048 product.
8. **Lock** the product for editing.
9. Locate the *Online/Offline* site attribute. Set the **Online** field to **No** for the Training site only.
10. Apply your changes and verify that the product is not available on the Training site. (Go to SiteGenesis and search for P0048).

Knowledge Check

Question	True	False
A Realm is a Demandware instance used only by developers.		
Merchants use Business Manager to manage products and catalogs.		
You can import SiteGenesis through site import at any time without risk.		
Catalogs are not shareable between sites within an organization.		
A site must have one and only one site catalog.		

Enter item number from Column B that matches the item in Column A

Column A		Column B	
	Sandbox instance	1.	Is a customer's live storefront
	Production instance	2.	Used for code testing

demandware
A Salesforce Company

Module 3: Cartridges

Learning Objectives

After completing this module, you will be able to

- Describe what a cartridge is, its directory structure, and its path in Business Manager.
- Create an empty cartridge.
- Create a new storefront cartridge.

Introduction

A cartridge is a directory structure that provides a flexible deployment mechanism for customized functionality. It can contain many different types of files including: static files (CSS, Javascript, etc.), image files, WSDL files, etc. It also contains folders for Demandware specific files, such as: pipelines, scripts, templates, and form definitions.

A cartridge is fully contained in one directory. Every cartridge has specific sub-directories where certain file-types must be stored. For instance, you must store all Demandware script files a **scripts** folder.

Note: UX Studio generates the required `storefront.properties` file when you create a new cartridge.

Lesson 3.1: Cartridge Path

In order for a site to use a cartridge, you must add the cartridge to the cartridge path in Business Manager. This is located in **Sites > Manage Sites > Site Genesis – Settings**.

When a call is made to a file, the Demandware server looks for the file starting with the first cartridge listed in the cartridge path. For instance, if a call is made to the `productfound.isml` file and that file is located in two cartridges that are both in the cartridge path, the Demandware server uses the first one it finds.

Exercise: Add a Cartridge to a Cartridge Path

1. In Business Manager, select **Administration > Sites > Manage Sites**.
2. Select the site where you want to add the cartridge. In this case, select **SiteGenesis**.
3. Select the **Settings** tab.
4. Enter the name of the cartridge to add.

   To add multiple cartridges, use a colon between cartridge names. In this case, delete the existing path completely and add the following path:

   ```
   training:storefront_pipelines:storefront_core:storefront_controllers
   ```

   Note: All names are case-sensitive and must match your cartridge name if they exist in Eclipse. There should be no spaces between each item.
5. Click **Apply**.

demandware
A Salesforce Company

Lesson 3.2: Cartridge Types

You can create three types of cartridges in UX Studio. Your business needs determine the type that you will create. Every new customer will have at least one storefront cartridge.

Cartridge Type	Description
Demandware Storefront Cartridge	A new storefront cartridge contains a copy of the default SiteGenesis cartridge available in the SiteGenesis Demo Site package. Most projects start with this SiteGenesis reference code.
Demandware Cartridge	Use to build site-specific, re-usable functionality when there are multiple sites in a production instance. You may want to add a new cartridge when functionality is: ▪ Generic: reusable code used in multiple sites. ▪ An integration to an external system. ▪ Specific to a localized site: CSS, images and resource files for a language-specific site.
Demandware Business Manager Extension Cartridge	Not covered in this course.

Best Practices

- Keep an original SiteGenesis cartridge in your project for comparison.
- Use a storefront cartridge for common code that you intend to reuse in multiple sites: `<client>_core`
- Create cartridges for site-specific functionality that might overwrite the core: `app_<site>`
- Place any integration code in a `int_<site>` cartridge.

Exercise: View WebDAV cartridge Directory in Business Manager

1. Log into the Business Manager instance where you want to view cartridge contents (i.e.: staging instance).
2. Select **Administration > Site Development.** Click **Development Setup**.
3. In the **WebDAV Access** section, click the Cartridges link.
4. The Authentication dialog displays. Enter your Business Manager username/password.
5. Click **OK**.
6. Click on the link that corresponds to the code version that you want to view.
7. Click a version to see the uploaded cartridges.

Exercise: Create a New Version on the Server

1. In Business Manager, select **Administration > Site Development.** Click **Code Deployment.**
2. Click **Add** to create **version2**. Click **Apply.**
3. Click the new version. Notice that the **Cartridges** directory is empty.
4. In UX Studio on the connection project, select **Demandware** > **Change Upload Staging Directory…** menu. The Change Upload Staging Directory dialog displays.
5. Select **version2** from the dropdown.
6. Click **OK.** Wait for the cartridges to upload.
7. In Business Manager, check the **version2.** Note the contents of the Cartridges directory.
8. In the **File Filter** field, enter a filename and click **Find** to see all versions of it.
9. Click **Activate** to make **version2** active.

At this point, any new cartridges are uploaded to version2, which is also the active version on the server.

Lesson 3.3: Create a New Cartridge

When you need to segregate code between sites, you may want to create a new empty cartridge. This enables you to add only the code you need for a site (or specific sites) in an organization.

demandware
A Salesforce Company

Module 4: Pipelines and JavaScript Controllers

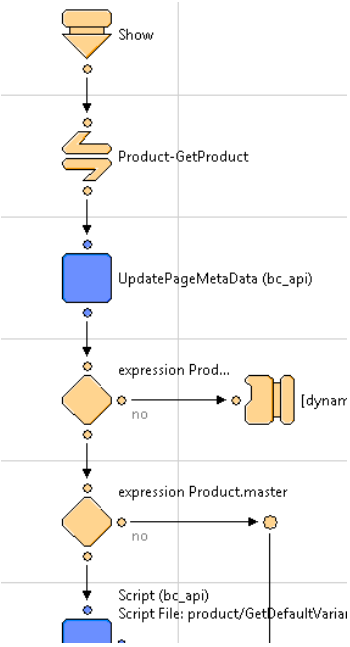
Learning Objectives

After completing this module, you will be able to

- Describe what a pipeline is, the pipeline dictionary, and the elements in a pipeline.
- Create a pipeline that includes: start, interaction, call, and jump.
- Use pipelets within pipelines.
- Execute and troubleshoot pipelines.
- Create, execute and troubleshoot Javascript controllers.

Introduction

A pipeline is a logical model of a particular business process, similar to a flowchart. Demandware UX Studio provides a visual representation of the process within the Eclipse IDE. The following example shows the Product-Show pipeline that renders the product detail page on the SiteGenesis site.



Pipelines are stored in XML files in the file system, both locally and on the server. You define and store pipelines within the context of a cartridge.

When the storefront application references a pipeline in a cartridge, it searches for the pipeline in the cartridge's path and uses the first one it finds. When a pipeline with the same name exists on two cartridges, the first one found in the path is used. Therefore, use unique pipeline names to ensure that the framework locates the correct pipeline.

There are many pipeline elements available as shown in the following table. Each node has a specific function.

demandware
A Salesforce Company

JavaScript Controllers

Beginning with the 16.1 platform release JavaScript controllers can be used in place of pipelines. Similar to pipelines, you need to develop code in a folder named "controllers" within the "cartridge" folder. In the development hierarchy **controllers take precedence over pipelines**. For example, if you have a pipeline and controller named "`HelloWorld`," the controller will be executed.

Why were controllers created?

- Server-side JavaScript controllers as an **alternative to** Demandware pipelines
- Enables greater developer efficiency and collaboration by reducing **pipeline-related pain points**
- Allows new projects based on SiteGenesis to be pipeline-free from project start
- Allows developers to build sites with Demandware using **a common, open technology**
- Eliminates **reliance on UX Studio** – providing developers a choice to use whichever IDEs they like
- Uses **current URL scheme** to leverage existing SEO features
- **Script Profiler** updated to mimic the Pipeline Profiler

Cartridge folder structure

Cartridges can contain either controllers and pipelines or controllers alone. Controllers must be located in a controllers folder in the cartridge, at the same level as the Pipelines folder. If you have controllers and pipelines in the same cartridge, and they have the same name, the platform uses the controller and not the pipeline. Indeed even if they are in different cartridges and if they have the same name, then the platform uses the controller in the path and not the pipeline.

Note: If your subpipeline is not named in accordance with JavaScript method naming conventions, you must rename it to selectively override it. For example, if your subpipeline start node is named 1start, you must rename it before overriding it with a controller method, because the controller method cannot have the same name and be a valid JavaScript method.

```
<cartridge>
        +-- modules
        +-- package.json
        +-- cartridge
        +-- controllers (the new JavaScript controllers)
        +-- forms
        +-- pipelines
        +-- scripts
        +-- static
```

What does a JavaScript Controller look like ?

```
var guard = require('storefront_controllers/cartridge/scripts/guard');
var ISML = require('dw/template/ISML');

function start() {

          ISML.renderTemplate(
                                'helloworld1.isml', {myParameteronISML:"Hello from
Controllers"}
```

```
                                    );

};
exports.Start = guard.ensure(['get'], start);
```

The ‘require’ keyword is used to import a class from API package to be used in the code.

The guard exposes the function with a new name. In this case the function ‘start’ is exposed to the URL with the name ‘Start’. It can also enforce a http method (In this case it is exposing the function with a ‘get’ method)

The ISML object is used to give the control to ISML which can display the results. We can use response object directly to display the results as shown below but it is not recommended.

```
response.setContentType('text/html');
response.getWriter().println('<h1>Hello World from Javascript controllers!</h1>');
```

demandware
A Salesforce Company

Exercise: Create a JHelloWorld JavaScript Controller

1. In the Business Manager, navigate to **Administration>Sites>Manage Sites>Site Genesis>**Settings.
2. In case not already there, add `storefront_controllers` cartridge to your cartridge path. Your cartridge path should now look something like:

   `training:storefront_controllers:storefront_pipelines:storefront_core`
3. Upload your cartridge to the Sandbox as shown below:

   Go to Eclipse->Right click **Demandware Server>Properties>Project References** and check `storefront_controllers` cartridge.
4. Create a new controller named `JHelloWorld.ds` (right click on controllers->New file)
5. Copy and paste the following structure to create a start function and use ISML and guard

```
/**
* A hello world controller.  This file is in cartridge/controllers folder
* @module controllers JHelloWorld
*/

var guard = require('storefront_controllers/cartridge/scripts/guard');
var ISML = require('dw/template/ISML');

function start() {

};
exports.Start = guard.ensure(['get'], start);
```

6. Inside the function named start, add the following code to render the control to ISML:

```
ISML.renderTemplate(
       'helloworld1.isml', {
                              myParameteronISML:
                            "Hello from Controllers"
                          }
                    );
```

7. Navigate to templates/default and create and ISML named `helloworld1.isml` under `templates/default` folder with the following code in it (We will later learn what pdict means)

```
${pdict.myParameteronISML}
```

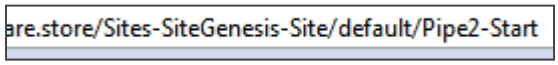
8. Run the controller as below:
9. Navigate to storefront
10. At the end of the url add `/default/JHelloWorld-Start`
11. Press enter key to execute the controller

Troubleshooting with the Request Log Tool

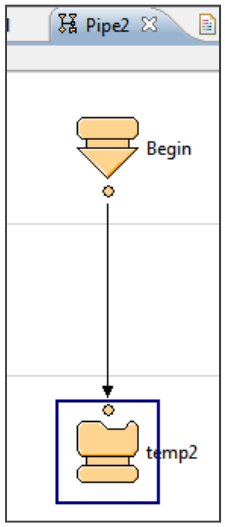
The Request Log tool enables you to troubleshoot error messages on your storefront. It is part of the Storefront Toolkit, which is available in all instances (except Production).

The Request Log tool displays the log for the last request and any request prior to that request during the current session. You can view both debug messages and error messages.

Common errors include typographical errors. In this example, a call was made to a pipeline named **Pipe2** with a start node called **Start**.



However, the name of the start node is **Begin**.



The request log displays the error “Start node not found (Start)”

Exercise: Create a JavaScript Controller named JCall.js with Your Instructor

1. Create a controller named `JCall.js` in controllers folder.
2. Use quickcard (section "Import an Object") to require `ISML` and `guard` in your controller.
3. Use quickcard (section "Function declaration and exposure") to declare the function and expose `start` function as `Start`
4. Inside the `start` function, past the template below:

```
	var myParam =
/* use quickcard section "Dealing with query parameters" get the Parameter named param */

	if (myParam.stringValue != null)
		{
 /* use quickcard section "Giving control to ISML" to give control to call/notEmpty.isml
and loading myParam on a variable paramOnPdict
 */

		}
	else{
			ISML.renderTemplate(
			 'call/jempty.isml', {paramOnPdict:'param not found'}
			);

	};
```

5. Follow the instructions in the template comments above to complete the code.
6. Create two templates `jnotEmpty.isml` and `jempty.isml` (under `templates/default/call`) that display different messages and assign them to the Interaction nodes.

   The successful path should have an ISML code in `jnotEmpty.isml` as shown:

   ```
   Got the parameter ${pdict.paramOnPdict.stringValue}
   ```

   The path which does not have parameter () should have ISML code in `jempty.isml` as shown:

   ```
   Could not find the parameter!
   ```

7. Execute the code by Navigating to storefront and adding `/default/JCall?param=1234` to the url at the end.
8. Try not providing the query parameter.

Jump Nodes

A Jump node invokes a specified sub-pipeline. After the sub-pipeline's execution, the workflow does not return to the calling pipeline. It is the responsibility of the sub-pipeline to complete the task.

- `CurrentVersion`

The pipeline dictionary is passed across sub-pipeline calls. Whenever a call or jump to another pipeline is executed, the same `pdict` is passed to the invoked sub-pipeline.

To view the values stored in the pipeline dictionary at run-time, run a pipeline from the storefront while in a debug session (covered next).

Pipeline Dictionary to global variables

JavaScript controllers **can use alternatives to pdict variables. Here are some of them.** Strikethroughs are implicit packages or classes in JavaScript controllers.

pdict keys	Alternatives
`CurrentSession`	~~`TopLevel.global`~~`.session`
`CurrentRequest`	~~`TopLevel.global`~~`.request`
`CurrentCustomer`	~~`TopLevel.global`~~`.customer`
`CurrentHttpParameterMap`	~~`TopLevel.global`~~`.request.httpParameterMap`
`CurrentPageMetaData`	~~`TopLevel.global`~~`.request.pageMetaData`
`CurrentForms`	~~`TopLevel.global`~~`.session.forms`

In other words, controllers have access to `request, response, session, customer` objects if you have used the valid import or require statements. They also have access to `CurrentHttpParameterMap` variable using `request.httpParameterMap` and `pageMetaData.`

importPackage vs. require

In previous versions of SiteGenesis, the `importPackage` statement was always used to import Demandware packages into your scripts. With controllers, you can also use require to import Demandware script packages.

For example: `require('dw/system/Transaction')`

Demandware **recommends** using the require method to import Demandware script packages, or other JavaScript or Demandware script modules instead of `importPackage`.
Unlike `importPackage`, which must be added at the beginning of a script, you can require a module anywhere in your script, allowing you to only load functionality if you need it. This can result in substantial improvements in performance.
Note: You will still see `importPackage` in code examples so that you understand the code in case you are reading older code.

Passing Parameters

You can pass parameters to the pipeline dictionary using a URL query string:

```
/default/Call-Start?param=1234
```

demandware
A Salesforce Company

Exercise: Create JavaScript Controller JShowProduct

1. Create a new JavaScript Controller called `JShowProduct.js`.
2. Copy and paste the following template to it

```
'use strict';

/** @module controllers/JShowProduct */

var ISML = require('dw/template/ISML');
var guard = require('storefront_controllers/cartridge/scripts/guard');

function start() {

}
exports.Start = guard.ensure(['get'], start);
```

3. Use the 'require' syntax to import `ProductMgr` class from `dw.catalog` package right after the guard
4. Inside the start() function paste the following code to get the parameter 'pid' from the url

```
var parameterMap = request.httpParameterMap;
 var parameterId =parameterMap.<parameter name>.stringValue
```

5. Get the product from ProductMgr as shown below

```
var product = ProductMgr.getProduct(parameterId);
```

6. Copy and paste the following code to forward the control to ISML.

```
if (product===null) {
      ISML.renderTemplate(
                  'productnotfound.isml', {message:'product with id
'+parameterId+' not found'}
                 );
    }
    else{
      ISML.renderTemplate(
                  'productfound.isml', {myProduct:product}
                 );


  }
```

7. If not already created, create `templates/default/productnotfound.isml` with the following code in it.

```
${pdict.message}
```

8. If not already created, create `templates/default/productfound.isml` with the following code in it.

demandware
A Salesforce Company

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.

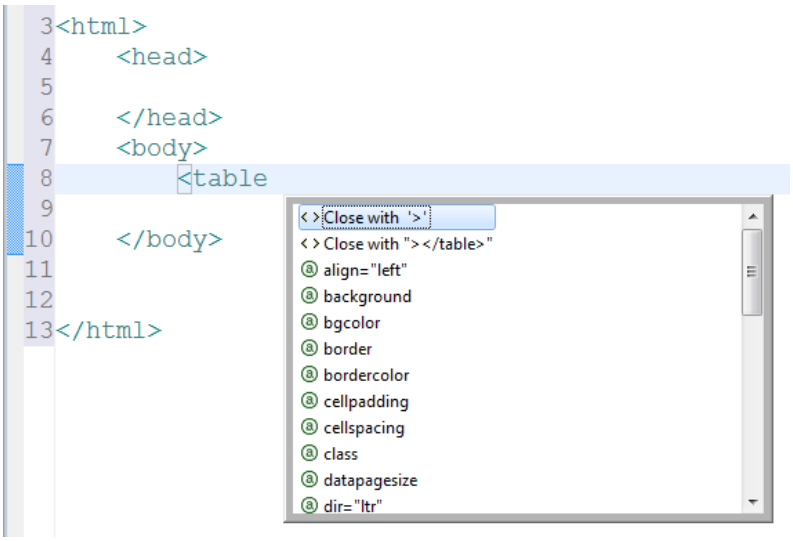
When describing a Demandware application using the Model-View-Controller (MVC) pattern, templates represent the view, pipelines represent the controller and the DW Script API represents the model.

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.



demandware
A Salesforce Company

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.

Exercise: Set and Retrieve Variables

1. Create a new pipeline or a JavaScript Controller (using quickcard as a guide) called `VarTest`. Add a **Start** node and an **Interaction** node to the pipeline.

   Note: If you are using `JavaScript` controller, you can use JVarTest.js from `jsolutions` cartridge also.

2. Create a new ISML template called `vartest`.
3. In the template, create a new variable called `sessionVar` with hardcoded text as the value and print the value to the page:

   ```
   <isset name="sessionVar" value="${1}" scope = "session"/>
   ```

4. Display the contents of the `sessionVar` variable. Its value is:

   ```
   ${session.custom.sessionVar}<br/>
   ```

5. Open a web browser and test the pipeline.
6. Add similar examples of request and page variables to the `vartest` template and display them.
7. Modify the examples using boolean and string values (i.e., "`${false}`" and "`Hello`").
8. Test the pipeline again to see the new variable values.
9. Increment the variables by using the following syntax:
   `"${request.custom.requestVar + 1}"`
10. Explain your findings.

demandware
A Salesforce Company

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.

demandware
A Salesforce Company

Exercise: Use Local Includes in JShowProduct JavaScript Controller

1. Study the template to be included:
   a. Locate and study the `producttile.isml` template.
   b. Notice that the first `<isset>` tag expects `pdict.product` in the pipeline dictionary.
2. Include `producttile.isml` in your current template:
   a. Open the `JShowProduct` controller from the last chapter.
   b. Note that you are outputting `myProduct` object on `pdict`, however `producttile` template expects `pdict.product`. These are not the same variables!
   c. Open the `productfound.isml` template from the `JShowProduct` pipeline you created earlier.
   d. Create a pipeline dictionary variable that matches the variable and scope expected in the `producttile.isml` template:

```
<isset name="product" value="${pdict.myProduct}" scope="pdict"/>
```

   e. Use a local include to display the product tile:

```
<isinclude template="product/producttile"/>
```

   f. Test the pipeline with an existing product: `JShowProduct-Start?pid=P0048`.

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.

demandware
A Salesforce Company

Exercise: Use Remote Include in JShowProduct JavaScript controller

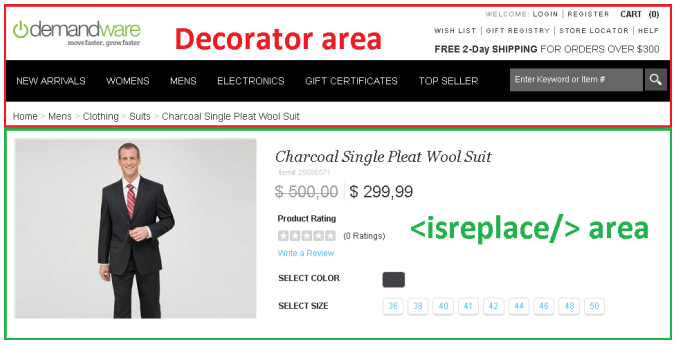
1. Study the Demandware Script API help for the `dw.web.URLUtils` class, `url()` method.
2. Locate and study the `Product-IncludeLastVisited` (look for `Product.js`) controller in the storefront cartridge.
3. Study the `lastvisited.isml` template, specifically:
   c. The use of the `<isloop>` tag.
   d. The use of the `pdict.LastVisitedProducts`.
4. Open the `JShowProduct` controller and the `productfound.isml` template.
5. Add a remote include at the bottom of the template to show the last visited products. Verify your syntax to make sure it is exactly as it appears below:

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

6. Test the controller with an existing product: `JShowProduct-Start?pid=P0048`.
7. On a different browser tab, visit at least three other products in the storefront.
8. Retest the controller: all the visited products should display.

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>
```

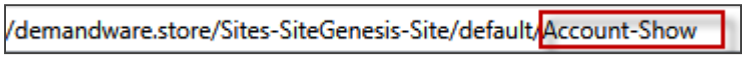
Using the <isdecorate> Tag

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
A Salesforce Company

Exercise: Use a Decorator in ShowProduct Pipeline or JShowProduct JavaScript controller

1. In UX Studio, using the Search function, locate the `product/pt_productdetails` template. Notice the different areas of the page this decorator defines.
2. Locate the `<isreplace/>` tag.
3. Open the `ShowProduct` pipeline or `JshowProduct` that you created earlier.
4. In your `productfound.isml` template, remove any html, body and head tags as the decorator already contains these.
5. Add the `product/pt_productdetails` decorator so it wraps the existing content on the page:

```
<isdecorate template="product/pt_productdetails">
...existing content...
</isdecorate>
```

6. Test the pipeline or the controller with an existing product: `ShowProduct-Start?pid=P0048` `(or JShowProduct-Start?pid=P0048)`

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"
/>
```

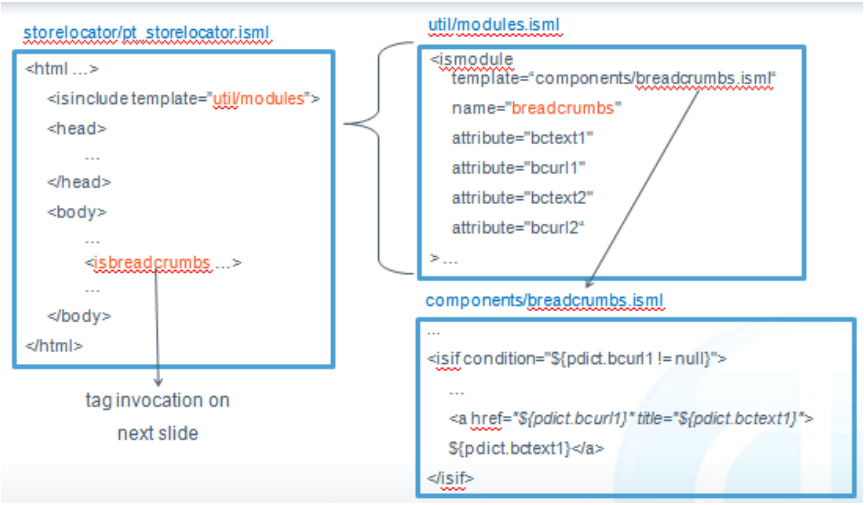
2. The ISML file which specifies what happens when the attributes are passed. See the code snippet from inside `breadcrumbs.isml`:

```
<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.



demandware
A Salesforce Company

Exercise: Use a Custom Tag in JShowProduct controller

In this exercise, you will invoke a custom tag already created in SiteGenesis.

1. Open the `util/modules.isml` template.
2. Locate the `producttile` custom tag definition. Note the different inputs defined for the `producttile` custom tag.
3. Locate the template that implements this custom tag, and study it: `producttile.isml`.
4. Open the `productfound.isml` template that you were referring to from `JshowProduct` controller.
5. Remove the remote include.
6. Change the existing local include to include the template that contains all custom tag definitions:

   ```
   <isinclude template="util/modules">
   ```

7. Invoke the `<isproducttile>` custom tag passing the product from the pipeline dictionary:

   ```
   <isproducttile product="${pdict.myProduct}"/>
   ```

8. Test the controller with an existing product: `JShowProduct-Start?pid= P0048`.
9. In the custom tag invocation, enable other attributes expected by the custom tag. Notice that a proper Demandware script expression is required to specify true or false:

   ```
   <isproducttile product="${pdict.myProduct}" showswatches="${true}"
   showpricing="${true}" />
   ```

10. Test the `JShowProduct` controller.

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 t`o 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>
```

demandware
A Salesforce Company

Exercise: Creating JBasket JavaScript Controller and Using Loop in ISML

1. Please visit the script API (Instructor will help you with this and visit the package `dw.order`)
2. In this package visit the class named `BasketMgr` and property named `currentBasket`. Study what it does. We are going to use it in our code.
3. Create a JavaScript controller named `JBasket.js`
4. Copy and paste the template below and complete instructions in the comment.

```
var ISML = /* get ISML object from dw.template package */
var guard = require('storefront_controllers/cartridge/scripts/guard');
var BasketMgr = /* get BasketMgr from dw.order package */

function start() {

        var basket=BasketMgr.currentBasket;

    /*use ISML to display basket on myBasket.  The rendered ISML should be
showBasket.isml (Use quickcard section "Giving control to ISML" for help*/


    }
exports.Start = guard.ensure(['get'], start);
```

5. Create an ISML named `showBasket.isml` under `templates/default` folder
6. Copy and paste the following code in the ISML to display the contents of the basket.

```
<br/>
<isloop
items="${pdict.Basket.allProductLineItems}" var="productLineItem">
${productLineItem.product.name}<br/>
</isloop>
```

7. Open a browser to your storefront. Add products to the cart first, including a product with an option (like a TV warranty).
8. Open another browser tab and invoke the `Basket-Start` pipeline.
9. Add a `status` attribute in the `<isloop>` tag so that you can see what the `count` and `index` parameters return.
10. Replace the `allProductLineItems` property of the basket with the method `getProductLineItems()`.
11. Execute the code(Navigate to storefront>add /default/JBasket-Start at the end of the url)

demandware
A Salesforce Company

Module 6: Content Slots

Learning Objectives

After completing this module, you will be able to:

- Create content slots for products and images.
- Use rendering templates with content slots.
- Configure content slots.

Introduction

A content slot is an area on the page where a merchant defines content to display based on certain qualifiers or rules.

To view a content slot, use the **Storefront Toolkit > Content Information** tool. Hover the mouse pointer around the page to reveal where content slots exist and to access a link to the slot's configuration page in Business Manager.

There are three contexts of slots:

- Global slots can appear on any page.
- Category slots appear on category-specific pages since they depend on the category ID.
- Folder Slots – appear in content library folders dependent on the folder ID.

A content slot is used to show different types of content:

- One or many products selected by the merchant
- Category attributes (images or other visual)
- Content assets from the content library
- Static HTML and images from the static library

There are many rules that drive the appearance of a slot: marketing campaigns, ranks, AB tests, customer groups, etc. Campaigns and A/B testing are out of the scope of this course.

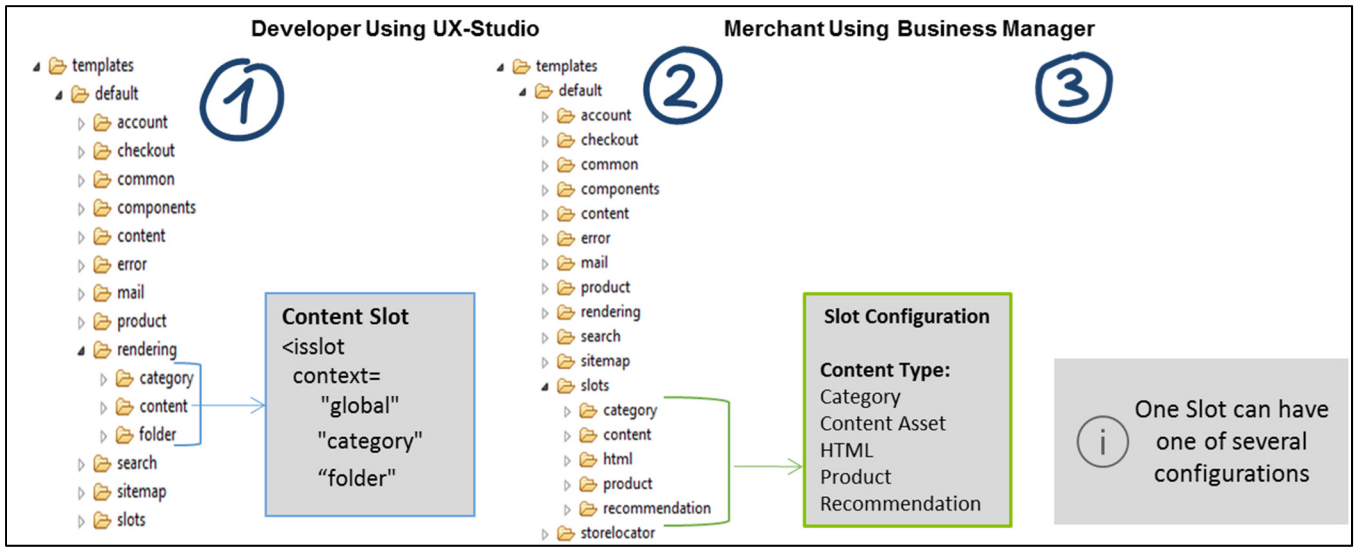
Content Slots vs. Content Assets

Slots are controlled by campaigns: start/end dates, customer groups, source codes, coupons and rank are qualifiers that affect the appearance of a slot. Content Assets are reusable elements that do not have qualifiers. Content slots and content assets are managed in different areas within Business Manager. Slots are a marketing tool, therefore configuration information for content slots reside in **Site > Online Marketing > Content Slots;** content assets are in the Content module.

Content Slots

Creating a content slot requires a collaborative effort:

1. The developer inserts a `<isslot>` tag in a template in the location where the slot will appear.
2. The developer creates a rendering template for the slot that defines how the slot data is to be presented.
3. The merchant creates a configuration for the slot in Business Manager.



demandware
A Salesforce Company

Lesson 6.1: Creating & Configuring Content Slots

Creating a content slot requires a collaborative effort:

1. The developer inserts a `<isslot>` tag in a template in the location where the slot will appear.
2. The developer creates a rendering template for the slot that defines how the slot data is to be presented.
3. The merchant creates a configuration for the slot in Business Manager.

Creating Content Slots - Developer Tasks

The developer creates a content slot inside a template using the `<isslot>` tag. The tag must be located exactly where it should appear on the page. Here are some examples of tag usage:

Global slot example

```
<isslot id="header_banner" description="…" context="global"/>
```

Category slot example

```
<isslot id="category_top_featured" context="category" description="…"
context-object="${pdict.ProductSearchResult.category}"/>
```

Folder slot example:

```
<isslot id="fldr-landing-slotbanner" context="folder" description="Large
Folder Landing Banner"
context-object="${pdict.ContentSearchResult.folder}"/>
```

Whenever the template is saved, the new content slot will automatically appear in the list of slots under **Site > Online Marketing > Content Slots.** The platform achieves this by automatically scanning any template for the use of the `<isslot>` tag.

Creating the Slot Rendering Template

The slot will display one type of content out of four possible types. The developer creates a rendering template that takes into account the type of content, how many objects to display, plus any CSS styling required for the slot.

The `header_banner` slot uses the `htmlslotcontainer` template as the rendering template:

```
<iscache type="relative" hour="24"/>
<div class="htmlslotcontainer">
   <isif condition="${slotcontent != null}">
         <isloop items="${slotcontent.content}"
               var="markupText">

<isprint value="${markupText.markup}"
encoding="off"/>
</isloop>
   </isif>
</div>
```

Using `slotcontent` and `<isprint>` in Rendering Templates

Every slot is rendered by a system pipeline inside the core cartridge: `_SYSTEM_Slot-Render`. You do not have access to this pipeline. It uses the slot configuration that the merchant creates and provides all the configuration information to the rendering template by means of the `TopLevel.global.slotcontent` constant. Only slot rendering templates get data via this constant.

The rendering template code checks that the `slotcontent` is not empty:

```
<isif condition="${slotcontent != null}">
 Then it loops through the slotcontent.content (the content provided for
the slot):
<isloop items="${slotcontent.content}" var="markupText">
   <isprint value="${markupText.markup}" encoding="off"/>
</isloop>
```

Inside the loop the code uses the `<isprint>` tag:

```
<isprint value="${markupText.markup}" encoding="off"/>
```

Note: For more information on the `<isprint>` tag in detail, there is extensive documentation and usage examples for it in SiteGenesis.

Using the `encoding="off"` setting enables the HTML snippet to be generated without encoding, so that the browser renders it correctly.

demandware
A Salesforce Company

Exercise: Create a Slot

Create a banner slot containing an image on the `nohits.isml` template. This template appears when a search does not return any products.

1. Use the storefront search box and search for a product which does not exist.
2. Investigate which template is used to generate that page (which tool would you use to find that out?).
3. Copy the found template from the storefront cartridge to the exact same location into your cartridge.
4. Before the `no-hits-footer div`, add a global slot:

```
<isslot id="search-no-hits-banner"
description="recommendations banner for search no results page"
context="global" />
```

5. Study the `htmlslotcontainer.isml` rendering template that is used to render HTML-type slots.

Creating Content Slot Configurations - Merchant Tasks

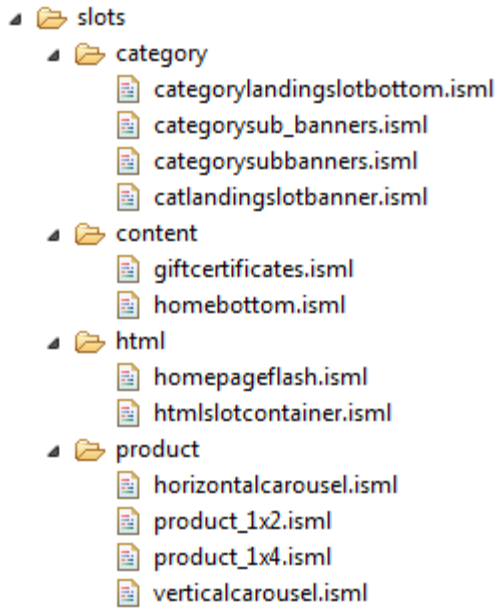
Merchants create content slot configurations by navigating to **Site > Online Marketing > Content Slots** and locating the specific slot that the developer created, e.g. `header-banner`. The merchant can select an existing configuration or click **New** to create a new one.

The merchant selects the type of content, for example, Product or HTML. Different fields display depending on the content type selected, for example:

- For a Product content slot, the **Product** field displays and the merchant enters the IDs of the products to be displayed. The merchant then selects one of the templates designed to display products from the **Template** drop-down menu.
- For an HTML content slot, an **HTML** text area displays and the merchant enters the HTML content. The merchant then selects one of the templates designed to display HTML from the **Template** drop-down menu.

The **Template** menu contains all possible rendering templates that are available in all cartridges in the cartridge path for this content type. The SiteGenesis storefront cartridge comes with default templates for every type of content. The templates are located in specially named folders that Business Manager discovers by default (for example, `slots/html` for the HTML type).

Here is the directory structure for the slot rendering templates in the SiteGenesis storefront cartridge:



The merchant can choose to reuse an existing rendering template or use a new one as instructed by the developer. This is why the collaboration between merchant and developer is important—without a rendering template, there is no way to visualize the slot.

The merchant also provides a schedule for the slot. This is either the default schedule or based on a marketing campaign.

Lesson 6.2: Using Content Link Functions

Demandware uses attributes of type HTML in many places: content assets, content slots with HTML-type content, product descriptions, etc. You can also add an attribute of type HTML to any system object where you may need to show HTML. These attributes are represented by the class `dw.content.MarkupText`.

Note: When using HTML in content assets or content slots, avoid hardcoding hyperlinks to pages or images in the storefront. They are instance-specific (e.g., Staging) and would have to be changed every time after a replication. Instead, Demandware offers the following Content Link Functions for use in attributes of type HTML:

- `$staticlink$` - Creates a static link to an image.
- `$url()$` - Creates an absolute URL that retains the protocol of the outer request.
- `$httpUrl()$` - Creates an absolute URL, with the http protocol.
- `$httpsUrl()$` - Creates an absolute URL, with the https protocol.
- `$include()$` - Makes a remote include call (relevant for caching purposes).

Here is an example of a function that creates a hyperlink to the `Page-Show` pipeline passing `cid=2-day-shipping-popup` in the query string:

```
href="$url('Page-Show', 'cid', '2-day-shipping-popup')$"
```

Exercise: Create a Slot Configuration

Complete the configuration for the content slot created previously.

1. In Business Manager, navigate to **Site > Online Marketing > Content Slots**.
2. Locate the new `search-no-hits-banner` slot in the global section.
3. Create a new configuration for the slot:
   a. Provide an ID: banner-for-everyone.
   b. Enable it.
   c. Make it the default.
   d. Select HTML for the content type.
   e. In the HTML editor:
      - Click the Insert/Edit Image icon.
      - Click **Browse Server.**
      - Locate the `/images/slot/` directory and select it.
      - On the Upload File section, find the `nohits.png` image in the `contentslot` cartridge, `static/default` folder, and upload it.
      - After uploading, select the image.
      - The generated HTML should look like this:

```
<p><img width="700" height="100" src="nohits.png?$staticlink$" alt=""
/></p>
```

   f. Select `slots/html/htmlslotcontainer.isml` as the rendering template for the slot.
4. Click **Add Schedule > Default Schedule** to ensure that the slot displays continuously.
5. Click **Apply** to save the configuration.
6. Test the slot by searching for some non-existent product: the `nohits` page should display with the new slot visible.

demandware
A Salesforce Company

Knowledge Check

Question	Answer
1. What contexts of content slots can you create?	
2. Can a slot be created in Business Manager?	
3. How does `<isprint>` preserve the markup of an HTML slot?	
4. Where can `<isslot>` be placed in templates?	

Demo: Create a Slot with a Rendering Template for a Vertical Carousel of Products

Create a content slot in the `nohits.isml` that displays some products selected by a merchant. The components involved will be: a rendering template, a style sheet, an ISML that has the content slot and finally an ISML to link to the style sheet.

1. Open the `nohits.isml` template in your cartridge. This is the page which shows up when the product that the customer searches is not found.
2. Below the `search-no-hits-banner` slot, add another global slot: `<isslot id="merchant-products" description="content for search no results page" context="global"/>`
3. Create a directory structure so that you have slots/product folder as follows.
   `training/cartridge/templates/default/slots/product`
4. Copy the `verticalcarousel.isml` from storefront to exactly the same location in the training cartridge . This is the rendering template that you are going to modify.
5. Rename this `verticalcarousel.isml` in the training cartridge to `verticalcarouselx4.isml`

6. Modify the carousel to match the following code:

```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<iscache type="relative" minute="30" varyby="price_promotion"/>

<isinclude template="util/modules"/>

<h2>${Resource.msg('global.carousel.featuredproducts','locale',null)}</h2
>

<div id="vertical-carousel">
   <ul>
   <li>
   <div class="productcarousel">
   <isloop items="${slotcontent.content}" var="product" status="status" >
   <div class="analytics capture-product-id"><isprint
value="${product.getID()}"/></div>
                    <isproducttile product="${product}"
showpricing="${true}"/>
                    <isif condition="${status.count%4==0
&& !status.last}">
                    </div>
                    </li>
                    <li>
                    <div class="productcarousel">
              </isif>
         </isloop>
         </div><!-- END: productcarousel -->
         </li>
   </ul>


   <a class="jcarousel-prev" href="${'#'}"></a>
   <a class="jcarousel-next" href="${'#'}"></a>

</div>

<!-- END: verticalcarousel -->
```

7. Create a template named `pt_productsearchresult_UI.isml` in the following location in the training cartridge: `training/cartridge/templates/default/search`

8. Add the following line to point to a new css file that redefines the vertical carousel styling.
   `<link href="${URLUtils.staticURL('/css/verticalcarouselx4.css')}" type="text/css" rel="stylesheet"/>`

9. Copy the provided `verticalcarouselx4.css` file to `/static/default/css/verticalcarouselx4.css` in your cartridge (create the directory structure if it is not there).

10. Copy the `pt_productsearchresult_nohits.isml` from the storefront cartridge into your cartridge to the same location (create the folder structure if it is not present).

The event handler for our buttons is in the Namespace named `storefront`. So modify the script block to match the following:

```
<isscript>
      var pageContext = {
            title: 'Product Search Results No Hits',
            type:'storefront',
            ns:'storefront'
      };
</isscript>
```

11. In Business Manager, select Site > SiteGenesis > Online Marketing> Content Slots.
12. Search for the `merchant-products` slot. Create a new slot configuration for this slot so that it displays multiple products using the new `verticalcarouselx4.isml` rendering template. The rendering template will have to be chosen from the training cartridge. Make sure that you add some products rather than the HTML (unlike you did in one of the previous exercise).
13. Navigate to the storefront from BM in the browser. Search for some non-existent product like `MyBestPants.`
14. Verify that the `nohits.isml` shows both the previous banner and multiple products in a vertical carousel.

demandware
A Salesforce Company

Module 7: Demandware Script

Learning Objectives

After completing this module, you will be able to:

- Describe the Demandware Script syntax.
- Describe the Demandware Script API packages.
- Use Demandware Script in ISML.
- Write custom Demandware Script to create a new script pipelet.
- Debug Demandware Script in UX Studio.
- Use the Resource API and resource bundles.

Introduction

Demandware Script (DWScript) is the server-side language used for coding in the Demandware Platform.

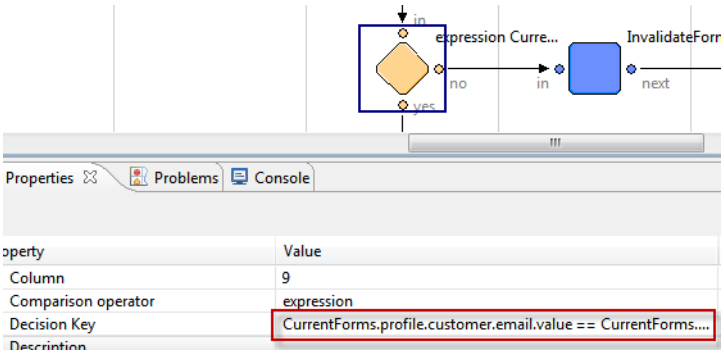
- It is based on JavaScript, which is standardized as ECMAScript. It implements ECMA-262 and the ECMA-357 standard, also known as ECMA for XML or E4X.
- It supports all JavaScript language extensions by Mozilla known as JavaScript 1.7 as well as optional type specification (from JavaScript 2.0/ECMA 4th edition proposal and ActionScript).

Use Demandware Script to access data about the system, such as: products, catalogs, prices, etc. You write DWScript in pipelines inside Decision nodes, and in ISML templates for expressions or inside `<isscript>` tags. Demandware Script is used extensively inside Script pipelets.

ISML

```
<isscript>
    var cat = pdict.ProductSearchResult.category;
    var path = new dw.util.ArrayList();
    while( cat != null && cat.parent != null )
    {
        if( !cat.online )
        {
            cat = cat.parent;
            continue;
        }
        path.addAt( 0, cat );
        cat = cat.parent;
    }
</isscript>
```

Decision Node



Lesson 7.1: Demandware Script API

Each new Demandware update includes a well-documented API. The Script and Pipelet APIs are available under the Studio Help menus. The ISML documentation is available in the Demandware documentation: https://info.demandware.com.

Demandware continually updates clients to the latest version. Deployments happen globally on Tuesday and Thursday between 2 and 7 am local POD time. The current version number appears at the bottom of the Business Manager screen and it corresponds to *Year.Deployment*, for example, version 14.4 represents the fourth deployment in 2014.

Demandware provides access to Preview releases by updating sandboxes prior to updating the PIG instances. This gives your organization an opportunity to test any new API updates and other customizations on your site prior to using that update in production. For more information, refer to the Global Release Process FAQ https://xchange.demandware.com/docs/DOC-1815.

The Global Release Process ensures that all Demandware clients stay on the same version of code and that updates containing defect corrections as well as new functionality can be applied uniformly with minimal down time.

API Packages

The Demandware Script API is organized in packages, just like Java. Unlike Java, inheritance is not possible from these classes or packages when you create a script. You can only use the properties and methods of these classes in your scripts.

In Demandware Script, the TopLevel package is the default package. It is similar to `java.lang` in Java. It does not need to be imported in scripts. It provides standard ECMAScript classes and extensions, such as: `Error, Date, Function, String, Math, Number, XML.`

The `TopLevel.global` class contains many of the common variables and constants used in pipelines and scripts, such as:

- Constants: `PIPELET_NEXT` and `PIPELET_ERROR` indicate the result of a script pipelet and determine which exit the pipeline takes after pipeline execution.

- Properties: `customer`, `request` and `session` provide access to the current customer and the current session.

Note: In the following packages there are many classes that end with `Mgr` (e.g., `dw.catalog.ProductMgr`. These classes **retrieve instances of business objects related to the package they belong to**. For example, use `ProductMgr.getProduct(String id)` to get a product using a unique identifier. The method returns a `Product` instance which you can use to find information about the product. This pattern is repeated for all Managers.

eCommerce API Packages	
`dw.campaign`	For campaign and promotions Classes: `PromotionMgr`, `Campaign`, `Promotion`, `SourceCodeGroup`, etc.
`dw.catalog`	For catalog, product, and price book Classes: `CatalogMgr`, `Category`, `Product`, `Recommendation`, `PriceBook`, etc.
`dw.content`	For non-product content management Classes: `ContentMgr`, `Content`, `Folder`, `Library`, etc.
`dw.customer`	For customer profile and account Classes: `CustomerMgr`, `Customer`, `Profile`, `ProductList`, `OrderHistory`, etc.
`dw.order`	For orders, including: basket, coupons, line items, payment, shipment Classes: `Basket`, `Order`, `ProductLineItem`, `ShippingMgr`, `TaxMgr`, etc.

Generic API Packages	
`dw.crypto`	Encryption services using JCA; DES, Triple-DES, AES, RSA, etc. Classes: `Cipher`, `MessageDigest`
`dw.io`	Input and output Classes: `File`, `FileReader`, `CSVStreamReader`, `XMLStreamReader`, etc.
`dw.net`	Networking Classes: `FTPClient`, `HTTPClient`
`dw.object`	System base classes and custom objects

	Classes: `PersistentObject`, `ExtensibleObject`, `CustomObjectMgr`, etc.
`dw.rpc`	Web services related APIs Classes: `WebReference`, `Stub`
`dw.system`	System functions Classes: `Site`, `Request`, `Session`, `Logger`
`dw.util`	Similar to the `java.util` API: `collections`, `maps` and `calendar` classes
`dw.value`	Immutable value objects Classes: `Money`, `Quantity`
`dw.web`	Web-processing Classes: `URLUtils`, `Forms`, `Cookie`, `HttpParameterMap`, etc.

Using Demandware Script in ISML

You can embed Demandware Script into ISML by using the `<isscript>` tag. The following example uses Demandware script to get the root category of a current site's navigation catalog as well as the category named 'sale'.

```
<iscomment>
    This template displays a 3-level category tree as top navigation.
    Only categories marked with showInMenu are shown.
</iscomment>

<isscript>
    // get root category of current site's navigation catalog
    var siteCatalog = dw.catalog.CatalogMgr.getSiteCatalog();
    var root = null;
    if(siteCatalog!=null) {root = siteCatalog.getRoot();}

    // get the "sale" category
    var saleCategory = dw.catalog.CatalogMgr.getCategory('sale');
</isscript>
<isif condition="${root != null}">
<div class="categorymenu">
```

Inside of the `<isscript>` tag you can fully qualify every class you want to use or you can import any packages at the top of the script:

```
<isscript>
   importPackage(dw.catalog);
   var siteCatalog = CatalogMgr.getSiteCatalog();
…
</isscript>
```

demandware
A Salesforce Company

Exercise: Calling a Script Pipelet in JShowProduct JavaScript Controller

1. You have already created the controller named `JShowProduct`. Keep a backup copy of it if you want to.
2. Copy `jsolutions/cartridge/scripts/ProductFinder.js` to your `training` cartridge in the same location.
3. In the present controller, remove all the previous code and copy and paste the following template

```
'use strict';

/** @module controllers/JShowProductCallingScript */

var ISML = require('dw/template/ISML');
var guard = require('storefront_controllers/cartridge/scripts/guard');
/*
  Use quickcard section "Invoking a Script". Use that as a help to complete the
following code to use the script named ProductFinder from the scripts folder.
   var ProductFinder= …require

*/
function start() {
    var parameterMap = request.httpParameterMap;
    var parameterId =parameterMap.pid.stringValue
    //var product = ProductMgr.getProduct(parameterId);

    var product=/* use quickcard section "Invoking a Script" again to invoke the
method on ProductFinder */

    if (product==null) {
            ISML.renderTemplate(
                    'productnotfound.isml', {message:'product with id
'+parameterId+' not found'}
                    );
        }
        else{
            ISML.renderTemplate(
                    'productfound.isml', {myProduct:product}
                    );

        }
}
exports.Start = guard.ensure(['get'], start);
```

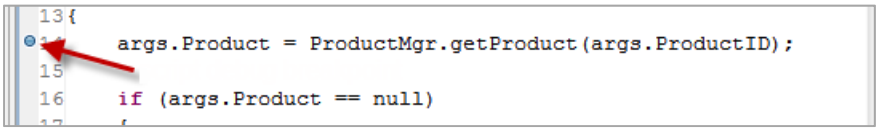
4. If not done already, modify your `productnotfound.isml` template so that it displays the contents of the `Log as follows:`

```
${pdict.Log}
```

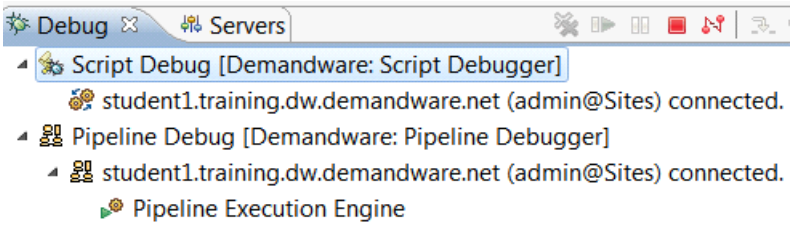
5. Test your controller. Verify that the product name appears as before and check the error path as well.

Lesson 7.3: Script and JavaScript Controller Debugging

UX Studio enables you to debug scripts and pipelines. To use the script debugger you must first create a script debug configuration. The process for creating a script debug configuration is identical to the pipeline debug configuration setup. To use the debug configuration, you need to add breakpoints in your script files.



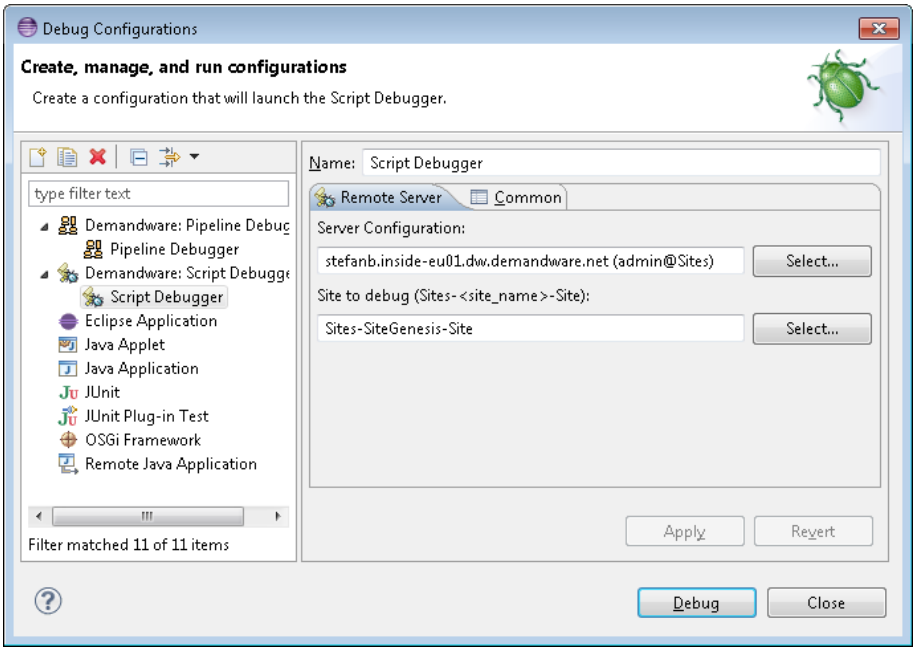
When debugging, it is possible to run the script debugger along with the pipeline debugger.



demandware
A Salesforce Company

Exercise: Create a Script Debug Configuration

1. In UX Studio, find the menu to create debug configurations.
2. Double-click on **Demandware: Script Debugger** to create a new configuration.
3. Complete the dialog as follows. Click **Select** to select your server and site.



4. Click **Debug** and change to the **Debug** Perspective.
5. Open the `ShowProduct` pipeline you previously created.
6. Put a breakpoint in the first executable line inside the pipelet's `execute()` function: double-click the gray border to the left of the highlighted line. The breakpoint display has a blue dot.

```
12 function execute( args : PipelineDictionary ) : Number
13 {
14     args.Product = ProductMgr.getProduct(args.ProductID);
```

7. Refresh the pipeline invocation on the browser to hit the breakpoint (**F5**).
8. The debugger stops at the breakpoint:

```
14     args.Product = ProductMgr.getProduct(args.ProductID);
```

9. Debug the script:
   a. Check the **Variables** window to see what `args` are coming into the `execute()` function.

b. Use **F5** to execute the line.

c. Study the `args.Product output` variable: it should not be `null`.

d. Execute through the end of the pipeline (**F8)**. The product name should display in the browser.

e. Fix any errors that you may have found, or just continue.

10. Debug the script again, but this time use an invalid product ID in the URL.

a. Change the `product` URL parameter on the browser to a non-existing product.

b. After the breakpoint, verify the `args.Product` variable: it should be `null` in this case.

c. Execute through the end of the pipeline.

demandware
A Salesforce Company

Lesson 7.4: Resource API and Resource Bundles

In storefront code, avoid hard-coding text strings that become visible to the user. Titles, labels, messages, button and field names should all be externalized by using resource bundles (a.k.a. properties files.). If you do not want to duplicate ISML templates in order to create locale-specific templates, you can use resource bundles to keep your template generic and reusable.

A resource bundle is a file with a `.properties` extension that contains the hardcoded strings to be used in ISML templates. In SiteGenesis bundles are loosely named by the functional area where the strings are used, but you can use any file name and organization you want.

Note: Property files can be suffixed by `Bundlename_<<locale_id>>.properties` where `<<locale_id>>` stands for a specific locale term **other than the default locale**. For example, "`de`" or "`en`" (or locale plus country like "`de_DE`" or "`en_GB`").

account.properties

```
account.header=My Account
account.sendfriend=Send to a Friend
account.sendlisttofriend=Send my List to a Frien

#############################################
# ISML Directory: account/
#############################################
accountoverview.welcome=Hello
accountoverview.welcome2=, Welcome Back
accountoverview.not=Are you not

accountredirect.title=Redirect
accountredirect.permanent=This account has been
accountredirect.accountlogin=login page.
account.forbidden=This page has timed out.  You

#############################################
# ISML Directory: account/addressbook/
#############################################
editaddress.editaddress=Edit Address
editaddress.addaddress=Add Address
editaddress.defaultaddress=Default Address
```

The resource bundles contain **key=value** pairs where the key might be compound (`key.subkey`) and the value is a hard-coded string that uses Java MessageFormat syntax to implement parameter replacement. Bundles are stored in each cartridge within the `/templates/resources` directory.

Strings from the bundles are accessible to all ISML templates via the `dw.web.Resource.msg(key : String , bundleName : String , defaultMessage : String )` method:

account.properties | accountlogin.isml

```
<isdecorate template="account/pt_account">
<isinclude template="util/modules"/>

<div class="accountlogin">
    <h1>${Resource.msg('account.header','account',null)}</h1>
```

Name of properties file

localized string value to use

Notice that the second parameter points to the `account.properties` file, which may be overridden by another cartridge in the cartridge path. The **null** in the third parameter means that the key itself will be used whenever that key is not found in any resource bundle. Instead of the `null` you can also show a string to display on the storefront in case the key could not be found.

Another useful method is the `dw.web.Resource.msgf(key : String , bundleName : String , defaultMessage : String , args : Object ...)`. Using this method, you can specify a key with placeholders which can be dynamically replaced by the parameters specified in the `args` argument of the method. For example, this usage of the method:

```
${Resource.msgf('singleshipping.wishlist', 'checkout', null,
owners.get(addressKey).profile.firstName )}
```

will be paired with the following Java MessageFormat definition in the resource bundle to allow the first name of the wishlist's owner to show up as **Stefan's Wishlist**:

```
singleshipping.wishlist={0}\'\'s Wishlist
```

demandware
A Salesforce Company

Knowledge Check

Demandware Script Review Questions	True	False
You have to mention the cartridge name when importing a script from another cartridge into a script file		
You can modify the functions of a Demandware pipelet.		
A script pipelet has full access to the Demandware Script API.		

demandware
A Salesforce Company

Exercise: Use a Resource Bundle in the Demandware Script in JavaScript Controller

Goal: Modify the `GetProduct` script and ISML to use an externalized string instead of a hardcoded string.

1. Open `controllers/JShowProduct.ds`. Inside this controller, just after checking if the product is null, add the following line of code to pick up a string `productnotfoundMsg` from th resource bundle named myBundle.properties:

   ```
   var errorMsg=dw.web.Resource.msgf('productnotfoundMsg', 'myBundle',
   null, parameterId);
   ```

2. Use quickcard as a guide (section "Giving control to ISML") and edit the next line to use the ISML to render the template `productnotfound.isml` and pass the json code `{message:errorMsg}`
3. Create a file `/templates/resources/myBundle.properties` with the following content (create the folder structure if not already there).

   ```
   productnotfoundMsg=The product with the id {0} is not found
   ```

4. Create a file `/templates/resources/myBundle_fr.properties`.
5. Change the encoding of this file to UTF8 to support French characters as follows.

   Right click on the file `myBundle_fr.properties` and change the default encoding to UTF-8

6. Type in the following in myBundle_fr.properties

   ```
   productnotfoundMsg = Le produit avec l\'\'ID {0} ne est pas trouvé
   ```

7. In the Business Manager, navigate to **Site-SiteGenesis>Site Preferences>Locales**>check the check box next to 'fr' and click on Apply button.
8. Now navigate run the `ShowProduct` pipeline or `JShowProduct` controller as you have been running before. Your url will look something like:

   `https://studentXX.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/JShowProduct-Start?pid=452345`

   Note: This is an example url and your url may vary. Also XX has to be replaced with your student id.

   You should be able to see the message of product not being found in English language

9. Now in the url, replace 'default' with 'fr' .

   You should be able to see the results in French language.

10. Now we are going to internationalize the product name. In the Business Manager, navigate to **Products & Catalogs>Products**
11. Search for the product with id 'P0048'
12. Click on the link to the product and lock it for editing.

13. Notice that the name of the product in 'default' language is 'Laptop Briefcase with wheels (37L)'. Change the value of the drop down box next to 'Select Language' as 'French'. The name entry will be now blank. Paste 'Laptop Briefcase avec des roues (37L)' without quotes in the name and click on the apply button.

14. In your isml which is displaying your product (`productfound.isml`), type in the following text to print the product name (remove the earlier text)

```
${Resource.msgf('productfoundMessage', 'myBundle',null,
pdict.myProduct.name)}
```

15. In templates/resources/myBundle_fr.properties add the following line:

```
'productfoundMessage'=Le nom du produit est {0}
```

16. In templates/resources/myBundle.properties add the following line:

```
'productfoundMessage'=The product is found and the name is {0}
```

17. Now navigate run the `ShowProduct` pipeline or `JShowProduct` controller with the parameter pid=P0048. Your url will look something like:

    https://studentXX.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/JShowProduct-Start?pid=P0048

    Note: This is an example url and your url may vary. Also XX has to be replaced with your student id.

    You should be able to see the product page with product name in English.

18. Now in the url, replace 'default' with 'fr' .

    You should be able to see the results in French language.

demandware
A Salesforce Company

Module 8: Forms Framework

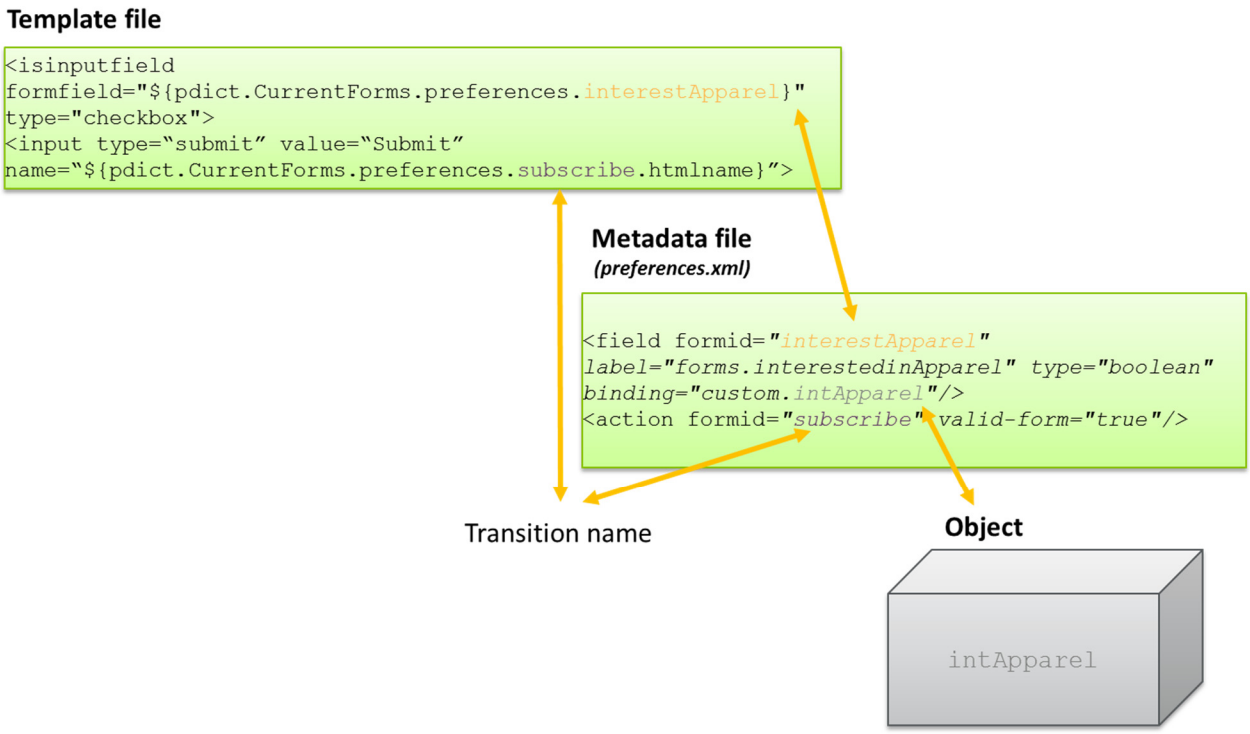
Learning Objectives

After completing this module, you will be able to:

- Describe the concepts and usage of the Demandware Forms framework.
- Create a new form and implement it in a pipeline.

Introduction

The Demandware platform provides a set of tools that help simplify form display and processing. Use the Demandware Forms framework to control how consumer-entered values are validated by the application, rendered in a browser, and possibly stored on a server.



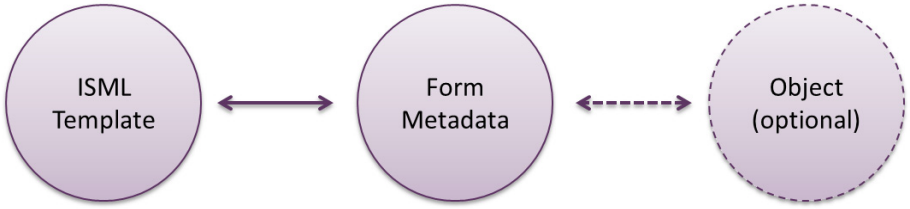
To use the Demandware Forms framework, you need the following files:

- An xml form to define and store the metadata
- A pipeline that will validate and process the form
- A properties file that contains externalized form labels and possible error messages
- An ISML template that will display the form to the user

There are three objects that interact when working with Demandware forms:

- XML metadata file: located in the `cartridge/forms/default` directory. It describes the fields, labels, validation rules and actions that apply when the field is used in an ISML template.

- ISML template: it uses the form metadata fields and actions to show an HTML form to the user.
- Object (optional): this object represents a single system or custom object in the `pdict`, and it can be used to pre-fill the metadata file as well as to store submitted form data to the database.



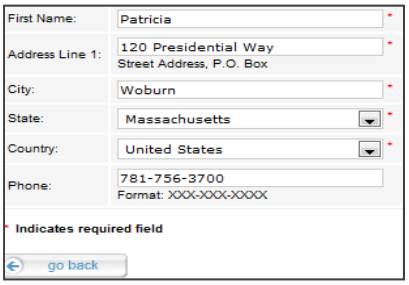
Example

Given this form metadata XML file:

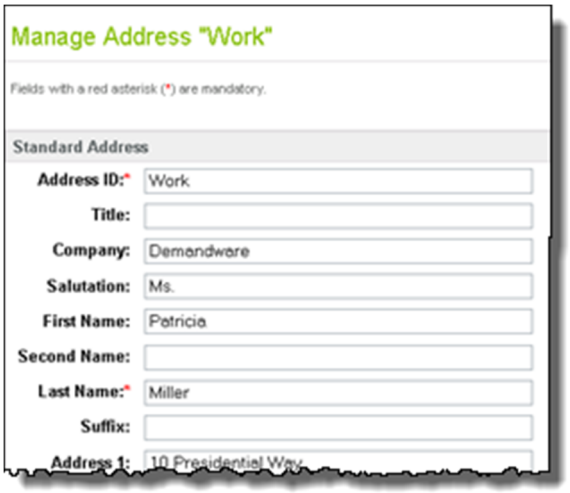
customeraddress.xml

```
<?xml version="1.0"?>
<form>
    <field formid="firstname" label="forms.custor
    <field formid="address1" label="forms.custome
    <field formid="city" label="forms.city.cities
    <field formid="state" label="forms.state.stat
    <field formid="zip" label="forms.customeraddi
    <field formid="country" label="forms.country
    <field formid="phone" label="forms.customerac
    <action formid="cancel" valid-form="false"/>
    <action formid="confirm" valid-form="true"/>
</form>
```

You can create this ISML template whose fields depend on the data from the form metadata.



Optionally, a `pdict` object containing data from the database can be bound to the form metadata file, allowing it to be pre-filled with data. This data would appear in the ISML template since it references the form fields.



demandware
A Salesforce Company

Lesson 8.1: XML Metadata File

As a developer, you will need to identify which fields a user will need to enter, and what actions can be taken when implementing a form. This information will probably come from a wireframe or a functional specification. Once the form fields are determined, create them in an xml form that will set the form field parameters and hold the data for the form.

The form metadata file uses the following xml elements:

Element	Description
`form`	Required: top level tag that contains all other elements inside `<form>…</form>`
`field`	Required: Defines data field with many attributes (see table below)
`options`	Use as a child element inside a field to pre-fill multiple options like months, days, etc.
`option`	Use as a child element inside an options element to specify a single option
`action`	Required: Defines a possible action the user might take on the form
`include`	Allows inclusion of one form metadata definition into another
`list`	Allows inclusion of several items (i.e. collection of addresses) as a single field
`group`	Allows grouping of elements to be invalidated together

The `field` element may use the following attributes:

Attributes	Description
`formid`	Required: unique ID to identify the field for ISML templates and pipelines.
`type`	Required: data type for field (see table below).
`label`	Usually a key to an externalized string in the `forms.properties` resource bundle.
`description`	Description for field, might be used in tooltips.
`min-length, max-length`	Restricts the field length for data entry.
`min, max`	Valid range for integer, number and dates.
`range-error`	Message shown if value provided does not fall within the specified range.
`regexp`	Regular expression for string fields: email, phone, zip code, etc.
`parse-error`	Message shown when the data entered does not match the regex. Usually a key to an externalized string.
`mandatory`	Field is required via server-side validation when true.

`missing-error`	Message shown if the primary key validation error is generated in a pipeline.
`value-error`	Shown if an element is invalidated in a pipeline.
`binding`	Used to match `field` to a persistent object attribute.
`masked`	Specify # of characters to mask.
`format`	Format for display of dates, numbers, etc.
`whitespace`	Specify whitespace handling (none or remove).
`timezoned`	Optional flag for date objects (true or false).
`default-value`	Pre-defines a value for a field.
`checked-value`	Value when field is checked in a form.
`unchecked-value`	Value when field is unchecked in form.

Field types can be as follows:

Field type	Description
`string`	Use for text data.
`integer`	Use for numeric data like days, months.
`number`	Use for quantity fields.
`boolean`	Use with multiple-choice fields.
`date`	Use this when `timezoned` or `format` are needed for dates.

Here is an example of a simple form metadata file:

```
<?xml version="1.0"?>
<form>
<field formid="fname" label="forms.contactus.firstname.label" type="string"
mandatory="true" binding="custom.firstName" max-length="50"/>

<field formid="lname" label="forms.contactus.lastname.label" type="string"
mandatory="true" binding="custom.lastName" max-length="50"/>

<field formid="email" label="forms.contactus.email.label" type="string"
mandatory="true" regexp="^[\w-\.]{1,}\@([\da-zA-Z-]{1,}\.){1,}[\da-zA-
Z-]{2,6}$"
parse-error="forms.contactus.email.parse-error"
value-error="forms.contactus.email.value-error" binding="custom.email"
max-length="50"/>

<action formid="subscribe" valid-form="true"/>
</form>
```

In this example, the fields `fname`, `lname` and `email` store the information needed to send a newsletter to a non-registered user. The fields are:

- Mandatory
- Contain label keys that point to the **`cartridge/templates/resources/forms.properties`** file

The email field has an extra requirement. It uses a **regular expression (`regexp`)** to define what an acceptable email can be. Additionally, it specifies a `parse-error` key which matches an error message in the `forms.properties` file.

The action `subscribe` identifies the possible actions that a user may take on the form. The attribute `valid-form="true"` means that this form requires validation: 3 required fields plus a valid email format for the last one will be enforced on the server side.

Note: Although it is not a requirement, it is a best practice to use lower-case letters when naming your xml forms. Pipelines are also xml files and use camel-case naming in SiteGenesis.

Lesson 8.2: ISML Form Template

Define an ISML template with the same tags needed for a valid HTML form:

```
<form>…</form>
```

You can implement your own form action by specifying a pipeline URL, but that would circumvent the Forms framework. When using the framework you specify an **Interaction Continue Node (ICN)** for the form action to post to, as follows:

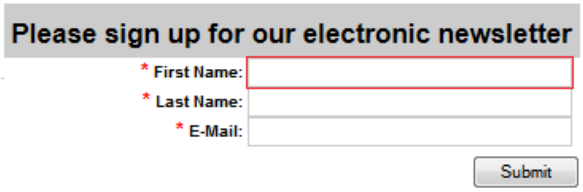
`<form action="${URLUtils.continueURL()}" method="post">`The method `dw.web.URLUtils.continueURL()` ensures that the form gets submitted back to the Interaction Continue Node (ICN) that displayed the form template. We will cover the ICN when we build the pipeline for the form.

When creating input fields, you must use the object `pdict.CurrentForms.<form metadata file>.<formid>` to reference the specific `formid` in the form metadata. SiteGenesis has an `<isinputfield>` custom tag which facilitates the creation of form fields. For example, to show the `fname` field from the `newsletter.xml` file as a text field in an ISML template, use:

```
<isinputfield formfield="${pdict.CurrentForms.newsletter.fname}"
type="input">
```

The custom tag will use `the fname formid` from the metadata file and build an HTML label using the `forms.properties` file to pull the text for the `forms.contactus.firstname.label` key. It also

creates an HTML input field to the right of the label with the necessary client-side JavaScript to enforce required fields, as shown.



You can modify the behavior of the `<isinputfield>` tag since it is a custom tag implemented in the SiteGenesis cartridge.

The final requirement in the ISML template is to implement the button that matches the action in the form metadata. For this we create a standard HTML button with a `name` attribute that points to a specific action in the form metadata:

```
<input type="submit"
 value="${Resource.msg('global.submit','locale',null)}"
name="${pdict.CurrentForms.newsletter.subscribe.htmlName}"/>
```

Here the `pdict.CurrentForms.newsletter.subscribe.htmlName` refers to the `htmlName` property of the action `subscribe` in the form metadata. In the debugger you can view the value of this property at runtime: `dwfrm_newsletter_subscribe`. This value identifies a specific action for a specific form, which is necessary when the pipeline ICN determines which form action to process.

Lesson 8.3: Form Pipeline Elements

A pipeline that uses the Demandware Forms framework has a very distinctive pattern because it uses the following elements:

- `ClearFormElement` pipelet to clear an existing form object in the `pdict` using a specified form metadata file.
- `InvalidateFormElement` invalidates the specified `FormElement` (To be used later in this book).
- Interaction Continue Node to show the ISML form, and to perform server-side validation.
- Transitions that match actions from the form metadata.
- A “next” transition that goes back to the ICN to handle validation errors.

To create a form using the form framework, follow these steps:

1. Create an xml metadata file that will hold your form data.
2. Create an ISML template that will display a form to a visitor.
3. Create a pipeline that includes at minimum:

To use forms in JavaScript controllers, you get hold of form object using the syntax:

```
var myformObject  =session.forms.<metadata>
```

To clear the form, the following can be used.

```
myformObject.clearFormElement();
```

To generate the form named newslettersignup and handle the submission using the `controller` named `JNewsletter`, the following code can be used.

```
ISML.renderTemplate('newsletter/newslettersignup', {
                  ContinueURL : dw.web.URLUtils.https('JNewsletter-HandleForm'),
                  CurrentForms :session.forms
                });
}
```

The following line can find out the submit button that was pressed

```
var submitAction = request.triggeredFormAction.formId;
```

demandware
A Salesforce Company

Exercise: Use the Forms Framework Using JavaScript Controller

Note: Steps 1 through 4 are only to be done if not already completed in the previous exercise

1. Define form metadata to store newsletter subscription data.
   a. Study the fields and action in the `newsletter.xml` file from the `solutions` cartridge.
   b. Save `newsletter.xml` into your cartridge at exactly **the same location** as in solutions.
2. Create `templates/resources/forms.properties` and externalize the keys in `newsletter.xml`. For example.

   ```
   forms.contactus.firstname.label=Enter your first name please.
   ```

3. Define a template to capture form data.
   a. Study the use of the `<isinputfield>` custom tag in the `newslettersignup.isml` template in the `solutions` cartridge.
   b. Study the use of the `URLUtils.httpsContinue()` method. What will this method accomplish in the context of the form action?
   c. Save `newslettersignup.isml` from the solutions cartridge into your cartridge under similar directory structure (`templates/default/newsletter`).
4. Create a template to display the form values submitted.
   a. Save `newslettersuccess.isml` from the solutions cartridge into your cartridge: this displays a "Thank you <fname> <lname> for signing up" under similar directory structure.
   b. Create resource bundle `templates/resources/locale.properties` and externalize the keys used in `newslettersignup.isml` and `newslettersuccess.isml`.

      For example:

      ```
      global.newslettersignup=Please sign up for our newsletter
      global.newsletterthanks=Thank you {0} {1} for signing up!
      ```

5. Create a JavaScript controller named JNewsletter.js.
6. Copy and paste the following template and follow the instructions in the comments to fill the code.

```
'use strict';

/** @module controllers/JNewsletter */

var ISML = require('dw/template/ISML');
var guard = require('storefront_controllers/cartridge/scripts/guard');
var newsletterForm=
  /* use quickcard section "Handling Forms" to get Form object of type newsletter */
```

```
function start() {

   /* use quickcard section “Handling Forms” to clear the form object above(newsletterform
*/

/* Use quickcard to render newsletter/newslettersignup isml.  The pipeline to handle the
submission is JNewsletter-HandleForm  */
}


function handleForm() {
    var TriggeredAction = request.triggeredFormAction;
      response.getWriter().println('Hello World from pipeline
controllers!'+TriggeredAction);
      if ( (TriggeredAction != null) && (TriggeredAction.formId == 'subscribe') ){
       //response.getWriter().println('Hello World from pipeline
controllers!'+newsletterForm.fname.value);
            ISML.renderTemplate('newsletter/newslettersuccess', {
                CurrentForms    : session.forms
            });
            return;

             }
      else{
          ISML.renderTemplate('newsletter/newslettersignup', {
              ContinueURL : dw.web.URLUtils.https('JNewsletter-HandleForm'),
              CurrentForms :session.forms
          });

      }

}

exports.Start = guard.ensure(['get'], start);

exports.HandleForm = guard.ensure([], handleForm);
```

7. Execute the code.

Knowledge Check

Forms Framework Questions	True	False
The `<isinputfield>` is a custom tag used to populate form field attributes.		
An Interaction node is used to display and submit a form.		
A transition node named ‘next’ will continue pipeline logic if a form has been successfully validated for a ‘subscribe’ action to occur.		

demandware
A Salesforce Company

Module 9: Custom Objects

Learning Objectives

After completing this module, you will be able to:

- Define custom objects and create instances programmatically.
- Use a transactional pipelet to save the custom object in the database.
- Implement custom logging to allow debugging and error messages to be written to logs.

Introduction

Previously, you created a simple form using an Interaction Continue Node. You validated the data being submitted, but did not store the data permanently. **Custom Objects (CO)** enable the data to be persistent.

Custom Objects extend the Demandware data model. They are basically a new table in the database where you specify the primary key and storage attributes (columns) that suit your business needs.

Note: Always first consider if you can use a Demandware System object (Product, Catalog, etc.) instead of creating a custom object. Although you can create custom objects, they are best used to store static data (like configuration parameters), not for uncontrolled amounts of data (like analytics). Custom objects searches can be slow if the data is large. You should consider data growth and cleanup in your Custom Objects. Demandware Platform Governance has quotas around custom object API usage and data size which will be enforced in the future.

Custom Object Creation

Custom objects are created at the organization level and are therefore available for use in all storefronts within the organization. You use two Business Manager modules to define and manage your custom objects:

- Custom Object Definitions: facilitates naming, primary key and column specification. It is located in **Administration > Site Development.**
- Custom Object Editor: facilitates instance creation and editing. It is located in **Site - <site> > Custom Objects > Custom Object Editor**.

When defining the Custom Object, specify the storage scope of the instances: site or organization.

- Organization Custom Objects can be used by any site.
- Site Custom Objects are created by one site and cannot be read by another.

The Custom Object type itself is always available to the entire organization. Also, you can specify if you want Custom Object instances to be replicable. This means you can copy them from Staging to Production during the replication process.

An example of Custom Object usage is a newsletter. Customers can sign up for it, but the platform does not have a system table to support. These subscriptions are intended for export since the platform should not be used for mass mailing campaigns. It is tempting to add the subscription data to the Profile system object, but this would imply that only registered users would be able to sign up. To enable anyone to get a newsletter, we need to define a Custom Object. This Custom Object should not be replicable, since subscriptions created in Staging should not be copied to Production.

You also need to consider how to clean up Custom Objects once they have been exported or after a certain expiration period. This means the creation of a cleanup batch job that should run on a schedule.

Custom Objects can also store configuration parameters to integrate with external systems, avoiding the need to create multiple Site Preferences. These Custom Objects need to be replicable if the settings made in Staging are suitable for Production.

You can either create custom objects using Business Manager or programmatically. Before you can create a custom object instance you must first define the custom object data type in Business Manager.

Creating a New Custom Object Type Using Business Manager

To create a new custom object type in Business Manager, follow these steps:

1. Log into Business Manager.
2. Click **Administration > Site Development > Custom Object Definitions.**
3. Click the **New** button to create a new Custom Object type.
4. Fill in the required fields for the Custom Object type:
    a. **ID**: the unique ID of the object type. It cannot contain spaces.
    b. **Key Attribute:** This is the unique key for the custom object type.
    c. **Data Replication**: Specify whether the custom object type data will be replicable to other instances.
    d. **Storage Scope**: Specify whether the custom object type will be available for a site or for the entire organization.
5. Click **Apply**. The **Attribute Definitions** and **Attribute Grouping** tabs will become available.
6. Click the **Attribute Definitions** tab. Notice the default values created with your Custom Object type. These values cannot be changed once they are created.
7. To create the attributes (values you wish to capture in the table), click **New**.
8. In the **ID** field, specify a unique name. In the **Value Type** drop-down, select the type of data being entered for the attribute.
9. Click **Apply**.
10. Click the **Back** button to add another attribute.

11. When you are finished adding attribute definitions, create an Attribute Group. Click the **Attribute Grouping** tab.
12. In the **ID** field, enter a name for your grouping. In the **Name** field, enter a name. Click **Add**.
13. Add field attributes to the group. Click the **Edit** link.
14. To the right of the ID field, click the ellipses to select field attributes.
15. Select the attributes you wish to add from the list by clicking in the checkbox next to each one. Then click **Select**.
16. You are now ready to view, add, and edit new instances of the custom object type you just created in the **Custom Object Editor** section.

Creating a New Custom Object Instance Manually Using Business Manager

1. In Business Manager, select the site for which you want to manage custom objects.
2. Select **Custom Objects > Custom Object Editor.** The Manage Custom Objects page display.
3. Select the custom object type you wish to manage from the drop-down list.
4. To create a new custom object, click **New**.
5. Enter data in each of the required fields. Click **Apply**.
6. You have now created a custom object. Click the **Back** button to exit the custom object editor.

Exercise: Create a Custom Object Definition

Define a custom object to store the customer data gathered from your Newsletter form.

1. In Business Manager, select **Administration > Site Development > Custom Object Definitions**.
2. Create a new Custom Object type with the following attributes:
   a. ID – `NewsletterSubscription`
   b. Key Attribute – `email`, type `String`
   c. Name of the Table - your choice
   d. Data Replication – `not replicable`
   e. Storage Scope – Site
3. Add the following attributes:
   a. `firstName`, type `String`
   b. `lastName`, type `String`
4. Create an attribute group for the `NewsletterSubscription` Custom Object:
   a. Name: `Presentation`.
   b. Attributes: `firstName`, `lastName` and `email`
5. Select **Site - SiteGenesis > Custom Objects > Custom Object Editor.** Find the new `NewsletterSubscription` type and manually enter a new subscription.

Lesson 9.1: Using Demandware Script to Create Custom Object Instances

The Demandware Script API provides the following classes in the **dw.object** package, among others:

- CustomAttributes: attributes defined by a user in Business Manager to extend a system object or Custom Object. Accessible via the syntax: `co_instance.custom.attribute.`
- CustomObject: represents an instance of a Custom Object.
- CustomObjectMgr: enables Custom Object instance creation.
- PersistentObject: enables persistent storage.
- ExtensibleObject: enables custom attributes to be added.

This is the inheritance tree for the CustomObject type:

```
Object > dw.object.PersistentObject > dw.object.ExtensibleObject >
dw.object.CustomObject (or dw.object.SystemObject)
```

This inheritance tree means that Custom Objects are persisted in the database and can have custom attributes added by an administrator or developer in Business Manager. As you inspect the Demandware documentation you will see that commonly used classes like `dw.catalog.Product`, `dw.system.SitePreferences` and many others share this inheritance tree: objects of these class types are saved in the database and can be extended to store extra attributes.

The following usage of the `CustomObjectMgr` class enables creation of an instance of a Custom Objects by providing the Custom Object type and the primary key:

```
CustomObjectMgr.createCustomObject("NewsletterSubscription", UUIDUtils.createUUID());
```

This will create an instance with a system-generated, unique PK. You could also use:

```
CustomObjectMgr.createCustomObject("NewsletterSubscription", args.email);
```

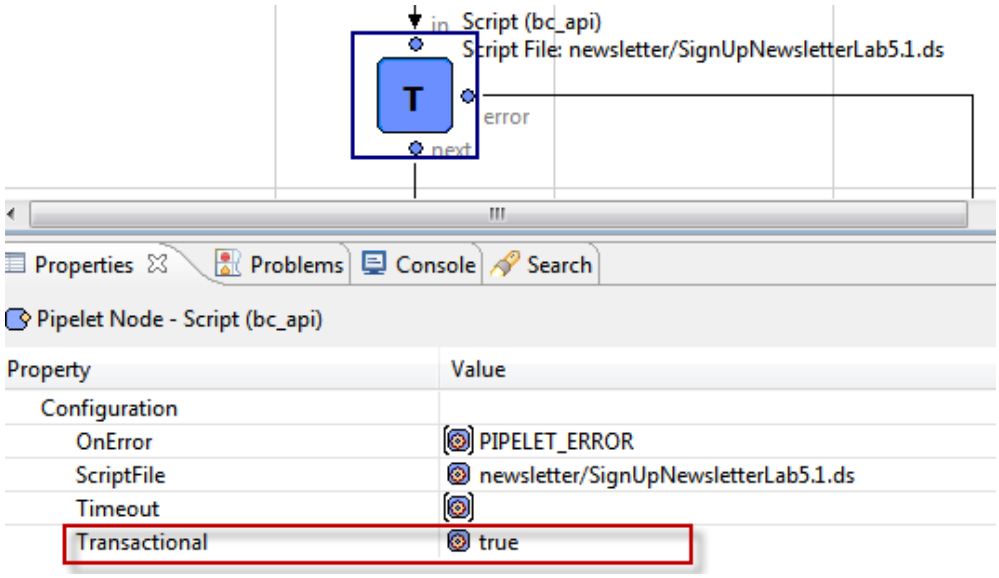
This assumes that the `args.email` value should be a unique string every time a Custom Object is created. Otherwise, a duplicate PK error will occur.

Database Transaction Handling

There are two approaches to database transaction handling in Demandware:

- Implicit – a transactional pipelet automatically begins a transaction. The transaction is automatically committed to the database when the pipelet returns `PIPELET_NEXT`, otherwise the transaction is rolled back.
- Explicit – the transaction is controlled via properties of the transition nodes in the pipeline.

For implicit transactions to work, a pipelet that performs changes to the database must set its Transactional property equal to **true**. This becomes visible as a black "T" on the pipelet node in the pipeline:



If such a transactional pipelet is executed, a database transaction will be started automatically and will be committed implicitly at the end of this pipelet's execution if the execution is successful.

Create a Custom Object via Script API

To create a custom object programmatically, follow these steps:

1. Create a custom object type in Business Manager before creating a custom object programmatically. If you have not already done so, create your custom object type as defined in the previous process step.
2. Create a script that uses the `dw.object.CustomObjectMgr` class to create a custom object:

```
importPackage( dw.system );
importPackage( dw.object );

function execute( args : PipelineDictionary ) : Number
{
var co = CustomObjectMgr.createCustomObject("NewsletterSubscription",
args.email);

co.custom.firstName = args.firstName;
co.custom.lastName = args.lastName;

args.subscription = co;

return PIPELET_NEXT;
}
```

Notice the use of the custom qualifier for all the custom attributes that you defined in the CO definition: without this qualifier the code will fail since the class `dw.object.CustomObject` does not have any standard attribute named `firstName`

demandware
A Salesforce Company

Example: Implicit and Explicit Transactions in JavaScript Controllers or Scripts

Transactions can be implicit or explicit and are handled in the code. The following are the examples of implicit and explicit transactions.

Implicit Transactions

```
Transaction.wrap(function() {
couponStatus = cart.addCoupon(couponCode);
});
```

Explicit Transactions

```
var Transaction = require('dw/system/Transaction');
Transaction.begin();
if {
code for the transaction
…
}
else {
Transaction.rollback();
code after the rollback
…
}
Transaction.commit();
```

Exercise: Create a Custom Object using JavaScript Controller

1. Create a controller `JNewsletterV2.js`.
2. Copy and paste the following template in it.

```
'use strict';

/** @module controllers/JNewsletterV2 */

var ISML = require('dw/template/ISML');
var Custombject=require('dw/object/CustomObject');
var guard = require('storefront_controllers/cartridge/scripts/guard');
var newsletterForm = session.forms.newsletter;
var Transaction = /* use require to use Transaction from dw.system package */

function start()
{
     newsletterForm.clearFormElement();
               ISML.renderTemplate('newsletter/newslettersignup', {
                        ContinueURL : dw.web.URLUtils.https('JNewsletterV2-HandleForm'),
                        CurrentForms :session.forms
                });
}

function handleForm() {

    var TriggeredAction = request.triggeredFormAction;
    var newsletterForm=null;
      if ( (TriggeredAction != null)
          && (TriggeredAction.formId == 'subscribe')   )
     {

/* use quickcard section “Explicit Transactions” to seek help and begin Transaction */

            var myModel = /* use require to use MyModel from script folder  */

            var co=myModel.createMyObject(newsletterForm);

 /* use quickcard section “Explicit Transactions” to seek help and commit Transaction */

                    ISML.renderTemplate('newsletter/newslettersuccessV2', {
                        CurrentForms    : session.forms,
                        Subscription    : co
                  });

         return ;
          }
          catch (e){
```

```
                    // response.getWriter().println("In th exception block
   "+e.causeMessage);
                     newsletterForm.email.invalidateFormElement();
                Transaction.rollback();
                      ISML.renderTemplate('newsletter/newslettersignup', {
                          ContinueURL : dw.web.URLUtils.https('JNewsletterV2-HandleForm'),
                          CurrentForms :session.forms
                    });
                   return;
            }

        }
        else{
                    //your email address is not even a valid email address go back and fill
forms again
                    //response.getWriter().println("session forms are here:
"+session.forms);
                 ISML.renderTemplate('newsletter/newslettersignup', {
                ContinueURL : dw.web.URLUtils.https('JNewsletterV2-HandleForm'),
                CurrentForms :session.forms
                    });
                 return;

        }

}

exports.Start = guard.ensure(['get'],start);
exports.HandleForm=guard.ensure([],handleForm);
```

1. Copy `MyModel.js` from the `JSolutions` cartridge to your `training` cartridge (in same location) and study how it is creating the object.
2. Copy `newsletter/newslettererrorV2.isml` and `newslettererrorV2` from `JSolutions` cartridge into your cartridge in the similar location.
3. Execute the controller and see if objects are being created.

Lesson 9.2: Custom Logging

The Demandware platform supports custom logging using log categories and severity levels as defined by the Apache log4j open source project.

Log4j supports multiple severities and categories of logging to allow the developer to capture debug messages at different levels of granularity. The severity levels are:

Debug < Info < Warn < Error < Fatal

If custom logging is enabled for a certain severity level, then it is enabled for higher severity levels as well (read from left to right). Fatal and Error are always enabled and cannot be turned off.

The developer can define as many levels of categories and subcategories as needed. Demandware does not impose a certain categorization; the developer determines the organization. For example:

- product
- product.import
- product.import.staging

If logging is enabled for a category (such as product), all its subcategories will also be enabled. For example, if Warn logging is enabled for product, then Warn, Error and Fatal errors are logged for product and all its sub-categories.

However, if Warn logging is enabled for product and Debug is enabled for product.import, then:

- Warn, Error and Fatal messages are logged for "product" and all its sub-categories.
- Debug and Info are logged for "product.import" and all its sub-categories.

To write to a custom log, you need to use the `dw.system.Logger.getLogger()` factory method. This method creates a Logger object for a specified category:

```
var logger = Logger.getLogger("logFilePrefix","category" );

 logger.debug("Input params received in pipelet
        firstName: {0}\n lastName: {1}\n email: {2}",
        args.firstName, args.lastName, args.email);

 try
 {
  … do something…
  return PIPELET_NEXT;
 }
 catch (e)
 {
   logger.warn("error description: {0}", e.causeMessage );
   return PIPELET_ERROR;
 }
```

Use the Logger object to write a message for a specific severity level: `Logger.error(String msg).`

The message uses the Java MessageFormat API, so you can specify placeholders. Typically these messages are not localized since they are read internally by site administrators, but they can be.

Enabling Custom Logging

In order to write to log files, you need to enable Custom Log Settings:

1. In Business Manager, select **Administration > Operations > Custom Log Settings**.
2. Create a log category. Enter it in the field under a given severity. Click **Add.**
3. **Enable** the check box next to the log category where you want to write. Click **Apply**.
4. Click **Log Debug to File** to enable debug messages to be written to a log up to 10 megabytes. Usually Debug and Info messages are written to memory only, and visible via the Request Log tool.
5. Run the pipeline you wish to debug.
6. In Business Manager, review the custom log file. Click **Administration > Site Development > Development Setup > Log Files**.
7. Open the log file that was just created. Search for the file by date. The custom log file will be named similar to: `customdebug-177.aaaq.demandware.net-appserverxxxx.log`. However, it you gave a prefix while creating the log the name of the file will be starting with `custom-<prefix name>`

demandware
A Salesforce Company

Exercise: (This may be a Demo based on Time) Custom Logging in JavaScript Controller

Modify the script from the Newsletter Subscription so that it writes debug messages to a log file, as well as error messages when a duplicate key is used.

1. Modify `MyModel.js` to Write to Debug Log.
   a. Open the `scripts/MyModel.js` script.
   b. Use the `require` to get `Logger` from `dw.system package`.
   c. Use the following code as a guide to create the try-catch block around the code written in the `MyModel.js`. Please note that the code is **not the exact code**. You have to correct it to reflect the name and email is coming from the form that was submitted.

```
var logger = Logger.getLogger("NewsLogs" ,"newsletter" );
logger.debug( "Input params firstName: {0} lastName: {1} email: {2}",
args.firstName, args.lastName, args.email);

//Instantiate the NewsletterSubscription custom object
try
{
… code from previous lab …
}
catch (e)
{
logger.error("A newsletter subscription for this email address
 already exists: {0}", e.causeMessage );
return null;
}
```

2. Enable Logging for Debug Messages.
   a. In Business Manager, select **Administration > Operations > Custom Log Settings**.
   b. In the Log Category field, set `root` to `DEBUG` Level. This will cover all categories.
   c. Click **Add** to enable debugging for All debug messages.
   d. Click **Log Debug to File**.
   e. Click **Save** to save these changes.
3. Test your pipeline with a duplicate email address and verify the **latest customwarn** log files.
4. Select **Administration > Site Development ⇒ Development Setup > Log Files**.
5. Verify the messages on the `customdebug` and `customerror` log files that appear with the most recent timestamp. The name of the file should start with `custom-NewsLogs`

   Verify the debug messages also appear on the request log.

Knowledge Check

Custom Object Questions	True	False
Custom objects are the only way to store custom data in Demandware.		
The “custom” keyword is required to access custom attributes of an object.		
A custom object needs a primary key.		
Custom object instances can only be created in Business Manager.		
An implicit transaction means that the pipelet commits if PIPELET_NEXT is returned.		

demandware
A Salesforce Company

Module 10: Data Binding and Explicit Transactions

Learning Objectives

After completing this module, you will be able to:

- Use data binding to pre-fill forms and update persistent data from the form.
- Use an explicit transaction to commit changes to the database.

Lesson 10.1: Data Binding with Forms and Objects

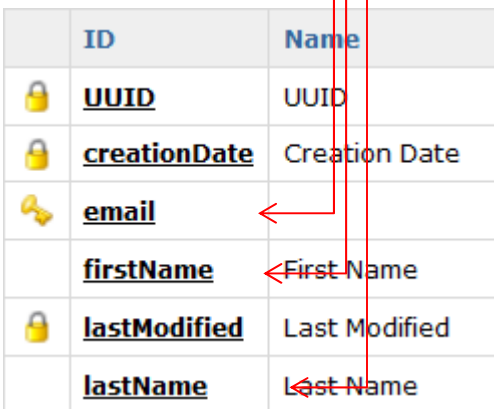
The Demandware forms framework supports binding of persistent objects to form fields by automatically updating a persistent object with form data without having to issue an insert statement or calling a Demandware API. The reverse mechanism is also supported: pre-populating a form object with data from a persistent object.

The object that is bound to the form must be a persistent object (system or custom), and must be available in the `pdict`. The form metadata must have field(s) with the `binding` attribute specified. The field `formid` attribute is not used to make the match; only the `binding` attribute identifies what fields match between the form and the object. The following form metadata uses `custom.firstName, custom.lastName, custom.email` as the bindings:

```
<?xml version="1.0"?>
<form xmlns="http://www.demandware.com/xml/form/2008-04-19">
    <field formid="fname" ... binding="custom.firstName" max-length="50"/>
    <field formid="lname" ... binding="custom.lastName" max-length="50"/>
    <field formid="email" ... binding="custom.email" max-length="50"/>

    <action formid="subscribe" valid-form="true"/>
</form>
```

This is because the `NewsletterSubscription` Custom Object we want to bind this form to have `firstName`, `lastName` and `email` fields which are all custom attributes. Notice that the fields do not have a lock icon (they were added by you as custom attributes of the Custom Object):



Exercise: Create a Custom Object Using JavaScript Controller

1. Save the controller `JNewsletterV2.js` as `JNewsletterV3.js` in your `training` cartridge.
2. In `JNewsletterV3.js` replace all occurences of `JNewsletterV2` as `JNewsletterV3` in the code.
3. Comment the following lines of code:

```
var myModel = require('~/cartridge/scripts/MyModel');
var co=myModel.createMyObject(newsletterForm);
```

4. Just below these lines, use the `require` syntax to get `CustomerMgr` form `dw.object` package.
5. Add a the next line as:

```
var co=CustomObjectMgr./*invoke a method to create object from
NewsletterSubscription custom object type with the
newsletterForm.email.value as the primary key.  */
```

(In the line above follow the instruction in the comment to make the line fully executable and working (use Script API as the guide if needed).

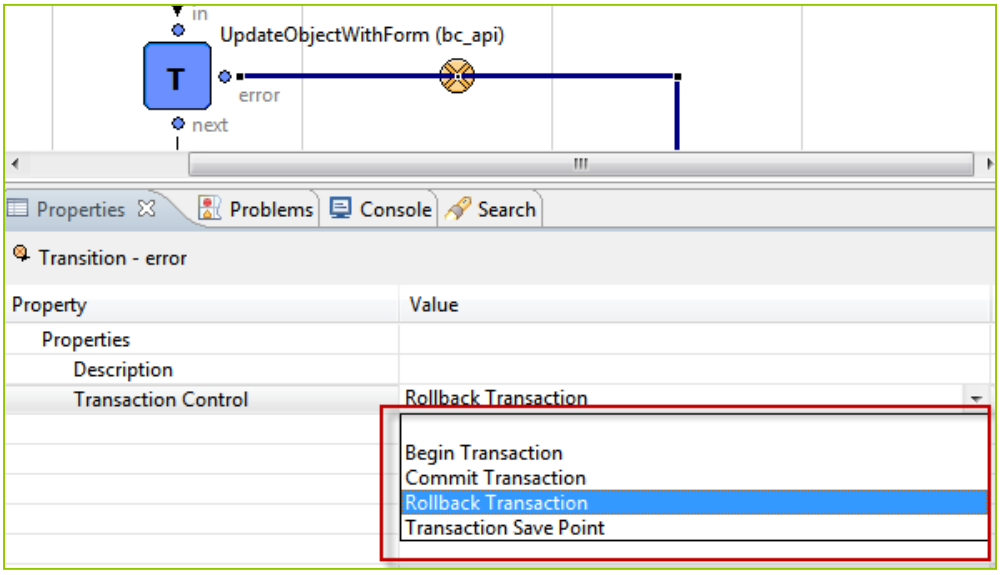
6. Use `copyTo` method (use quickcard section “Handling Forms”) to store `newsletterForm` to the object created above.
7. Execute the JavaScript controller.
8. Check in the Business Manager>Merchant Tools>Custom Objects>Custom Object Editor to see if the object has been created.

Lesson 10.2: Explicit Database Transaction Handling

Transaction handling can occur implicitly when executing a transactional pipelet; the commit or rollback is controlled by the `PIPELET_NEXT` or `PIPELET_ERROR` return values. However, in some circumstances the transaction spans several pipelets or steps. In this case, you decide where the transaction begins and ends. This mechanism is called Explicit Transaction Handling. This is implemented at the pipeline level by changing the Transaction Control property of a transition or connector using the following controls:

Begin transaction	
Commit	
Rollback	

To set transaction handling on the transition node, open the properties window for the transition, and select the type of Transaction Control you want.



Use this to override the built-in implicit transaction to group changes that need to be part of an atomic transaction.

Exercise: Store and Retrieve the Preferences using JavaScript Controller JEditPreferences.js

Create a new `JEditPreferences.js` controller such that you pre-fill the form with the logged-in customer preferences. Once the customer changes his/her preferences you save the data to the database.

Note: Attempt Step 1 through 3 only if you have not done those in the previous exercise.

1. Extend the `Profile` System Object.
   a. In the Business Manager, extend the `Profile` system object with the following custom attributes: (Administration>Site Development>System Object Definitions)
      - `interestApparel : Boolean`
      - `interestElectronics : Boolean`
      - `newsletter : Boolean`
   b. None of the attributes are mandatory.
   c. Add them to an Attribute Group `Preferences` to view the settings later in the Customer area.
2. Modify the Content Asset that shows the Account Overview.
   a. Login to your Storefront account (register as a new customer if you haven't already).
   b. On the account overview page, use the **Storefront Toolkit > Content Information** to locate the `account-landing` content asset which is located in the middle of the page (or go to Business Manager and locate it).
   c. In the `account-landing` asset, add a new list item that calls the `JEditPreferences` pipeline. Use the `$httpsUrl(JEditPreferences-Start)$` content link function to invoke your pipeline (this syntax was covered in the Content Slot module):

```
<li>
        <a title="View and modify items on your list or invite friends"
href="$httpsUrl(JEditPreferences-Start)$">
            <i class="fa fa-bookmark"></i>
            <h2>Preferences</h2>
            <p>View and modify your preferences</p>
        </a>
    </li>
```

3. Edit the Form Metadata to add bindings and externalization of strings.
   a. Open the `preferences.xml` form metadata file.
   b. Externalize the keys in `preferences.xml` in `templates/resources/forms.properties` file for example:

`forms.preferences.apparel=Are you interested in Apparel ?`

Likewise externalize the other two keys.

c. For each custom attribute you defined in the `Profile` system object, make sure there is a corresponding form field with a binding that matches the spelling you used as the object attribute. For example:

```
<field formid="interestApparel"
   label="forms.preferences.apparel" type="boolean"
   binding="custom.interestApparel"/>
```

4. Copy the `editpreferences.isml` from the `customerpreferences` cartridge to your own cartridge under similar directory structure (`templates/default/account/user`) . Make sure the formfields are matching the formids of the `preferences.xml` metadata file.
5. Create JEditPreferences.js controller in your cartridge.
6. Copy and paste the following template in your controller and complete the code in the comments using help from quickcard.

```
'use strict';

/** @module controllers/JEditPreferences */

var ISML = require('dw/template/ISML');
var guard = require('storefront_controllers/cartridge/scripts/guard');
var URLUtils = /* use require syntax to get URLUtils from dw.web package */
var Transaction = /* use require syntax to get Transcation from dw.system package */
var preferencesForm = /* use quickcard section “Handling Forms” to get Form object from the
meta data named preferences*/

function start() {

      /* use quickcard section “Handling Forms” clear the preferences from  */

      /* use quickcard “Handling Forms” to find a method to prefill the form from
customer.profile system object */

      ISML.renderTemplate('account/user/editpreferences.isml', {
            ContinueURL : dw.web.URLUtils.https('JEditPreferences-HandleForm'),
            CurrentForms : session.forms
      });
}


function handleForm() {

      var TriggeredAction = request.triggeredFormAction;

      Transaction.begin();
         /* store preferencesForm to the sytem object customer.profile */
      Transaction.commit();
      response.redirect(URLUtils.https('Account-Show'));
      return;
```

```
}

    exports.Start = guard.ensure([ 'get', 'https', 'loggedIn' ], start);
    exports.HandleForm = guard.ensure([ 'post' ], handleForm);
```

7. Execute the controller from the account page (You will have to modify the content asset to invoke `JEditPreferences` (not `EditPreferences`)).

demandware
A Salesforce Company

Module 11: Site Maintenance

Learning Objectives

After completing this module, you will be able to:

- Use the Pipeline Profiler and implement page caching.
- Replicate code and data in the Primary Instance Group (PIG).

Introduction

Demandware provides tools that you can use to improve site performance as well as replicate code and data.

Note: The *Demandware Customization, Integration, and Performance* course provides additional information on this topic.

Lesson 11.1: Site and Page Caching

Page download time is a critical factor in keeping visitors in your storefront. The longer it takes to download a page, the higher your risk of losing a sale. Therefore, it is best to cache your pages as much as possible to minimize page download times.

Furthermore, rendering pages containing many business objects or complex calculations such as category and search result pages or product detail pages can consume a lot of resources. Since this information generally does not change from one user to another, not caching these pages can excessively waste processing resources which will slow down the entire site for all users (including job processing) and not just for the requested pages.

Demandware controls caching on a per page basis, via the ISML template for the page. Set caching on a page using the `<iscache>` tag:

```
<iscache type="relative" hour="24">
```

Demandware follows these rules when using the tag:

- If `<iscache>` tag occurs multiple times in a template or its locally included templates, the shortest duration is used.
- Caching from a local include affects the including template.
- If there is no `<iscache>` defined, the template is not cached.

Use the `<iscache>` to set the following parameters:

Parameter	Description
`type = "relative | daily"`	Relative enables you to specify a certain period of time, in minutes and hours, after which the page will be deleted from the cache. Daily enables you to specify an exact time when the page will be deleted from the cache.
`hour = integer`	Indicates either the caching duration or the time of day. If the `type` attribute is set to `daily`, the `hour` value must be an integer, ranging from 0 to 23. If `type` is set to `relative`, all integer values greater than 0 are valid (the default value is 0, meaning either the page is never cleared from the cache or only the `minute` attribute is relevant).
`minute = integer`	Indicates either the caching duration or the time of day. If the `type` attribute is set to `daily`, the `minute` value must be an integer ranging from 0 to 59. If `type` is set to `relative`, all integer values greater than 0 are valid (the default value is 0, meaning either the page is never cleared from the cache or only the `hour` attribute is relevant).
`varyby= "price_promotion"`	Enables you to mark a page as personalized: this does not mean that the page is unique for a person but rather that different versions of the same page showing different prices, promotions, sorting rules or AB test segments will be cached by the Demandware platform. For example, this parameter is necessary for product pages since a customer belonging to a customer group might get special promotions that other customer groups don't get. While the ISML template is the same, the generated pages vary, and therefore caching every version of the page benefits performance. For performance reasons, a page should only be **marked with the** `varyby` **property if the page is really personalized**; otherwise, the performance can unnecessarily degrade.

Frequently changing pages benefit from a shorter caching period. Stored pages are only invalidated and a new one pulled from the application server if any of the following occur:

- The defined caching time is exceeded.
- A replication has been performed (with the exception of coupons and geolocation data).
- An explicit page cache invalidation is triggered by a merchant in Business Manager.

As a best practice, disable page caching on sandboxes, development and staging environments in order to see changes immediately. In Production caching is always on by default.

Portions of pages can be cached separately. You can assemble a page from snippets with different caching attributes using remote includes. Each part:

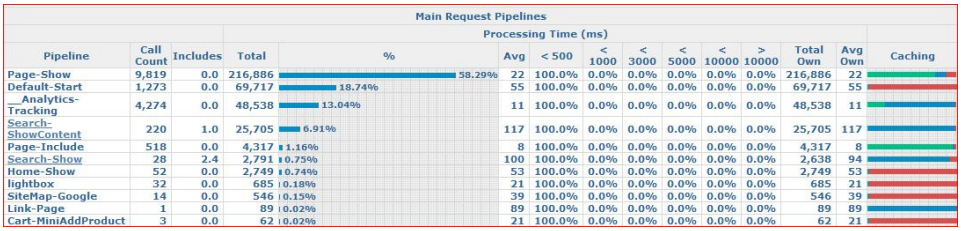
- Must be a result of a pipeline request to the application server.
- Is included using the `<isinclude url="">` or the `<iscomponent pipeline=….>` syntax.

- Can have different cache times or no caching at all.

In general, do not cache pages that show buyer or session information.

Studying Page Analytics to Determine Caching Problems

To access Demandware caching metrics, select **Site > Analytics > Technical Reports**. Shown is the **Pipeline Performance** report.



Main Request Pipelines

Pipeline	Call Count	Includes	Processing Time (ms) Total	%	Avg	< 500	< 1000	< 3000	< 5000	< 10000	> 10000	Total Own	Avg Own	Caching
Page-Show	9,819	0.0	216,886	58.29%	22	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	216,886	22	
Default-Start	1,273	0.0	69,717	18.74%	55	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	69,717	55	
__Analytics-Tracking	4,274	0.0	48,538	13.04%	11	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	48,538	11	
Search-ShowContent	220	1.0	25,705	6.91%	117	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	25,705	117	
Page-Include	518	0.0	4,317	1.16%	8	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4,317	8	
Search-Show	28	2.4	2,791	0.75%	100	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2,638	94	
Home-Show	52	0.0	2,749	0.74%	53	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2,749	53	
lightbox	32	0.0	685	0.18%	21	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	685	21	
SiteMap-Google	14	0.0	546	0.15%	39	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	546	39	
Link-Page	1	0.0	89	0.02%	89	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	89	89	
Cart-MiniAddProduct	3	0.0	62	0.02%	21	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	62	21	

These types of analytics are only collection on Production instances, not Sandboxes. In this example, it reveals that the `Home-Show` pipeline (which generates the homepage) is not cached: the Caching column shows red for all hits. If you see this trend in your analytic data, you may decide to alter the caching settings or the caching interval.

Across Demandware customers, the two critical metrics we look at from a pipeline performance perspective are the average response times of `Search-Show` and `Product-Show` pipelines. The reason for this is these pipelines are used across all customers and are the main components of most pages on Demandware installations.

- For `Search-Show` the average response is 400ms. Customers should be <= to this value to be in a good performance range.
- For `Product-Show` the average response is 320ms-400ms. Customers should be <= to this value to be in a good performance range.

Demandware strongly recommends that you check analytics reports each week and after you make code changes to track these metrics.

Page Level Caching

Once the `<iscache>` tag is added to an ISML template, the entire ISML page will be cached for the time specified in the tag.

For example, the page shown will be cached for 1 hour and 30 minutes:

```
1<!--- TEMPLATENAME: cached.isml --->
2<isprint value="${new Date()}" style="DATE_TIME">
3<h1>This part of the page is cached.</h1>
4<iscache type = "relative" hour = "1" minute = "30"><br/>
5This entire page is cached.
6
```

Exercise: Page-Level Caching

1. Create an ISML template named `cachedpage.isml` that has caching enabled for 30 minutes:

   ```
   <iscache type="relative" minute="30" />
   ```

2. Add a `Date` object to the page that prints the current time:

   ```
   <isprint value="${new Date()}" style="DATE_TIME" />
   ```

3. Create a new pipeline named `Caching` or a javaScript controller named `JCaching` to display the above template.
4. Test the template in your SiteGenesis storefront. Refresh your page. Does the time change on refresh?
5. Enable caching on your SiteGenesis site. Retest the template. You may need to wait a minute before you see the page has been cached.

Partial Page Caching

Generally, a single page should not be cached completely. Some parts of the page should be cached, while other parts should not be cached. In this case you need to use remote includes for every part that has unique caching characteristics. Every remote include calls a different pipeline which generates an ISML template, each template having (possibly) different page caching.

The syntax for a remote includes uses the `URLUtils` class to call a remote pipeline with optional parameters appended:

```
<isinclude url="${URLUtils.url('Page-Include', 'cid',
        'COOKIE_TEST')}">
```

You can also use the newer `<iscomponent>` tag to implement a remote include.

demandware
A Salesforce Company

Exercise: Partial Page Caching

1. In the template `cachedPage.isml`, add a remote include call to the `Product-IncludeLastVisited` (pipeline or controller)

   ```
   <iscomponent pipeline="Product-IncludeLastVisited" />
   ```

2. Invalidate the cache in Business Manager.
3. Go to another tab and visit a few products (3 at most).
4. Refresh the `Caching-Start` pipeline or JCaching-Start controller
5. Visit more products on the other browser.

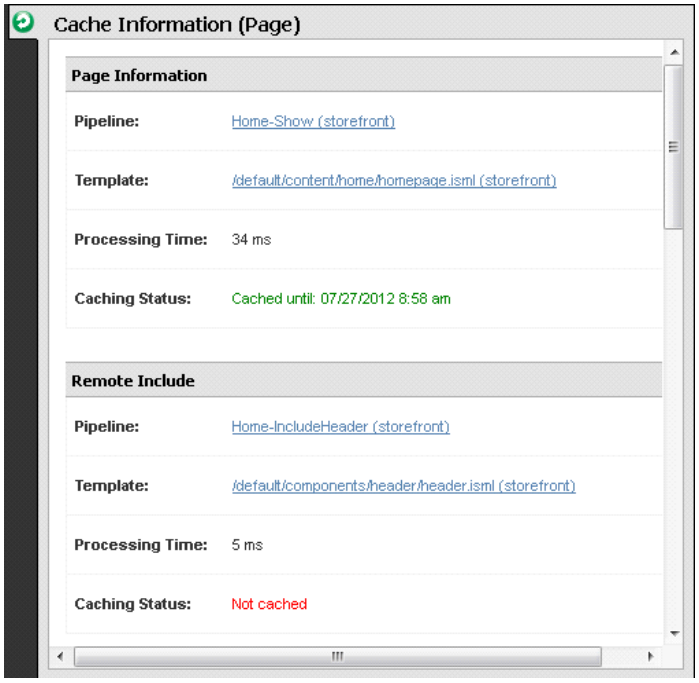
   Result: the time remains unchanged while the last visited products change every time a new product is visited.

6. Once you have finished this exercise, do not forget to turn your caching off again in Business Manager (Administration>Sites>Manage Sites>Site Genesis>Cache)

Using the Storefront Toolkit to Determine Cache Settings

You can enable the Cache Information tool in the Storefront Toolkit to see how partial page caching is implemented for a page:

The page will now show special icons that you can click to reveal how the whole page and its remote includes are cached:



demandware
A Salesforce Company

Exercise: Use the Cache Information Tool

1. Browse the SiteGenesis home page.
2. Turn on **Storefront Toolkit > Cache Information**.
3. Study the cache information for the whole page.
4. Study the cache information for a content slot and open the template to see the cache settings.
5. Study the cache information for the Cart remote include. Why is this page not cached?

demandware
A Salesforce Company

Lesson 11.2: Site Performance

The Pipeline Profiler is a Business Manager tool that provides insight into pipeline and script performance. It tracks pipeline execution metrics, which is a critical component of overall page and site load and performance. This enables you to proactively identify bottlenecks in performance while developing applications.

To track the performance of a pipeline using the Pipeline Profiler, follow these steps:

In Business Manager, select **Administration > Operations > Pipeline Profiler**.

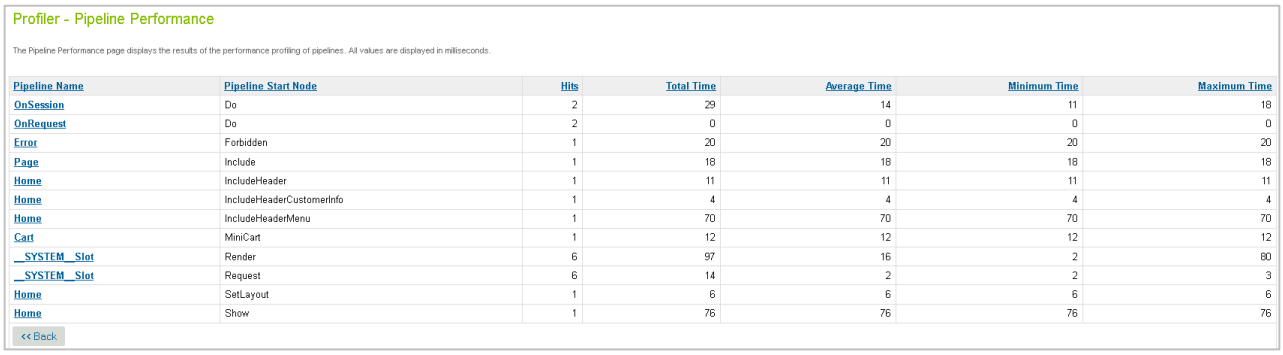
Reset previously collected statistics and turn on the Pipeline Profiler.

Browse specific pipeline in storefront.

Return to profiler and analyze the collected data.

a. Click the link for that site where you want to capture data:

b. You will get a high-level view of response times per pipeline, such as hits, total time for a page to be generated, average time, etc.



Profiler - Pipeline Performance

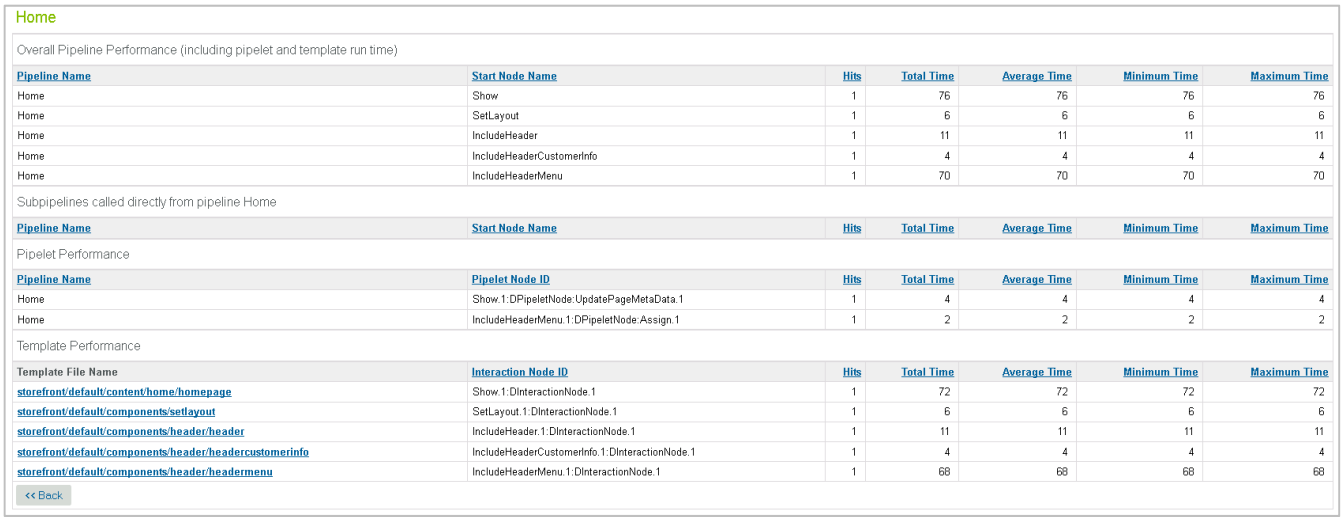
The Pipeline Performance page displays the results of the performance profiling of pipelines. All values are displayed in milliseconds.

Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
OnSession	Do	2	29	14	11	18
OnRequest	Do	2	0	0	0	0
Error	Forbidden	1	20	20	20	20
Page	Include	1	18	18	18	18
Home	IncludeHeader	1	11	11	11	11
Home	IncludeHeaderCustomerInfo	1	4	4	4	4
Home	IncludeHeaderMenu	1	70	70	70	70
Cart	MiniCart	1	12	12	12	12
__SYSTEM__Slot	Render	6	97	16	2	80
__SYSTEM__Slot	Request	6	14	2	2	3
Home	SetLayout	1	6	6	6	6
Home	Show	1	76	76	76	76

<< Back

c. Look for pipelines with high average run times and high hits. These will be the first areas to improve performance.

d. To view data for a pipeline at a more granular level, click on the pipeline name.



Home

Overall Pipeline Performance (including pipelet and template run time)

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Home	Show	1	76	76	76	76
Home	SetLayout	1	6	6	6	6
Home	IncludeHeader	1	11	11	11	11
Home	IncludeHeaderCustomerInfo	1	4	4	4	4
Home	IncludeHeaderMenu	1	70	70	70	70

Subpipelines called directly from pipeline Home

Pipeline Name	Start Node Name	Hits	Total Time	Average Time	Minimum Time	Maximum Time

Pipelet Performance

Pipeline Name	Pipelet Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
Home	Show.1:DPipeletNode:UpdatePageMetaData.1	1	4	4	4	4
Home	IncludeHeaderMenu.1:DPipeletNode:Assign.1	1	2	2	2	2

Template Performance

Template File Name	Interaction Node ID	Hits	Total Time	Average Time	Minimum Time	Maximum Time
storefront/default/content/home/homepage	Show.1:DInteractionNode.1	1	72	72	72	72
storefront/default/components/setlayout	SetLayout.1:DInteractionNode.1	1	6	6	6	6
storefront/default/components/header/header	IncludeHeader.1:DInteractionNode.1	1	11	11	11	11
storefront/default/components/header/headercustomerinfo	IncludeHeaderCustomerInfo.1:DInteractionNode.1	1	4	4	4	4
storefront/default/components/header/headermenu	IncludeHeaderMenu.1:DInteractionNode.1	1	68	68	68	68

<< Back

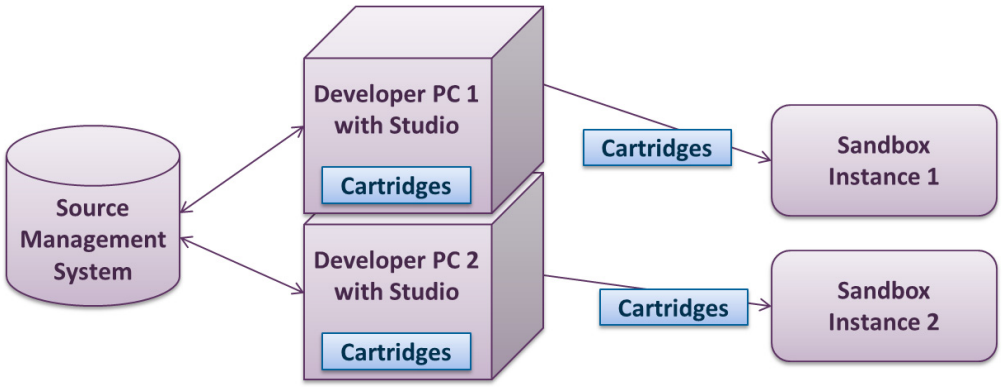
5. Test the pipeline or a different one again.
6. While the pipeline profiler runs you have access also to captured script data.
7. Turn off the profiler and analyze the results.
8. If you make modifications to the pipeline, retest to verify if performance has improved.

Lesson 11.3: Code Replication

Code replication is set and managed in Business Manager. Once you have uploaded a new code version to the PIG staging instance, you can set code replication to occur between staging and development or staging and production.

Code Replication Overview

In a typical development environment, a source management system is used for code version control. Each developer uses their own sandbox for development, while checking in their code to a source management system.

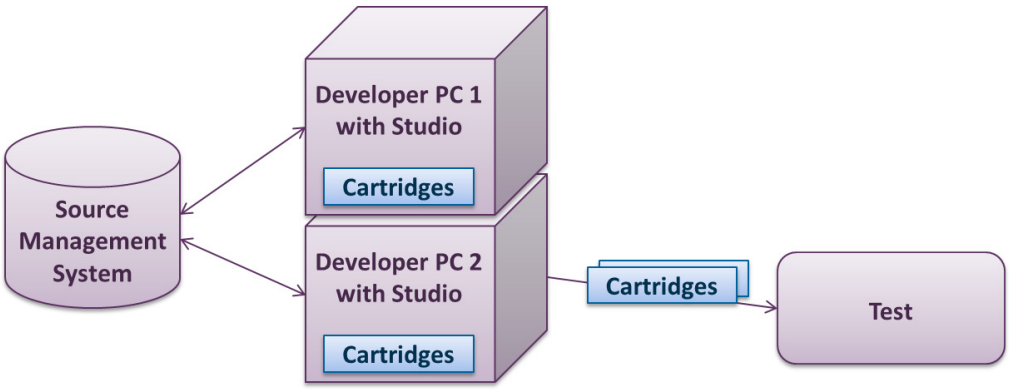


UX Studio integrates with SVN for source management. To learn more about using SVN in UX Studio, view our online webinar in XChange: http://xchange.demandware.com/docs/DOC-2667.

When a developer has tagged a new code version and is ready to upload the new code to staging, he/she creates a new code version on Staging in Business Manager from **Administration > Site Development > Code Deployment** page.

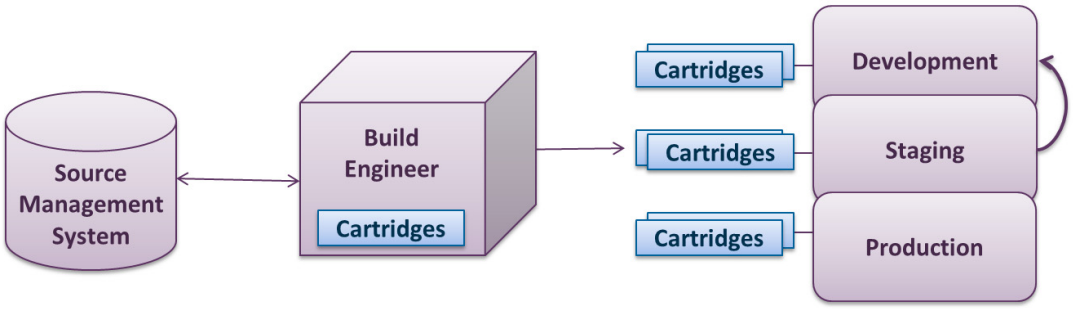
Next, the developer uploads custom cartridges with UX Studio or WebDAV client using 2-factor authentication and tests the storefront in Staging. A rollback to a previous version is available.

For major code changes, it is recommended to use a sandbox for testing:



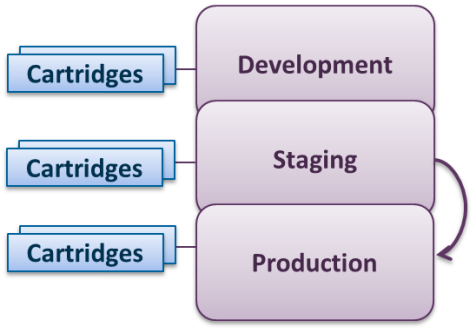
To test in a sandbox, you will need to export site data to the global directory from staging and import it into your sandbox using the Site Import/Export module in Business Manager.

When you need to test code metadata (site preferences, new attributes, etc.), the build engineer replicates from Staging to Development:



This is also good practice for testing processes without impacting the production storefront (i.e. Product import feed).

The last step in code replication is moving code from Staging to Production using Business Manager.



To replicate code from Staging to Development or Staging to Production, follow these steps:

1. Log into the Staging Business Manager with an account that has code replication permissions.
2. Select **Administration > Replication > Code Replication.**
3. Click **New** to create a new replication process.
4. From the **Target** drop-down menu, specify whether the replication process is to Development or Production.
5. Select whether you want to process to run manually or automatically. Click **Next**.
6. Specify what type of replication you want:
   a. Code Transfer & Activation: immediately activates the new code version.
   b. Code Transfer: Only transfers the code.
7. Click **Next**.
8. Click **Start** to start the replication process. Click **Create** to add the replication process to the list.
9. If you selected the process to run manually, you will need to start the job from the list by clicking **Start**.

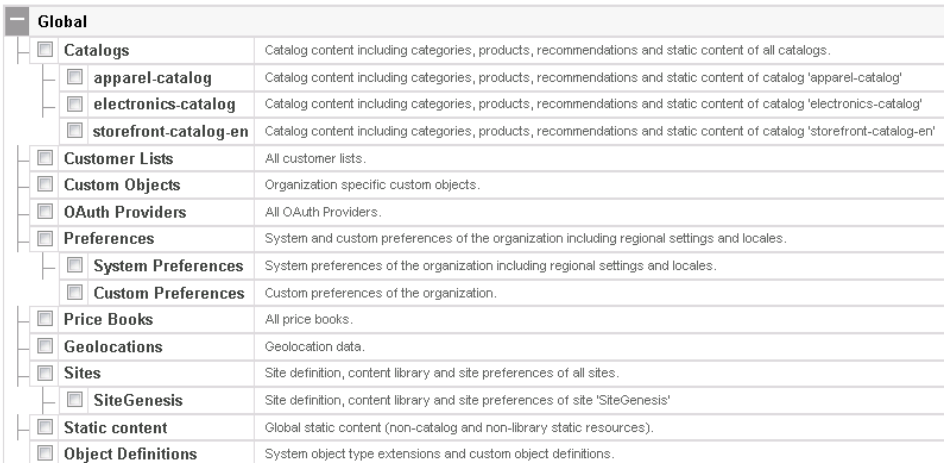
demandware
A Salesforce Company

Lesson 11.4: Data Replication

Data replication is a process to promote merchant edits, product and system objects from Staging to Production (or Development). The best practice is to replicate to development first, verify that data and storefront work and then replicate from staging to production.

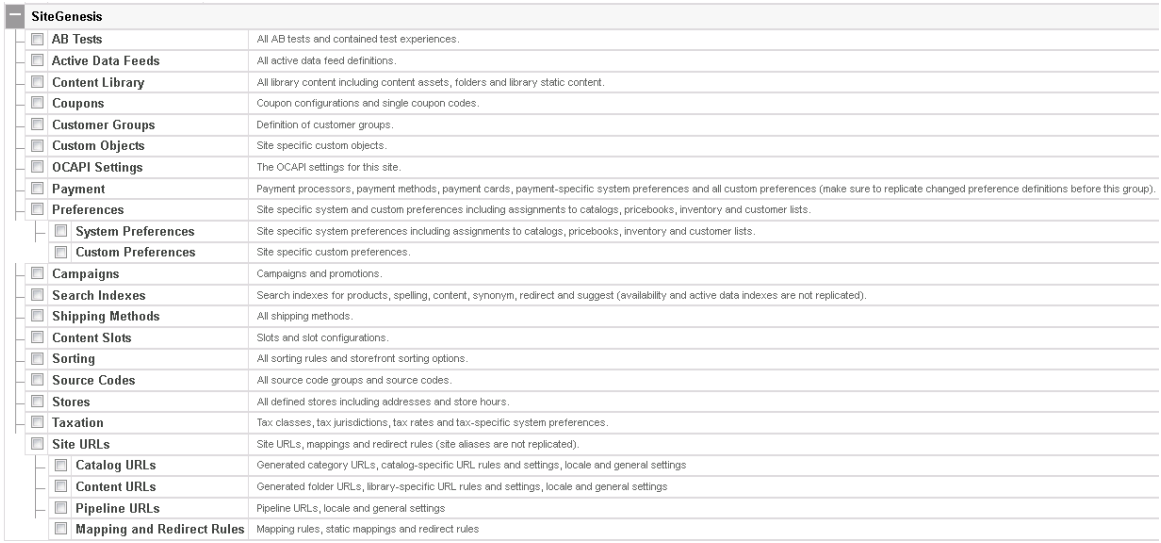
Data can be replicated granularly:

- Organization objects



Global	
Catalogs	Catalog content including categories, products, recommendations and static content of all catalogs.
apparel-catalog	Catalog content including categories, products, recommendations and static content of catalog 'apparel-catalog'
electronics-catalog	Catalog content including categories, products, recommendations and static content of catalog 'electronics-catalog'
storefront-catalog-en	Catalog content including categories, products, recommendations and static content of catalog 'storefront-catalog-en'
Customer Lists	All customer lists.
Custom Objects	Organization specific custom objects.
OAuth Providers	All OAuth Providers.
Preferences	System and custom preferences of the organization including regional settings and locales.
System Preferences	System preferences of the organization including regional settings and locales.
Custom Preferences	Custom preferences of the organization.
Price Books	All price books.
Geolocations	Geolocation data.
Sites	Site definition, content library and site preferences of all sites.
SiteGenesis	Site definition, content library and site preferences of site 'SiteGenesis'
Static content	Global static content (non-catalog and non-library static resources).
Object Definitions	System object type extensions and custom object definitions.

- Per Site objects



SiteGenesis	
AB Tests	All AB tests and contained test experiences.
Active Data Feeds	All active data feed definitions.
Content Library	All library content including content assets, folders and library static content.
Coupons	Coupon configurations and single coupon codes.
Customer Groups	Definition of customer groups.
Custom Objects	Site specific custom objects.
OCAPI Settings	The OCAPI settings for this site.
Payment	Payment processors, payment methods, payment cards, payment-specific system preferences and all custom preferences (make sure to replicate changed preference definitions before this group).
Preferences	Site specific system and custom preferences including assignments to catalogs, pricebooks, inventory and customer lists.
System Preferences	Site specific system preferences including assignments to catalogs, pricebooks, inventory and customer lists.
Custom Preferences	Site specific custom preferences.
Campaigns	Campaigns and promotions.
Search Indexes	Search indexes for products, spelling, content, synonym, redirect and suggest (availability and active data indexes are not replicated).
Shipping Methods	All shipping methods.
Content Slots	Slots and slot configurations.
Sorting	All sorting rules and storefront sorting options.
Source Codes	All source code groups and source codes.
Stores	All defined stores including addresses and store hours.
Taxation	Tax classes, tax jurisdictions, tax rates and tax-specific system preferences.
Site URLs	Site URLs, mappings and redirect rules (site aliases are not replicated).
Catalog URLs	Generated category URLs, catalog-specific URL rules and settings, locale and general settings
Content URLs	Generated folder URLs, library-specific URL rules and settings, locale and general settings
Pipeline URLs	Pipeline URLs, locale and general settings
Mapping and Redirect Rules	Mapping rules, static mappings and redirect rules

A Data Replication process consists of two phases:

- Transfer – long running processes where data is copied from Staging into shadow tables and folders on Production. No changes are shown in storefront.

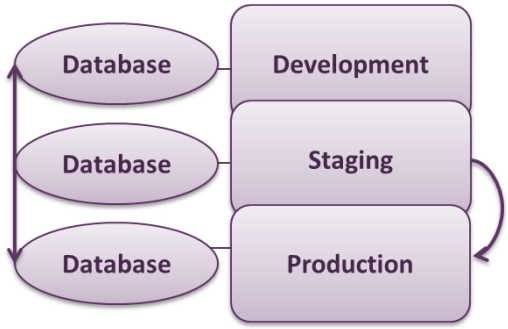
- Publishing – Very fast process. Changes in shadow tables and folders become active, the page cache is purged, and the new version is shown in storefront.

After data has been replicated, a one-time rollback (undo) is possible. This reverses the data to the state of the last successful replication.

You can view the progress of a replication by monitoring the staging logs on the staging and production instance.

Just as code replication is set up in Business Manager, so is data replication. The process is almost identical with the exception of being able to select which data you want to replicate.

Just as code replication can only occur between Staging and Development or Staging and Production, so too is the data replication process only allowed one-way from Staging to the other primary instances.



Best practices can be found in the XChange Portal: https://xchange.demandware.com/videos/1433.

To replicate data from Staging to Development or Staging to Production, follow these steps:

1. Log into the Staging Business Manager with an account that has code replication permissions.
2. Select **Administration > Replication > Data Replication.**
3. Click **New** to create a new data replication process.
4. Specify the target for the data replication process: Development or Production.
5. Select whether you want to process to run manually or automatically. If automatically, specify the date and time the process should run.
6. Specify when you want an email notification and who should receive it. Click **Next**.
7. At the next screen, specify what type of replication you want: Data Transfer & Publishing or Data Transfer.
8. Next, expand the sites to select the site data you wish to replicate. Click **Next**.
9. Click **Start** to create and trigger the process immediately. Click **Create** to add the replication process to the list. Click **Cancel** to go back to the list of replication processes without saving anything.

10. If you clicked **Create**, to start the process, click **Start** from the list of processes.

Knowledge Check

Question	Answer
1. What instance is used to replicate data in a PIG?	
2. What two caching types can be used when using the <iscache> tag?	

demandware
A Salesforce Company

Congratulations

This concludes Developing in Demandware. At this point, you should apply what you have learned to the Demandware platform in preparation for DEV101: Developing in Demandware certification exam. Additionally, you should further your knowledge and skills by taking the DEV201: *Demandware Customization, Integration, and Performance* course and its associated certification.