



# Developing in Demandware

Instructor Guide  
V14.1



### Trademarks & Copyright Acknowledgements

Demandware® is a registered trademark of Demandware Inc.

### Corporate Headquarters

United States  
5 Wall St., Burlington, MA 01803 USA  
Email: [info@demandware.com](mailto:info@demandware.com)  
Phone: +1 (888) 553 9216

### EMEA Offices

Benelux & Nordics  
Newtonlaan 115  
3584 BH Utrecht  
The Netherlands  
Email: [info@demandware.nl](mailto:info@demandware.nl)

France  
54/56, Avenue Hoche, 75008  
Paris, France  
Email: [info@demandware.fr](mailto:info@demandware.fr)

Germany  
Erika-Mann-Str. 57, 80636 München, Germany  
Leutragraben 2-4, 07743 Jena, Germany  
Email: [info@demandware.de](mailto:info@demandware.de)

UK  
City Point, 1 Ropemaker Street, London EC2Y 9HT  
Email: [info@demandware.co.uk](mailto:info@demandware.co.uk)

### APAC Office

China  
Suite 927  
1788 West Nanjing Road  
Shanghai 200040, China  
Email: [info@demandware.cn](mailto:info@demandware.cn)

### Corporate Sites

External site: <http://www.demandware.com>

XChange Community: <https://xchange.demandware.com>

Documentation: <https://documentation.demandware.com>

# 1. Table of Contents

## INTRODUCTORY MATERIAL

<b>Getting Started</b>	<b>6</b>
About This Guide	6
The Program in Perspective	9
Program Preparation	10
<b>Modules</b>	
<b>1. Demandware Architecture</b>	<b>11</b>
SiteGenesis Foundation Architecture	18
Review	27
<b>2. UX Studio Overview</b>	<b>28</b>
Overview	29
Creating a Workspace	30
Creating a Server Connection	33
Importing Projects (Cartridges)	37
Demandware Views	43
Searching for Text in Files	46
Review	48
<b>3. Cartridges</b>	<b>49</b>
What is a Cartridge?	50
Cartridge Types	53
Creating a New Cartridge	57
Creating a Storefront Cartridge	59
Review & Lab	62
<b>4. Pipelines</b>	<b>63</b>
Pipeline Elements	64

Creating a Pipeline	66
Executing a Pipeline	71
Troubleshooting with the Request Log Tool	79
Call Nodes & End Nodes	81
Jump Nodes	87
The Pipeline Dictionary	88
Troubleshooting with the Pipeline Debugger	90
Pipelets	98
Review & Lab	103
<b>5. Internet Store Markup Language (ISML)</b>	<b>105</b>
ISML Templates	106
ISML Tags and Expressions	109
Creating and Accessing Variables	112
Reusing Code in Templates	115
Conditional Statements and Loops	127
Review & Lab	131
<b>6. Content Slots</b>	<b>133</b>
Content Slots	134
Creating Content Slots - Developer Tasks	136
Review & Lab	144
<b>7. Demandware Script (DS)</b>	<b>147</b>
Overview	148
API Packages	150
Using Demandware Script in ISML	153
Script Pipelets	155
Script Debugging	163
Resource API and Resource Bundles	166
Review & Lab	168

<b>8. Forms Framework</b>	<b>169</b>
Overview	170
XML Metadata File	172
ISML Form Template	175
Form Pipeline Elements	177
Review & Lab	182
<b>9. Custom Objects</b>	<b>184</b>
Defining Custom Objects	185
Using Script to Create Custom Objects	192
Review	201
<b>10. Data Binding and Explicit Transactions</b>	<b>202</b>
Data Binding with Forms and Objects	203
Explicit Transaction Handling	208
Review & Lab	211
<b>11. Data Integration</b>	<b>215</b>
Data Integration Overview	216
Data Integration: Simple Feeds	217
Data Integration: Web Services	232
<b>12. Integration Framework</b>	<b>245</b>
Integration Framework Overview	246
<b>13. Deconstructing SiteGenesis</b>	<b>252</b>
Using the Storefront Toolkit: Page Information	254
Storefront Files	256
Product Detail Page	257
Shopping Cart & Checkout Processing	260
Review & Lab	276
<b>14. Site Maintenance</b>	<b>277</b>
Site & Page Caching	278

## Instructor Guide

Site Performance	287
Code Replication	290
Data Replication	294
Review & Lab	298

## 2. Getting Started

### About This Guide

---

#### What's the purpose of this guide?

This leader guide provides a master reference document to help you prepare for and deliver the program.

---

#### What will I find in the guide?

This leader guide is a comprehensive package that contains the workshop delivery sequence, checklists of necessary materials and equipment, presentation scripts and key points to cover, and instructions for managing exercises, case studies, and other instructional activities.

---

#### How is this guide organized?

This section, “Getting Started,” contains all of the preparation information for the program, such as learning objectives, pre-work, required materials, and room set-up.

Following this section is the “Training At A Glance” table. This table can serve as your overview reference, showing the module names, timings, and process descriptions for the entire program.

Finally, the program itself is divided into *modules*, each of which is comprised of one or more *lessons*. A module is a self-contained portion of the program, usually lasting anywhere from 50 to 180 minutes, while a lesson is a shorter (typically 5-40 minute) topic area. Each module begins with a one-page summary showing the Purpose, Time, Process, and Materials for the module. Use these summary pages to get an overview of the module that follows.

---

## About This Guide, continued



---

### How is the text laid out in this guide?

Every action in the program is described in this guide by a text block like this one, with a margin icon, a title line, and the actual text. The icons are designed to draw attention to “what to do and how to do it.” For example, the icon to the left indicates that you, the instructor, say something next. The title line provides a brief description of what to do, and is followed by the actual script, instruction set, key points, etc. that are needed to complete the action.

A complete list of the margin icons used in this guide is provided on the following page.

---

### IMPORTANT NOTE

You may also occasionally find important notes such as this one in the text of this guide. These shaded boxes provide particularly important information in an attention-getting format.



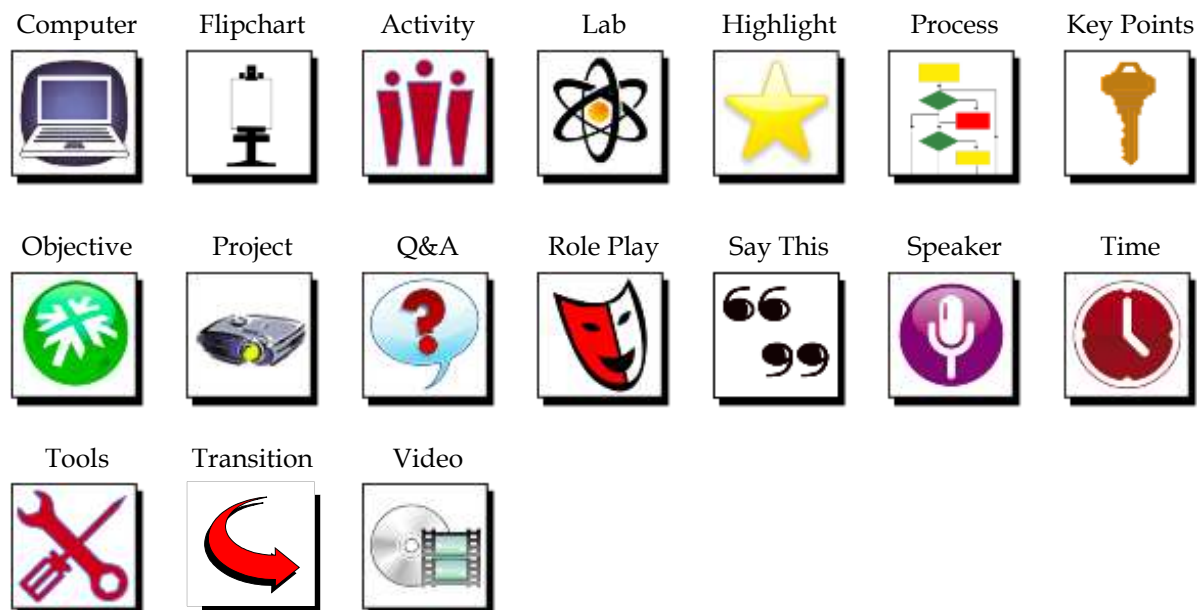
## About This Guide, continued

### Graphic Cues

#### Module Blocks



#### Lesson Blocks



# The Program in Perspective



## Why a course?

This five-day intensive course teaches developers how to modify and customize the Demandware reference application, SiteGenesis. Participants learn about core Demandware programming concepts, files, and the scripting language used throughout SiteGenesis.



## Learning Objectives

After completing this program, participants will be able to:

- Work with cartridges
- Create and work with pipelines, the foundation for all business logic
- Program with Internet Store Markup Language (ISML)
- Program with Demandware Script and use the Demandware APIs
- Work with Demandware Forms
- Work with custom objects to create data
- Use data binding to access and update persistent data
- Update and integrate data from other systems
- Deconstruct SiteGenesis
- Customize core objects in the system

## Program Timing

This course is designed to take 4 days of instruction with 1 half-day of review and testing, followed by a certification exam.

## Number of Participants

For on-site delivery and one instructor, the maximum number of participants should not exceed 15.

# Program Preparation

## Pre-Work

Prior to beginning this course, each student is required to do the following:

- Register in the training under the education space in XChange for the respective upcoming course
- Watch the online training course **Demandware Essentials**, which can be accessed in XChange, the Demandware Community portal.
- Download and install Eclipse for Java EE, accessible at [eclipse.org](http://eclipse.org)

## Required Materials

Every student must have a personal laptop or desktop with administrator rights.

## Room Set-Up

For on-site course delivery:

- Overhead projector connected to instructor's computer
- High-speed internet access for instructor and every student

## Instructor Preparation

Prior to delivering this course:

- Have a Demandware sandbox ready for each student.
- Ensure all students know the location of the training and start/end times for each day's session.
- Have all solutions cartridges uploaded and available in the instructor sandbox.

# 1.Demandware Architecture



---

## Goal

This module is intended to give you an overview of the Demandware SaaS Platform Architecture.

---



---

## Time

1.5 Hours

---



---

## Overview

We will discuss Software as a Service, Demandware Realms, Primary Instance Groups and SiteGenesis, the reference storefront application. We will also create a new empty site, import a copy of SiteGenesis into a site, and configure the settings for the imported site.

---



---

## Materials Needed

Each student will need a training sandbox instance and a login for it.

---



### Introduction

Demandware is a SaaS platform specializing in eCommerce.

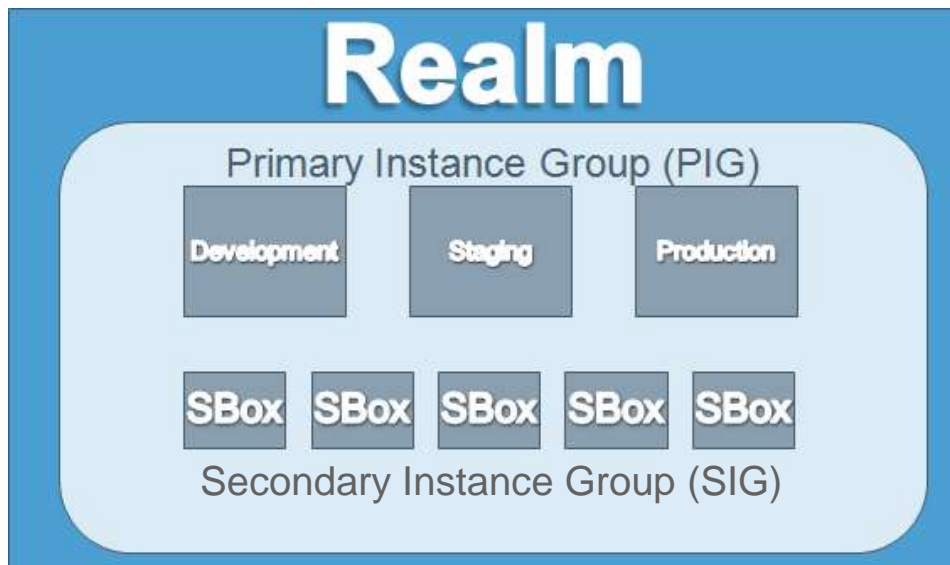
The term SaaS (Software as a Service) is used to define the method of delivering software to a customer. Software and applications are hosted on the internet and delivered to customers who pay for access. Software updates are performed on the hosted platform without customer intervention required.

### Point of Delivery

A Point Of Delivery or POD refers to the Demandware hardware deployed in a data center. A POD contains multiple state-of-the-art application servers, database servers and the clustering and backup infrastructure to support multiple client Realms.

### Realm

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.

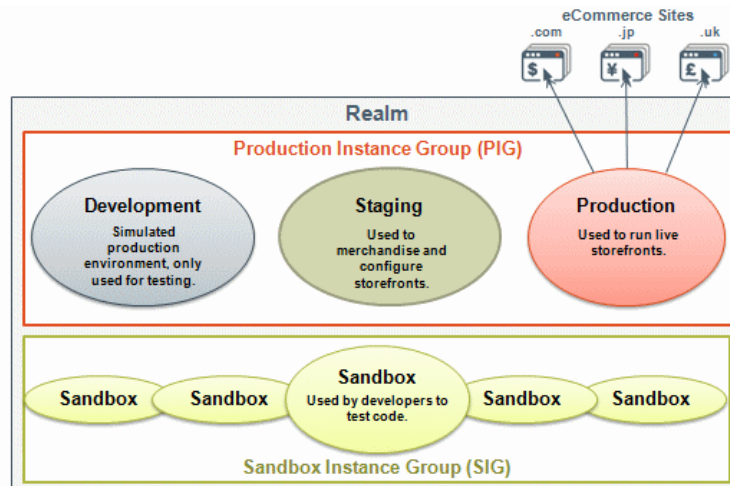


# SaaS, POD, Realm, PIG and SIG, continued



## Instance:

A single running Demandware server, hosted in the Demandware hardware infrastructure



## Primary Instance Group

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

1. **Production** – the production environment, or also known as ‘live’ instance, is used as the actual eCommerce storefront.
2. **Staging** – the staging instance is used for uploading code and preparing it for testing in the Development instance.
3. **Development** – can be used to test processes without impacting the production storefront (i.e. Product import feed)

## Secondary Instance Group

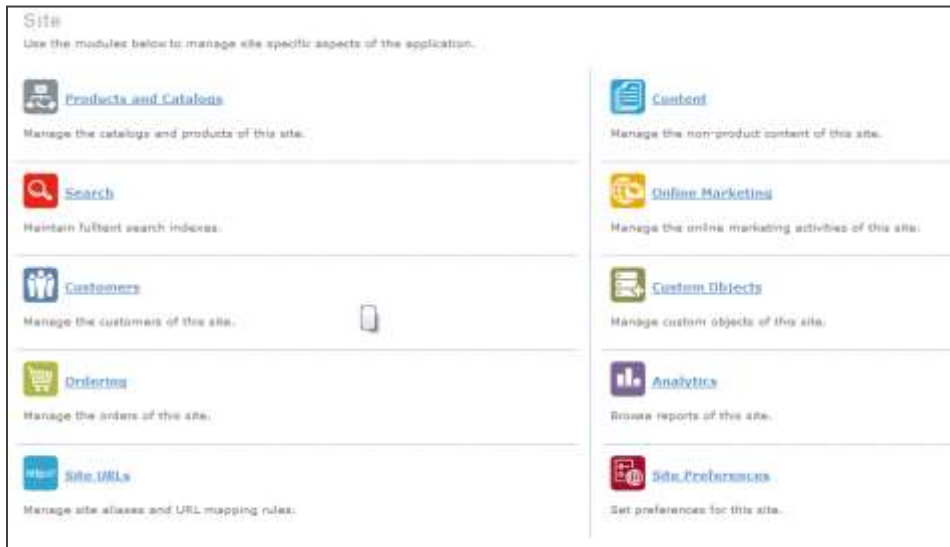
Every Realm includes a Secondary Instance Group (SIG) which includes 5 Demandware **Sandboxes**. Sandboxes are used by developers to develop and test their code. They are not as powerful as PIG instances in performance, memory and storage.

# Business Manager Overview



## Introduction

Business Manager is the user interface used by both merchants and developers for managing administrative tasks within their organizations. Every Demandware instance has a Business Manager portal for each organization. For instance, a marketer would log into the Business Manager portal in the Staging instance, Sites organization to manage the live site or sites for a company.



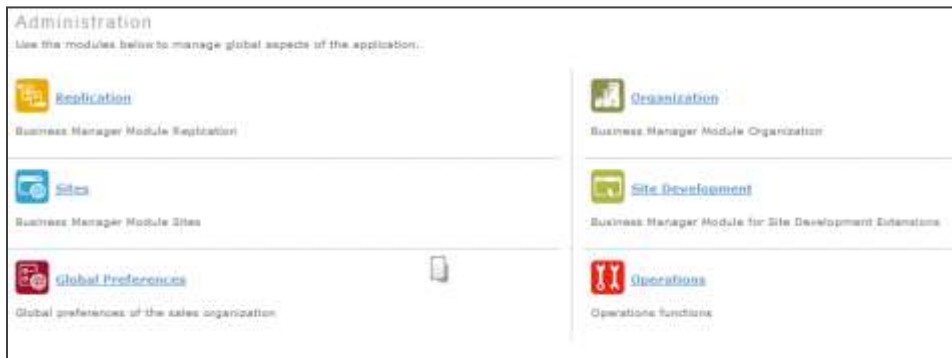
Merchandisers use Business Manager to control:

- Products & Catalogs
- Content
- Marketing campaigns
- Search settings
- Customers
- Site Analytics
- Site URL's
- Site Preferences

## Business Manager Overview, continued

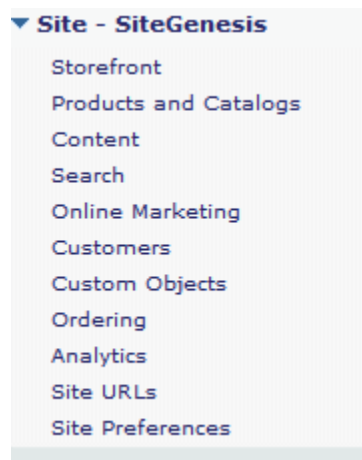
**Developers use Business Manager to manage:**

- Code & Data Replication
- Code Versioning
- Site Development
- Data Import/Export
- Global Preferences for all sites /organization



### Run the Business Manager Navigation Activity

Have your students click on each of the Merchant menu links in SiteGenesis so they become aware of some of the tasks built into that area:





Then have them click on the Administration links:



## An Empty Site



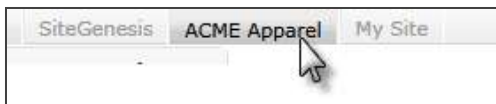
### Introduction

When you first log into your Business Manager portal for the Sites organization, you will have no storefront to manage. By default, the Sites organization is deployed with no sites. You must either create a new empty site (which will have no data) or create a copy of SiteGenesis (discussed in the next chapter) and import it into your Sites organization.

### 1.1. Exercise: Create an empty site

To create a new empty site, follow these steps:

1. Log into Business Manager.
2. Click the **Administration** link on the left-side of the page:
3. Click **Sites**
4. Click **Manage Sites**
5. Click **New**
6. Enter an **ID** for the site as **Training**. Do not use any spaces. This is a required field.
7. Enter a **Name** as **Training** as well. This can be any text string. This is a required field.
8. Click the drop-down button next to **Currency** and select the site's currency. You can only have 1 currency per site. This is a required field.
9. Click the **Apply** button when you are finished. You will be able to configure your new site.
10. To view your new storefront, click the name of the site on the site list on the main Business Manager menu. (The screenshot below is showing an example for another site **ACME Apparel**. You will see your site as **Training**).



11. Click the **Storefront** link on the left. A browser window will open up to the storefront URL:



# SiteGenesis Foundation Architecture



## Introduction

Demandware provides a sample site, called SiteGenesis, which you can use as the basis of your own custom sites. SiteGenesis is a resource for both developers and merchants. For developers, it provides sample code – pipelines, scripts, and ISML templates – which you can inspect to see current best practice. For merchants, it provides sample configurations for catalogs, categories, products, and so on. In short, SiteGenesis is a full featured demonstration eCommerce site, which you can use to explore the Demandware platform and its capabilities.



In order to get the latest version of SiteGenesis the read-only SiteGenesis package needs to be imported as a sample site in every Sandbox instance.

### Caution !

Never import SiteGenesis into an instance in your Primary Instance Group (PIG). You can easily **import SiteGenesis into each instance in your SIG**. However, if you import SiteGenesis into a sandbox that contains other customized sites, you may overwrite existing attributes and lose data, so 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, make sure you **import SiteGenesis first before importing the custom sites**. By importing SiteGenesis first, you ensure that custom attributes for your custom sites are retained if there are conflicts, as your custom attributes will overwrite the imported SiteGenesis custom attributes. 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.

## 1.2. Exercise: Import SiteGenesis

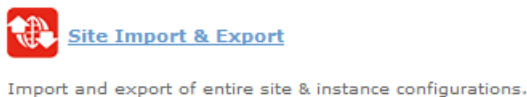
We will learn how to import SiteGenesis from a SiteGenesis package

**To import the latest version of SiteGenesis , follow these steps:**

1. Log into the Business Manager portal in the Sites organization.
2. Click on the **Administration** link.
3. Click the **Site Development** button.



4. Click the **Site Import & Export** link.



5. Depending on whether you want to import a site from a local copy on your machine or from a remote instance, follow the appropriate steps below:

Import locally:

- Be sure the **Local** radio button is selected.

Site Import & Export

**Site Import & Export**

This page allows you to export the current configuration of your organization including all of its site

**Import**

**Upload Archive:**

☒ Local ☐ Remote

Select	Name	Location
<input type="radio"/>	SiteGenesis Demo Site	

- You can import the SiteGenesis Demo Site below or click the **Browse...** button to retrieve another file from your hard drive, followed by the **Upload** button.
- You will be asked:



- Click **Ok**.

- You can view the status of the import in the **Status** section of the page.

Status	
Select All	Process
<input type="checkbox"/>	Site Import (demosite-creation.zip)
Refresh	
Showing 1 - 1 of 1 items.	

When your import is finished, you will see the new site listed in Business Manager. You will also receive an email that the job is complete.

### Import remotely:

- Select the **Remote** radio button.
- Enter all required data for accessing the remote server account:

Site Import & Export

**Site Import & Export**

This page allows you to export the current configuration of your organization.

**Import**

**Upload Archive:**

☐ Local ☒ Remote

**Hostname:\***

**Login:\***

**Password:\***

Connect

- Click the **Connect** button. You will be able to view the importable files from the remote server.
  - Select the import file you want to use by clicking the radio button next to the file name.
  - Click **Import**.
- a. You will be asked:

	<b>Are you sure that you want to import the selected archive?</b>
---	---

- b. Click **Ok**.
- c. You can view the status of the import in the **Status** section of the page.

Status	
Select All	Process
<input type="checkbox"/>	Site Import (demosite-creation.zip)
Refresh	

6. When your import is finished, you will see the new site listed in Business Manager. You will also receive an email that the job is complete.



### Site Configuration

After you have imported the SiteGenesis site, you will need to disable site caching in order to see your code changes immediately in the site. This is something that developers do 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.

## 1.3. Exercise: Disable caching for SiteGenesis site

1. From Business Manager, click **Administration**→**Sites**→**Manage Sites**→**SiteGenesis** (or name of your site if not SiteGenesis).
2. Click the **Cache** tab.

Manage Sites > SiteGenesis - General

General Settings Cache Security Robots

SiteGenesis - General

Fields with a red asterisk (\*) are mandatory.  
Click **Apply** to save the details. Click **Reset** to revert to the last saved details.

ID: SiteGenesis

Name\*: SiteGenesis

Currency\*: US Dollar

Taxation\*: Net

Status: Live

Description:

3. Set the **Time to live** value to zero and uncheck the **Enable page caching** setting.

Static Content and Page Caches

The 'Time to live (TTL) of static content' field defines the time span (in second) of static files set this field to 0.

Click 'Invalidate' to start invalidation of the static content and page cache (will t

Time to live (TTL) of static content: 0

Last invalidated: 3/10/11 12:14:28 pm Invalidate

Page Cache Only

Check the 'Enable page caching' box to allow the web server to cache selected

Click 'Invalidate' to start invalidation of cached dynamic storefront pages (will t

Enable page caching: ☐

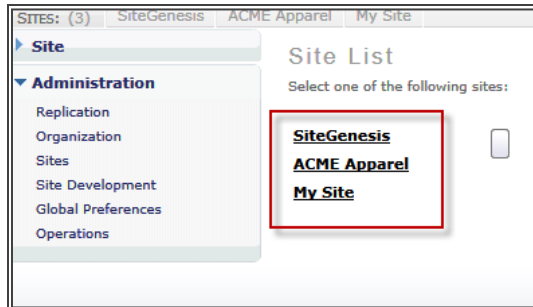
Last invalidated: 3/10/11 12:14:28 pm Invalidate

4. Click the **Apply** button.
5. It is also a good idea to invalidate the cache at this stage. Press the two **Invalidate Page Cache** buttons one after the other.
6. Repeat the exercise for **Training** site as well.
7. In **Site** → **SiteGenesis** → **Site Preferences** → **Storefront URLs** uncheck the **"Enable Storefront URLs"** (will be explained in chapter "Deconstructing SiteGenesis")



## 1.4. Exercise: Re-Index Search for SiteGenesis

1. Log into Business Manager
2. Click on the site you wish to index (**SiteGenesis**) from your site list:



3. Click **Search** → **Search Indexes**
4. Select the top checkbox to select all the indexes:

<input checked="" type="checkbox"/>	Index Type / Locale
<input checked="" type="checkbox"/>	Product / Spelling Indexes
	Product Index
<input checked="" type="checkbox"/>	Default
	Spelling Index
	Default
<input checked="" type="checkbox"/>	Content Index
<input checked="" type="checkbox"/>	Default
<input checked="" type="checkbox"/>	Redirect Index
<input checked="" type="checkbox"/>	Default
<input checked="" type="checkbox"/>	Synonym Index
<input checked="" type="checkbox"/>	Default
<input checked="" type="checkbox"/>	Suggest Index
<input checked="" type="checkbox"/>	Default
<input checked="" type="checkbox"/>	Availability Index
<input checked="" type="checkbox"/>	Non-Localized
<input checked="" type="checkbox"/>	Activedata Index
<input checked="" type="checkbox"/>	Non-Localized
Reindex	

5. Click the **Reindex** button.

The indexes will begin rebuilding. When they are complete, the status will change from *Rebuilding* 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.

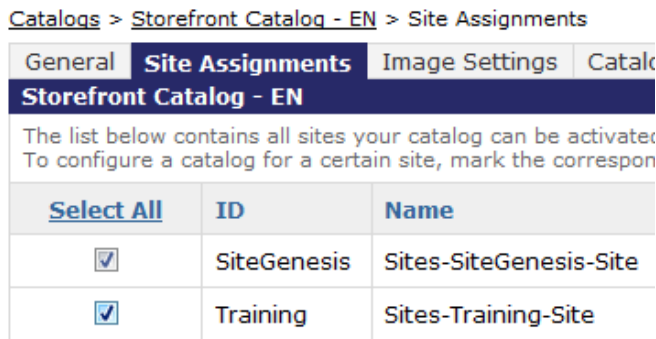
The sharing of catalogs allows for master catalogs containing all products to be shared at the organization level, and for specific site catalogs to contain categories and products to control navigation for each site.

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.

## 1.5. Exercise: Sharing a Catalog Between Sites and Setting a Product Offline

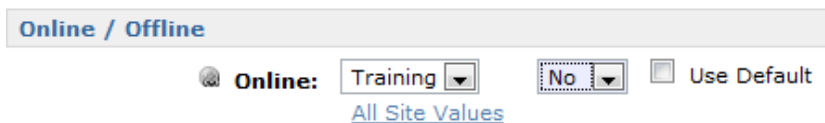
1. From Business Manager, click **SiteGenesis** → **Products and Catalogs** → **Catalogs**.
2. Open the **storefront-catalog-en** catalog, click on **Edit**.
3. Share this catalog with the **Training** site: open the **Site Assignments** tab, check **Training** site and **Apply** the change:



4. Under **Training** → **Search** → **Search Indexes**, set the **Incremental Index Updates** checkbox:

<input type="checkbox"/>	Index Type / Locale	Status	Documents	Index Size	Last Update
<input type="checkbox"/>	Product / Spelling Indexes	<input type="checkbox"/> Scheduled Index Rebuild *			
		<input checked="" type="checkbox"/> Incremental Index Updates			

5. Click **Apply Index Settings** at the bottom of the table: this will ensure that indexing occurs automatically after any changes that require it.
6. Use **Training** → **Products and Catalogs** → **Products** to find the **P0048** product.
7. **Lock** the product so it can be edited.
8. Locate the site attribute *Online/Offline* and set it to **No** for the **Training** Site only:



9. Apply your changes and verify that the product is not available on the **Training** site. (Go to SiteGenesis and search for P0048). You might have to wait for incremental indexing to complete to be able to see the results.

# Review

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



**Transition to UX Studio Overview**

## 2.UX Studio Overview



### Goal

The purpose and goal of this module is to perform common tasks with UX Studio.

---



### Time

1.5 Hours

---



### Overview

We will demonstrate how to create a new workspace, server connection, and getting around the user interface.

---



### Materials Needed

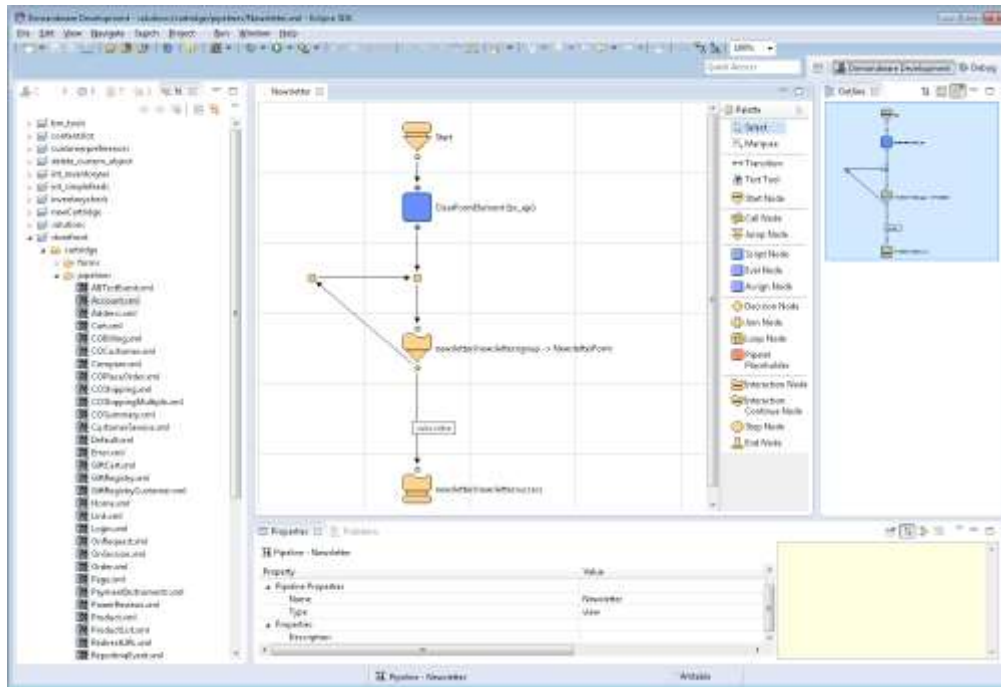
All course solutions cartridges from the instructor sandbox: solutions, int\_simplefeeds, test\_simplefeeds, etc.

# Overview



## Introduction

UX Studio is an Integrated Development Environment (IDE) used for programming in the Demandware platform.



UX Studio is a plugin built on the Eclipse open-source development platform ([www.eclipse.org](http://www.eclipse.org)). Eclipse is used by many Java developers to build Java applications. It is not necessary to know Java to use UX Studio. This topic will introduce you to the UX Studio navigation structure as well as how to create a workspace and server connection.

# Creating a Workspace



## Introduction

A workspace is an Eclipse-specific local directory that contains Eclipse projects. Normally Eclipse projects are connected to Java source directories (packages). In Demandware Studio projects are different: they either define a connection to a Demandware instance or they point to a Demandware cartridge. They are never used to compile Java projects since Java code is not used in Demandware application programming.

Each workspace should have only 1 Demandware server connection (covered later in this module). For example, if you are a developer working on numerous client projects, you will want to create a separate workspace for each client. Each client workspace will then have only 1 specific server connection.



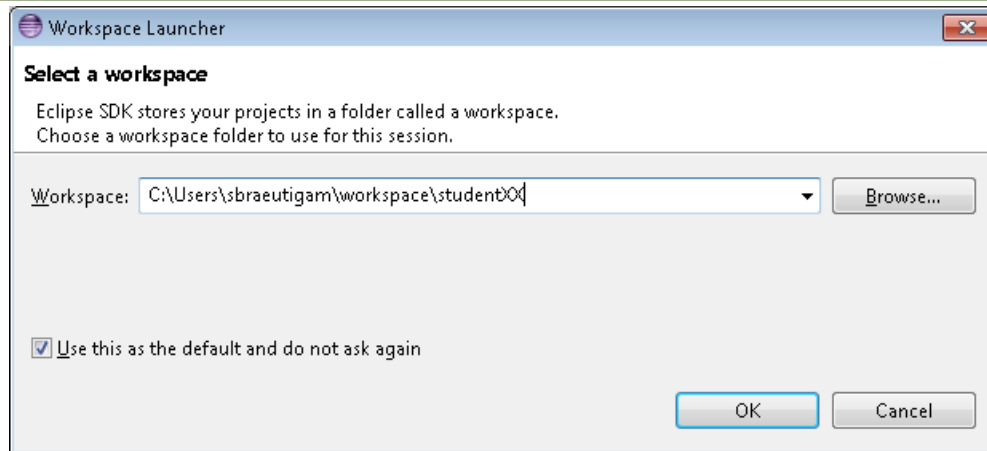
## Run the Create a Workspace activity.

Demonstrate how to create a new workspace.

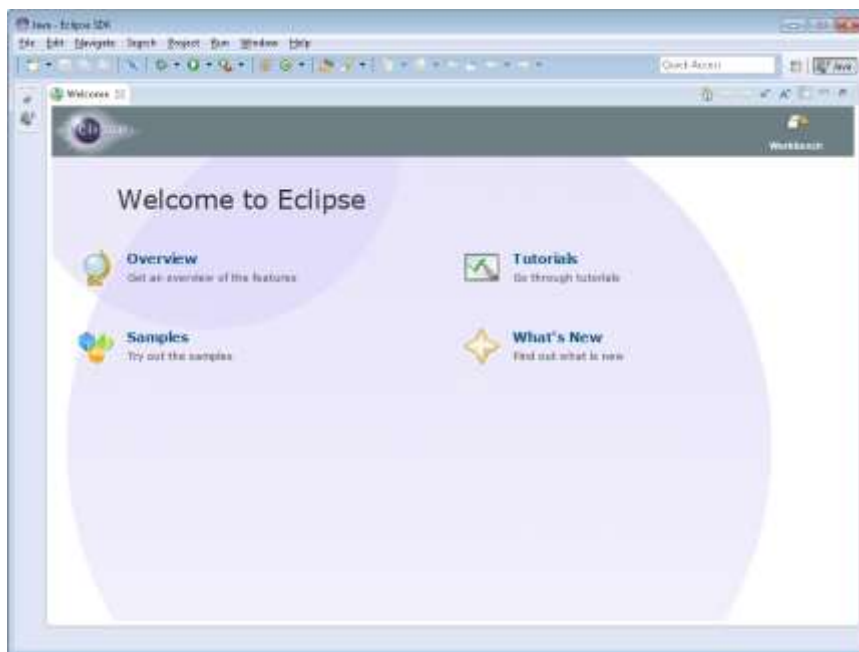


**To install the UX Studio plugin into Eclipse and to create a new workspace (when using UX Studio for the first time), follow these steps:**

1. The first time you use the application, you will be prompted to create a new workspace name. Give your workspace a name that references the client you are working with.



2. Eclipse will first display the Welcome message in the main working area.



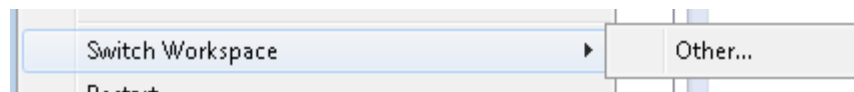
3. Open the Install dialog box via 'Help → Install New Software...'
4. Select the Add... button. The Add Repository dialog appears.
5. Enter Demandware UX Studio in the Name field.
6. Enter one of the following URLs into the Location field. Choose the URL for your version of Eclipse :
  - Indigo - [http://updates.demandware.com/uxstudio\\_ea\\_14.1/3.7](http://updates.demandware.com/uxstudio_ea_14.1/3.7)
  - Juno - [http://updates.demandware.com/uxstudio\\_ea\\_14.1/4.2](http://updates.demandware.com/uxstudio_ea_14.1/4.2)
  - Kepler - [http://updates.demandware.com/uxstudio\\_pr/4.3](http://updates.demandware.com/uxstudio_pr/4.3)
7. Give a name to the URL
8. Select 'Demandware' from the list and select "Next". At this point Eclipse will compare the UX Studio requirements with what is available to ensure



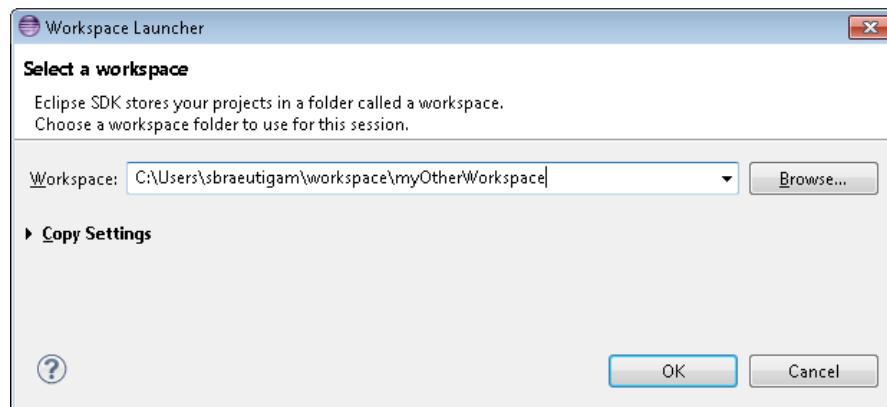
- compatibility. After this process is complete, the Install Details dialog appears (see Calculating Requirements note).
9. Select “Next” in the Install Details dialog. The Review Licenses dialog appears.
  10. Select the license agreement radio button and then select Finish. The Installing Software dialog appears showing the progress of the installation.
  11. Select OK in the Security Warning dialog.
  12. Select “Yes” when prompted to restart Eclipse.
  13. UX Studio is now installed. You can now use the “Demandware Development” perspective in the upper right corner.
  14. Optionally you can get the Font enhancer from <http://eclipse-fonts.googlecode.com/svn/trunk/FontsUpdate/>, using the same installation method like our plugin.

If you already have a workspace and need to create a new one, follow these steps:

1. From the main menu in UX Studio, click on **File→Switch Workspace→Other...**



2. The Workspace Launcher window will open.
3. Enter a new workspace name in the file path.



4. Studio will close and reopen.
5. You will need to accept the terms of the license again and click ‘OK’.

# Creating a Server Connection

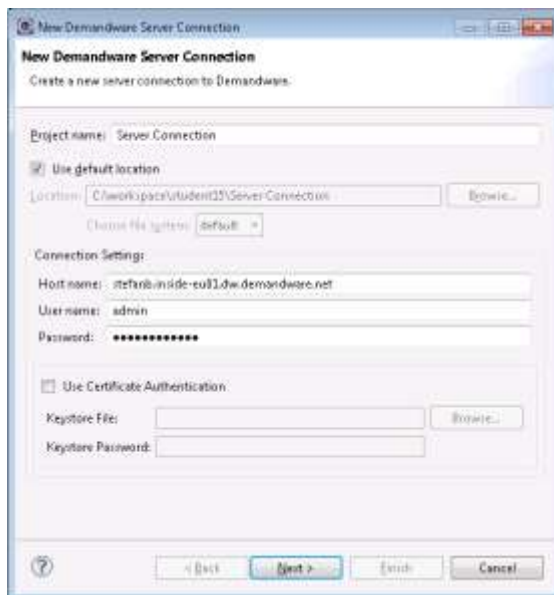


## Introduction

In order to upload your code to a Demandware server, you will need to create a server connection in UX Studio. A server connection allows you to push your code to the server instance but you will not be able to pull the code onto your personal computer from the Demandware server. The connection is a 1-way push only.

### 2.1. Exercise: Create a new server Connection

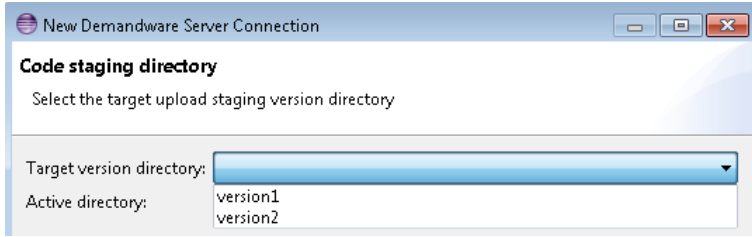
1. From UX Studio, click **File→New→Demandware Server Connection**. The new server connection box opens.
2. Complete it as follows.
  - a. In the **Project name** and **Host name** fields, use the host name provided by your instructor or client:
    - i. **student##.training-eu.dw.demandware.net**, where # is unique for every student.
    - ii. **partner##.cloud01.pod3.demandware.net** where partner## varies by partner company
  - b. Enter your password.



3. Click **Next**.
4. A security warning regarding an invalid certificate for your sandbox shows up. Click **Yes** to continue.



5. Select **version1** as the target version you want to upload your files to:



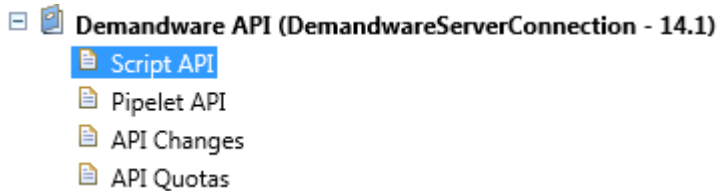
6. Click **Finish**.

Your connection project is now connected to your sandbox and will be used to upload any cartridge projects to that sandbox, as seen later in this module.

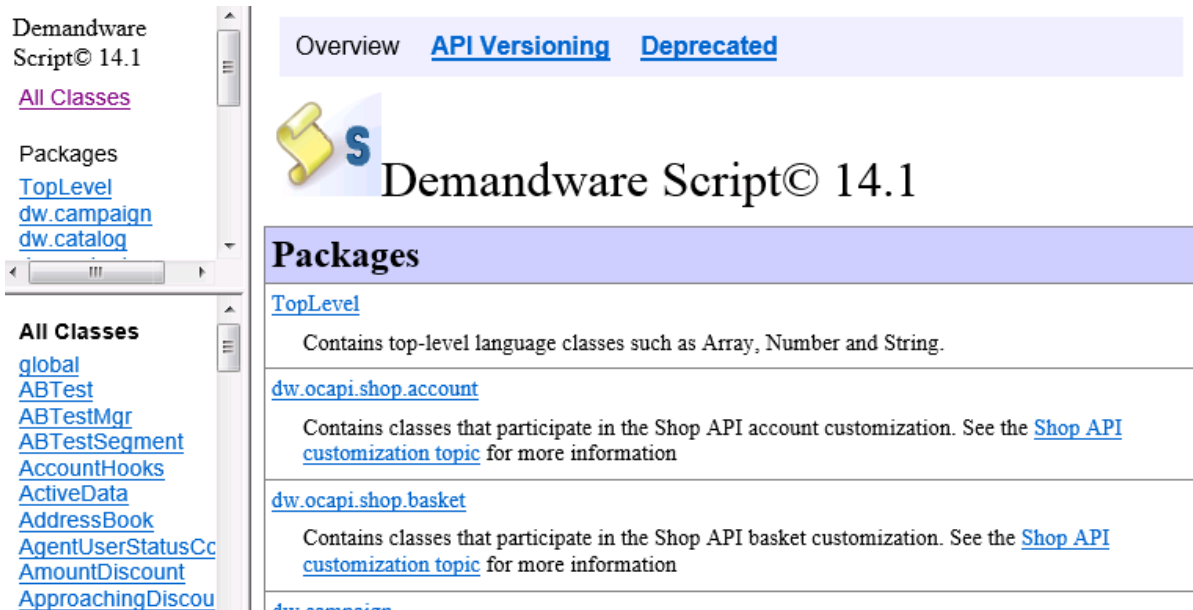
## 2.2. Exercise: Run the View Demandware Help.

To open the Demandware API Javadoc:

1. From UX Studio, click **Help** → **Help Contents**.
2. Expand the **Demandware API** link.
3. Select any of the available help items:



4. The first two items offer Javadoc-style help:



# Importing Projects (Cartridges)



## Introduction

A project in UX Studio is a cartridge. We will cover cartridges in the next module. All you need to know for now is that a cartridge is a folder with specific sub-folders and files inside of it.

There may be times when you will need to add other cartridges (projects) to your workspace. The process steps below will walk you through importing those projects to your workspace.



## Importing Projects

Walk the students through importing the solutions cartridges into their workspace. Before you do this, you must:

1. Please provide all students with a copy of the **DevelopingInDemandwareSolutions.zip** file.
2. Have it extracted to the C:\ drive or some other favorite folder.

This creates a C:\projects directory where all solution cartridges are located.

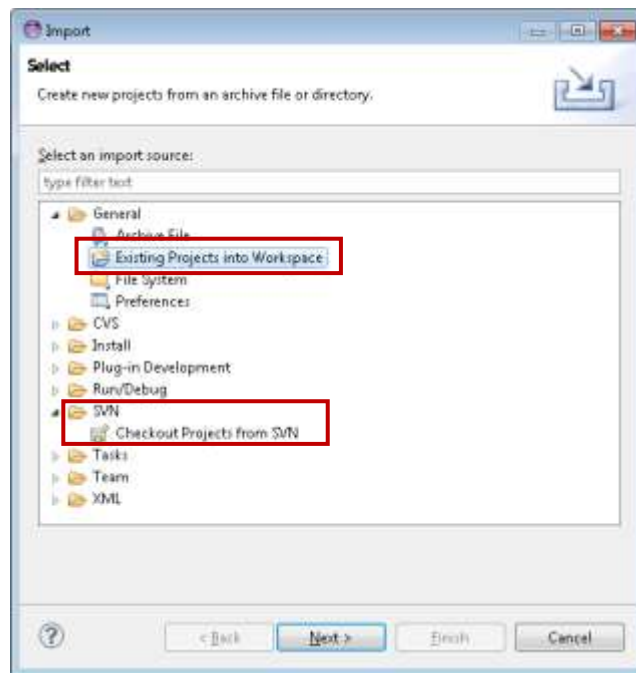
### 2.3. Exercise: Import a projects in Studio

1. From within UX Studio, click on **File→Import...** an import window will open.
2. From the Import window, click to expand the **General** menu.
3. Click the **Existing Projects into Workspace** option. If you have an SVN server, you could import projects directly from a repository, which is the most common way to obtain cartridges when you are working on a project. Both options are highlighted below.
4. If you don't see the SVN import / export functionality you need to install the plugin via Help → "Install New Software" and mention the URL

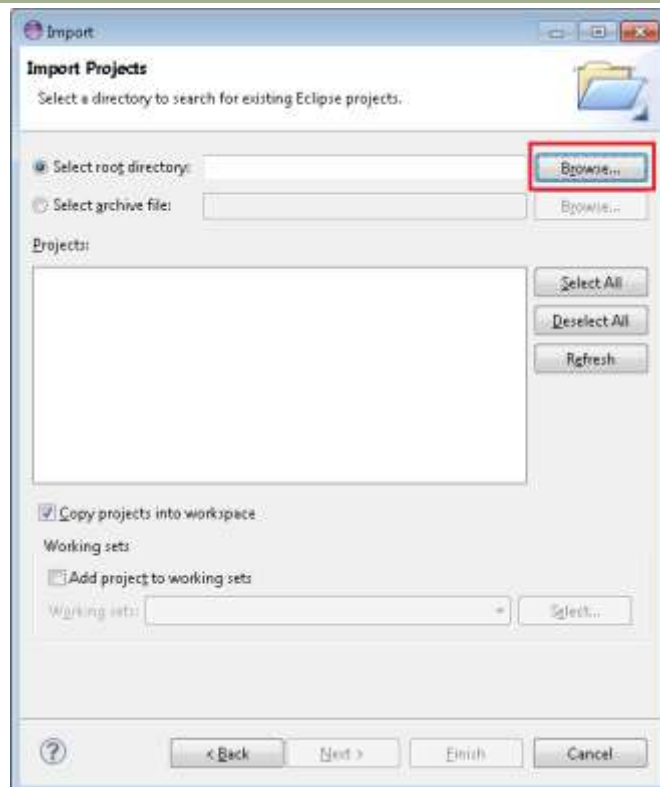
Name: Subclipse 1.8.16 (Eclipse 3.2+)

URL: [http://subclipse.tigris.org/update\\_1.8.x](http://subclipse.tigris.org/update_1.8.x)

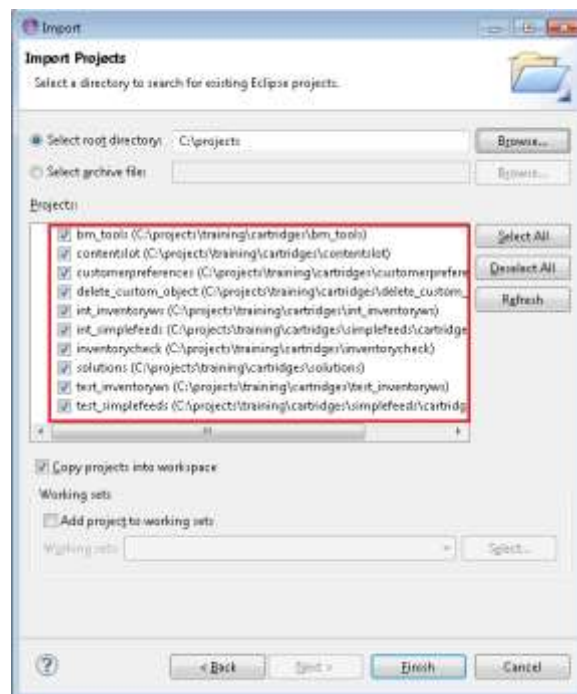
5. For this course we do not use an SVN repository since you are not sharing code:



6. Click '**Next**'.
7. In the next window, click to '**Browse...**' button.



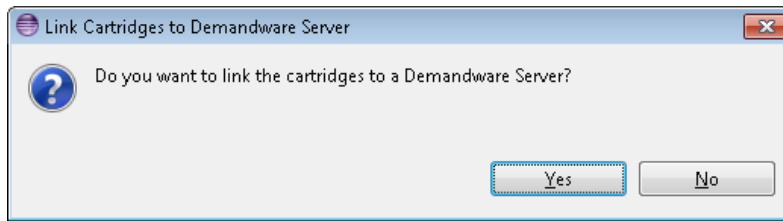
8. Locate the folder on your hard drive where cartridges are located. Your instructor will provide a zip file with all solution cartridges for you to install locally. Click **OK**.
9. Any cartridges in the folder structure (including subfolders) will be displayed in the **Projects** box. Click **Select All**:



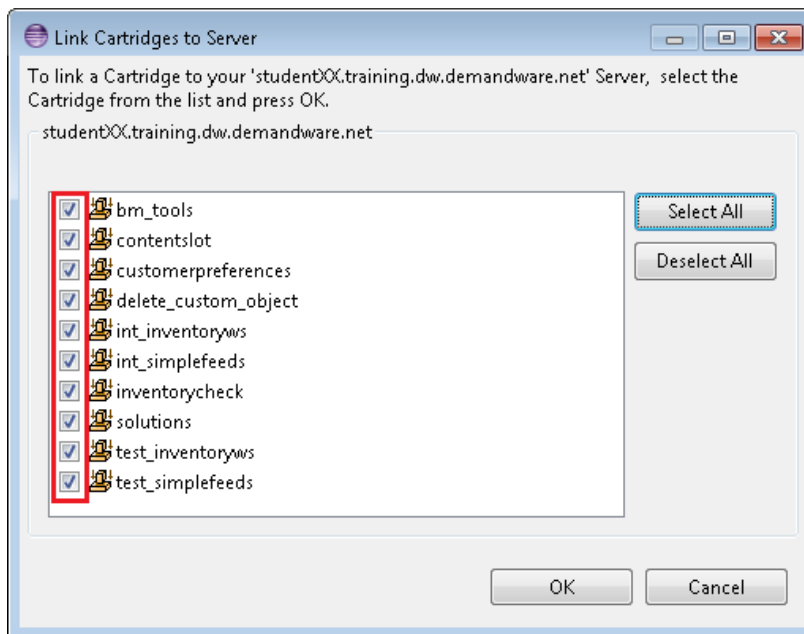
10. Click **'Finish'**.



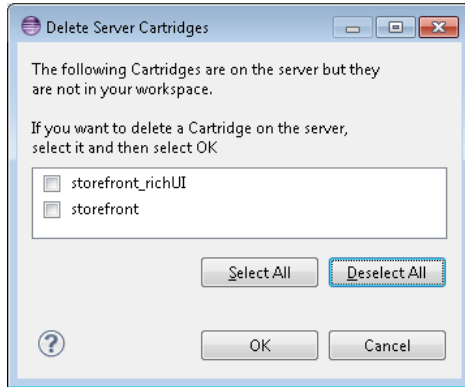
11. If you already have an active server connection in your workspace, the next dialog prompts you to link your imported projects with that server connection:



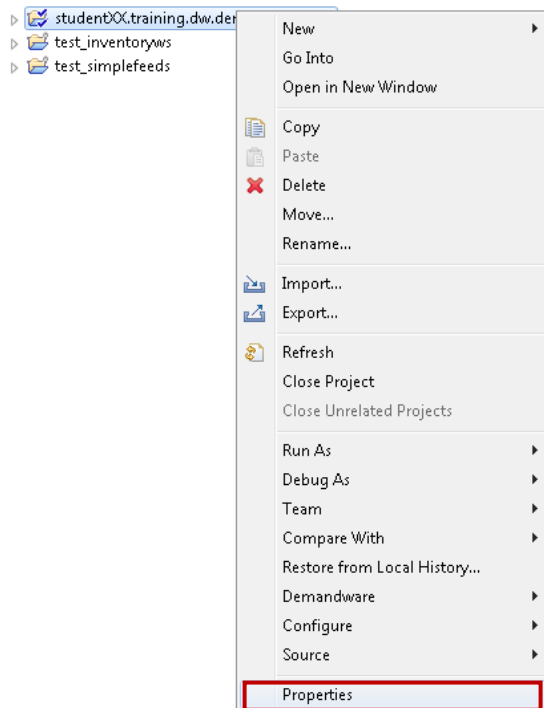
12. If you intend to upload the imported cartridges to the active server (why not?), click **Yes**. Otherwise the cartridges will reside in your workspace but will *not* be uploaded automatically when you make changes.
13. The next dialog allows you to select the specific cartridges you want uploaded to your server connection. Click **Select All** but unselect for now `bc_library` (**takes too long**) !



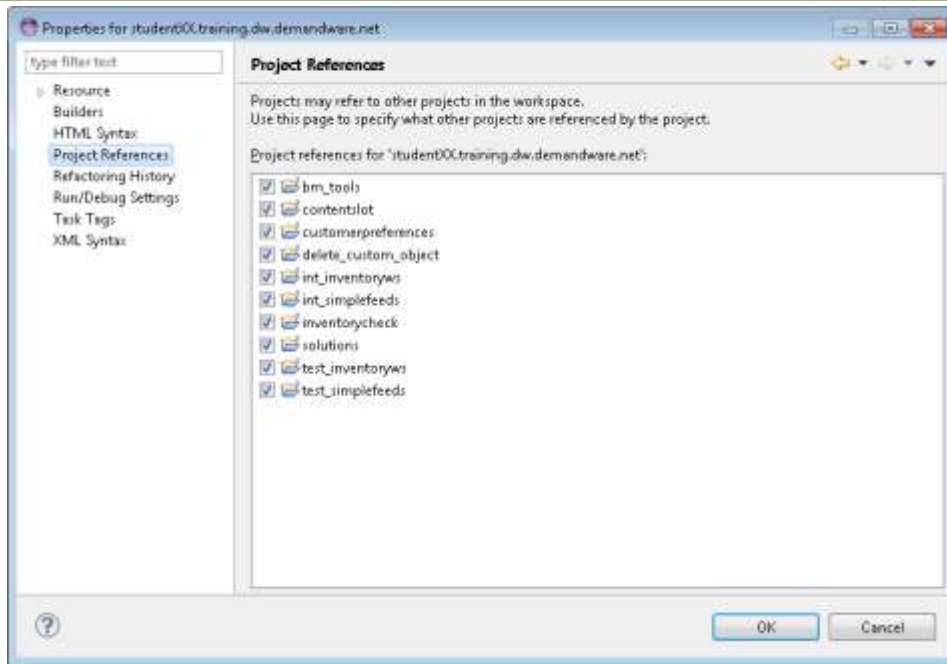
14. Click **OK** to upload the cartridges and finish the import process.
15. You might receive a dialog stating to delete projects on the server not matching the ones in your workspace.  
If you're the only one working on that instance e.g. it's your personal sandbox you might recognize the projects there.  
**If you are working on a collaborative instance consult first with your colleagues !**



16. If you import cartridges before you have an active server connection or somehow forgotten to link a cartridge to the server, do the following to ensure that cartridges will get uploaded correctly: **right-click the server connection** and select **Properties**:



17. Select **Project References** and then select every cartridge that you want uploaded to the server connection:



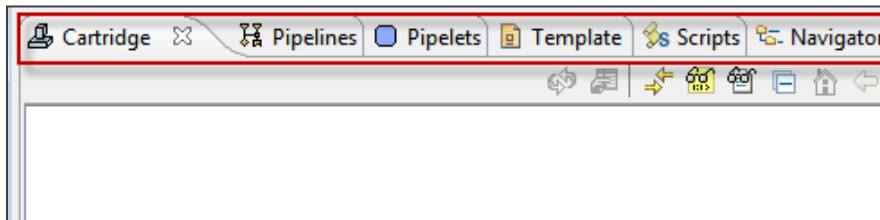
18. Click **OK**.

# Demandware Views

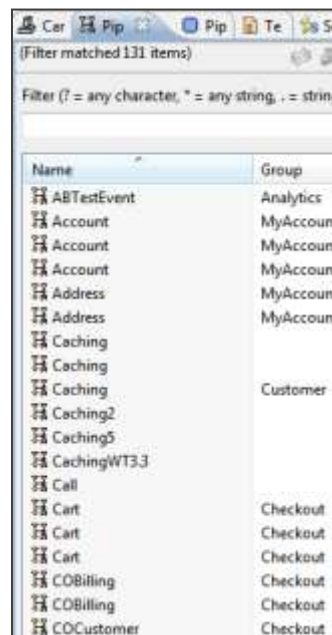


## Introduction

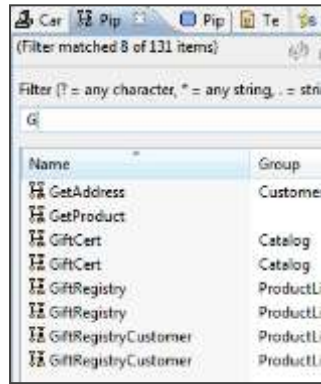
The Demandware UX Studio plug-in gives you access to specific Demandware programming files. Those files are sorted and given their own custom views in the application.



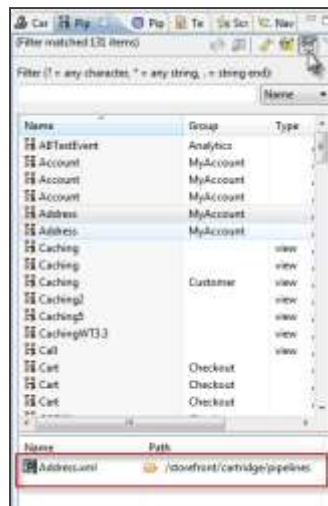
The primary files as you see listed above, are Cartridges, Pipelines, Pipelets, Templates, and Scripts (we will discuss each file type in greater detail later in this course.) For example, to view all Pipelines in a workspace, you only need to click on the 'Pipelines' tab.



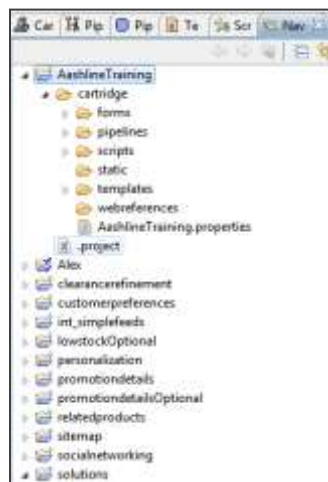
As you can see from the screenshot below, you will have the ability to filter the results by typing in the first few letters of the file name you are searching for:



To view the file path for a file, click on the **Show/Hide Resource Part** icon:



The **Navigation** tab will allow you to view all files in your workspace in a tree view structure. It also allows common tasks like copy/paste, file comparisons, etc.





### **Run the Demandware views activity.**

Navigate through the different views:

1. Cartridges
2. Pipelets
3. Templates
4. Scripts
5. Pipelines
6. Navigator

Do not go into deep detail of each file type. Just discuss what file types are in each view and how to search for a file in the search box.

# Searching for Text in Files



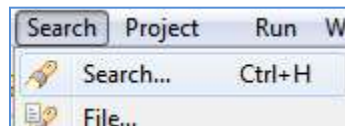
## Introduction

As you dive deeper into the Demandware platform, you will learn there are often numerous files that are used to display one single web page to a browser. So being able to quickly find specific code within a number of files is very important. To do this you will use the Search tool in UX Studio.

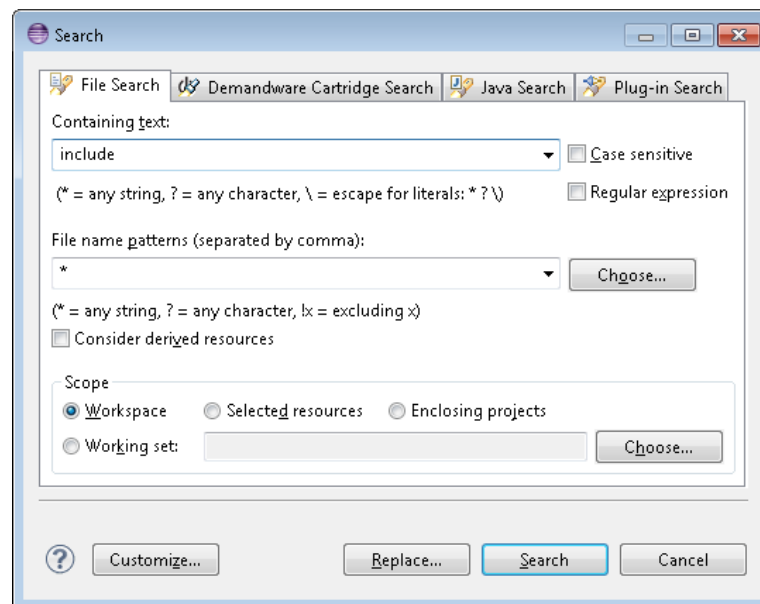
For those developers who have been using Eclipse for development, you may skip this topic since you most likely will have already utilized the search capabilities of the application.

## 2.4. Exercise: Search for text in files

1. Click the **Search** menu option in the main toolbar.

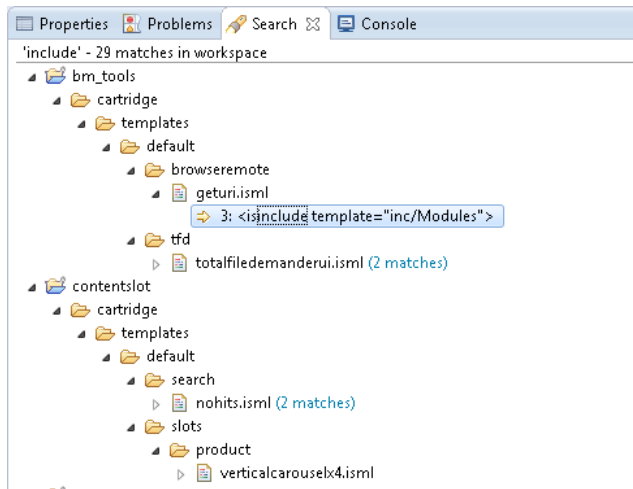


2. The search window will open.
3. Enter your text search criteria in the **Containing text** field:



4. Enter any file name patterns you want to filter by in the '**File name patterns**' field.

- When you are ready, click the **Search** button. Your results will be displayed in the **Search** box.



- To view a result, double-click on the file you want to view. It will open in the workspace area with the search term highlighted in the file.





# Review

Question	Answer
<p>1. To upload your code to a Demandware server</p> <ul style="list-style-type: none"> <li>a) Copy files to the root folder on the web server</li> <li>b) You will connect to a production server in a P.I.G.</li> <li>c) You will need to create a server connection in Studio</li> <li>d) Contact Demandware support to open a server connection for you</li> </ul>	
<p>2. You can find text in any workspace file by:</p> <ul style="list-style-type: none"> <li>a) Clicking on <b>File→Find Text</b></li> <li>b) Using the Search function in Studio by clicking on <b>Search→Search...</b></li> <li>c) Using the <i>Windows 7</i> search option</li> <li>d) Clicking on <b>Edit→Find</b></li> </ul>	




---

**Transition to Cartridges**

---

## 3.Cartridges



---

### Goal

The purpose and goal of this module is to familiarize you with Demandware cartridges.



---

### Time

1.5 Hours



---

### Overview

We will discuss what a cartridge is, cartridge directory structure, the cartridge path in Business Manager, and we will demonstrate how to create an empty cartridge as well as a new storefront cartridge.



---

### Materials Needed

N/A

---

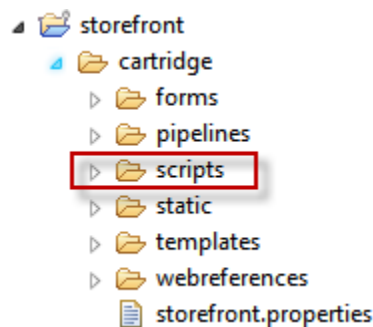
# What is a Cartridge?



## Introduction

A cartridge is a directory structure that is used as a flexible deployment mechanism for customized functionality. A cartridge can contain many different types of files: static files such as CSS, Javascript, and images, as well as WSDL files. There are also folders for Demandware specific files: 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, all Demandware script files must be stored in a **scripts** folder.



The **storefront.properties** file is generated by Studio when a new cartridge is created. This file is required.

## Cartridge Path

In order for a site to use a cartridge, the cartridge must be added to the cartridge path in Business Manager:



When a call is made to a file, the Demandware server will look for the file starting with the first cartridge listed in the cartridge path. For instance, if a call is made to

an ISML file called *product.isml* and that file is located in two cartridges that are both in the cartridge path, the Demandware server will use the first one it finds.

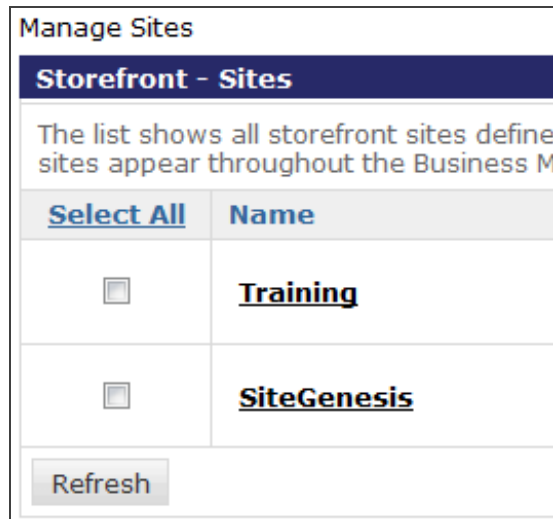
### 3.1. Exercise: Add Cartridge in Cartridge Path

To add a cartridge to the cartridge path, follow these steps:

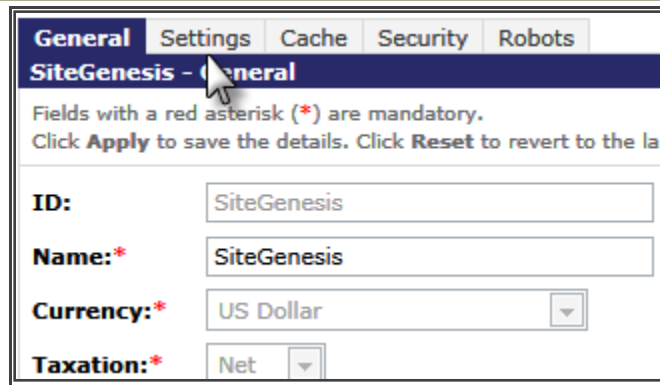
1. Log into Business Manager.
2. Click on **Administration**.
3. Click on **Sites**.



4. Click on **Manage Sites**
5. Click on the site **SiteGenesis** you wish to add a cartridge to.



6. Click on the **Settings** tab.



**General** Settings Cache Security Robots

**SiteGenesis - General**

Fields with a red asterisk (\*) are mandatory.  
Click **Apply** to save the details. Click **Reset** to revert to the la

**ID:** SiteGenesis

**Name:\*** SiteGenesis

**Currency:\*** US Dollar

**Taxation:\*** Net

- Here you can type in the name of the cartridges you wish to add. Additional cartridges can be added to the cartridge path by typing a **colon (:)** between cartridge names. In this case delete the existing path completely and add the following path. All names here are case sensitive and must match the names of your cartridges in eclipse. (Right now you may not have these in Eclipse, but soon you will). There should be no spaces in-between each item.

training:storefront\_richUI:storefront



**Instance Type: All**

**Cartridges:** storefront:promotiondetails

- Click the **Apply** button.

# Cartridge Types



## Introduction

There are three types of cartridges you can create in UX Studio:

1. Storefront cartridge
2. Demandware cartridge
3. Business Manager cartridge (not covered in this course)

Your business needs will determine what type of cartridge you will create in UX Studio. Every new customer will have at least one storefront cartridge.

## 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 Cartridges

If you need to build re-usable functionality that is specific to a site when there are multiple sites in a production instance, then you may need to create a new cartridge.

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

## Some best practices

- Keep an original SiteGenesis cartridge in your project just for comparison purposes, but don't make it part of the cartridge path
- Use another 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>`
- Any integration code should go in a `int_<site>` cartridge

### 3.2. Exercise: View WebDAV cartridge directory in Business Manager

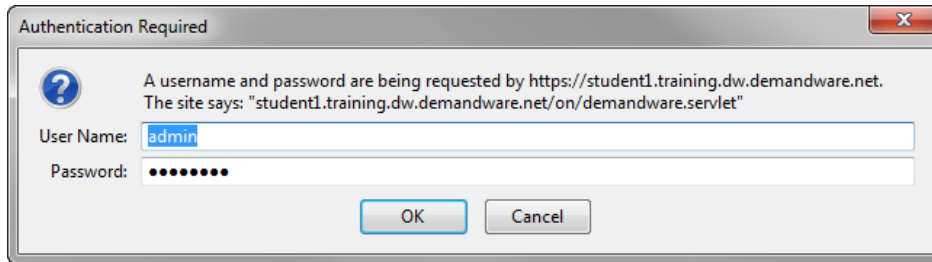
1. Log into the Business Manager instance you want to view cartridge contents (i.e.: staging instance.)
2. Click **Administration** → **Site Development** → **Development Setup**.



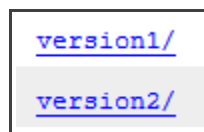
3. In the WebDAV Access section, click the link for Cartridges.



4. An Authentication window opens. Type the username/password that you used to log into Business Manager:



5. Click **OK**.
6. Click on the link for the code version you wish to view:



7. Click a version to see the uploaded cartridges.

## 3.3. Exercise: Create a new version on the server

1. Log into the Business Manager.
2. Click **Administration** → **Site Development** → **Code Deployment**.

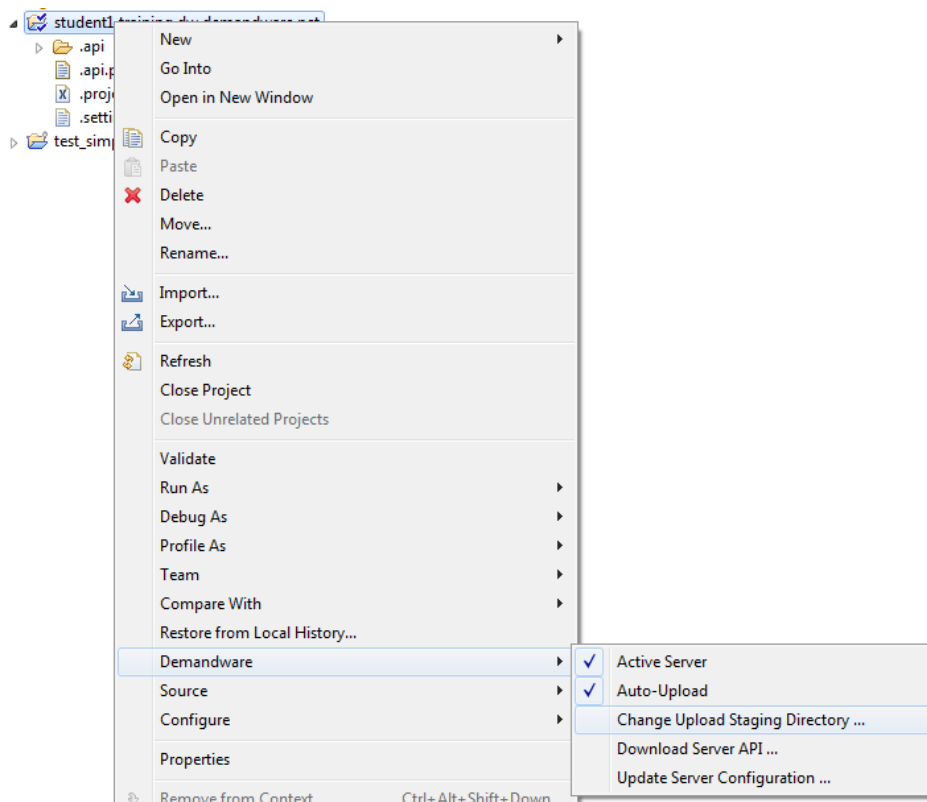
Select All	Active	Code Version	Compatibility Mode	Version Timestamp	Actions
<input type="checkbox"/>	✓	<a href="#">version1</a>	<a href="#">2.10.6</a> *	6/6/11 4:04:49 am	
<div>Refresh</div> <div>Rollback Delete Add</div>					

3. Click **Add** to create **version2**. Click **Apply**.
4. Click on the new version. The cartridges directory is empty:

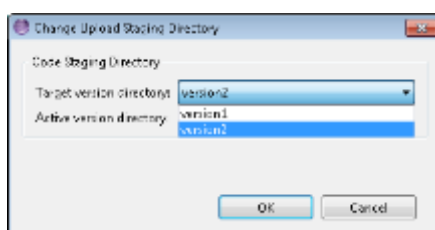
**WebDAV:** <https://student1.training.dw.demandware.net/on/demandware.servlet/webdav/Sites/Cartridges/version2>

**Cartridges:**

5. In Studio, open the **Demandware** → **Change Upload Staging Directory...** menu on the connection project:



6. Select **version2** from the dropdown:





## Instructor Guide

- Click **OK**. Wait for the cartridges to upload:

Upload Cartridges: (100%)

- Back in Business Manager, check the **version2** again:

**WebDAV:** <https://student1.training.dw.demandware.net/on/demandware.servlet/webdav/Sites/Cartridges/version2>

**Cartridges:** storefront, clearancerefinement, lowstockOptional, relatedproducts, test\_simplefeeds, solutions, int\_simplefeeds, personalization, lowstock, promotiondetailsOptional, socialnetworking, sitemap, customerpreferences, promotiondetails

- Now search for a file to see all versions of it (which is important when one cartridge overrides another):

**File Filter**  
   
Search is case-insensitive and supports standard wildcards '\*' and '?'.

Filename	Size	Last Modified
storefront/cartridge/forms/default		
product.xml	463 B	8/5/11 4:42:38 pm <a href="#">Download</a>
storefront/cartridge/pipelines		
Product.xml	24.26 KB	8/5/11 4:42:37 pm <a href="#">Download</a>
lowstockOptional/cartridge/pipelines		
Product.xml	25.92 KB	8/5/11 4:41:59 pm <a href="#">Download</a>

- Click the **Activate** button to make **version2** active:

**WebDAV:** <https://student1.training.dw.demandware.net/on/demandware.servlet/webdav/Sites/Cartridges/version2>

**Cartridges:** storefront, clearancerefinement, lowstockOptional, relatedproducts, test\_simplefeeds, solutions, int\_simplefeeds, personalization, lowstock, promotiondetailsOptional, socialnetworking, sitemap, customerpreferences, promotiondetails

From now on any new cartridges will be uploaded to version2, which is also the active version on the server.

# Creating a New Cartridge

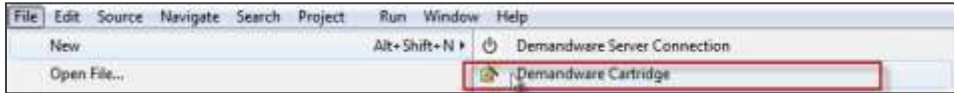


## Introduction

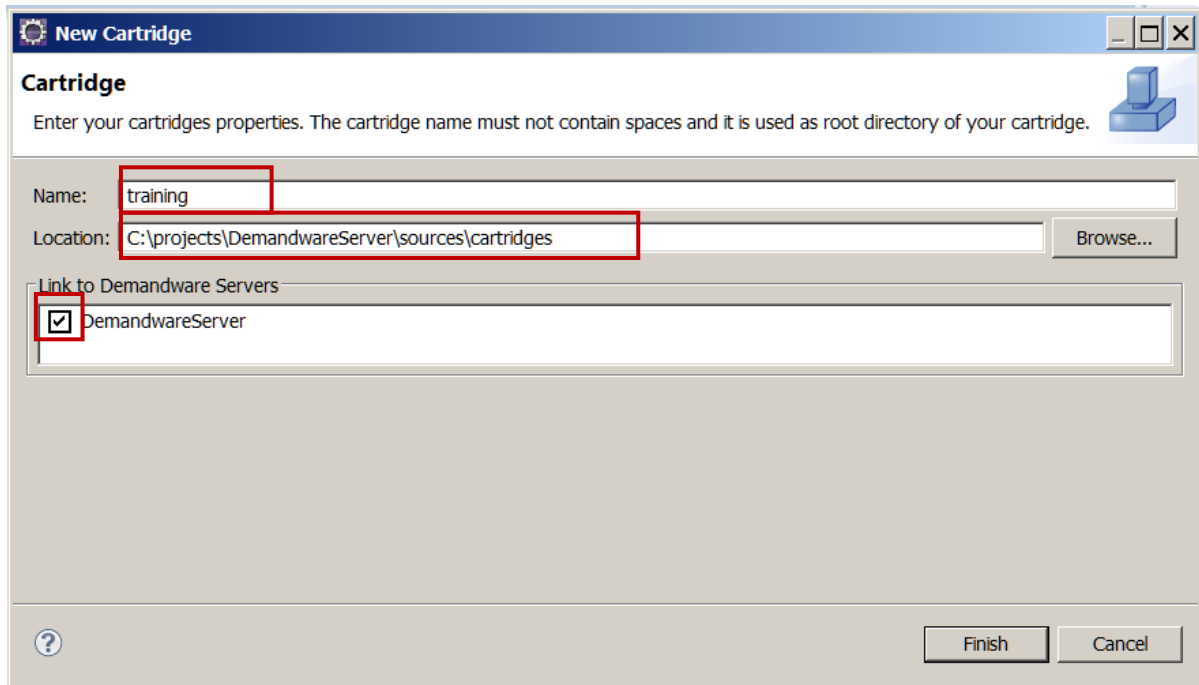
When you need to segregate code between sites, you may want to create a new empty cartridge. Creating a new empty cartridge will enable you to add only the code you need for a site or specific sites in an organization.

### 3.4. Exercise: Create a new empty cartridge

1. Log into your **workspace** in UX Studio.
2. From the main menu, click **File→New→Demandware Cartridge**.



3. The **New Cartridge** dialog appears: complete it as follows:



4. Name the cartridge as **training**. ('t' is not capital)
5. Examine the **training** cartridge. Notice that the directory structure was created but there are no files.

# Creating a Storefront Cartridge



## Introduction

When you create a new storefront cartridge in UX Studio, a copy of the **sitegenesis\_storefront\_richUI** and **sitegenesis\_storefront\_core** cartridges will be downloaded to your workspace and renamed with the name you specify. As mentioned before, these reference cartridges have all of the code needed for a SiteGenesis storefront to work in the Demandware platform.

As part of the architecture approach since the 12.6 deployment, Demandware will be separating the Site Genesis cartridge into two distinct cartridges based on the following functional guidelines:

### 1) Core Cartridge

This cartridge will embody the business layer and contain all of the “server” side components such as Pipelines and Demandware Scripts as well as a non-javascript version of the storefront. This cartridge will embody the simple presentation layer and contain ISML templates, Common CSS files, Forms and Resource files. There will be no use of JavaScript, AJAX, or any advanced UI elements inside of this cartridge. As a result the rendering output of Core is just a stream of HTML & CSS assets that represent the page styled to look as close as possible to Site Genesis wireframes today without the use of JavaScript. So at this level the site will be “usable”, and only partially visually esthetic. The goal of this cartridge is to provide a fundamental starting point for new site development under the circumstances when the client’s design differs significantly from Site Genesis’ wireframes. The benefit being that you can start with just the essentials and not have to “undo” or “separate” any pre-conceived Site Genesis UI design.

### 2) Rich UI Cartridge

This cartridge will hold the overloaded any specific CSS and advanced UI elements required to turn the Core Cartridge into the advanced look and feel of the Site Genesis storefront as we know it today. That being said, it is important to know that the Rich UI Cartridge does not contain ISML templates, but rather uses an initialization mechanism in order to effect changes to the rendering of Core templates in the client browser.

When you make changes to the new storefront cartridge, your changes will be uploaded to the Demandware server and can be viewed immediately as long as you:

- Set the cartridge to be uploaded to your workspace server
- Put the new cartridge in the cartridge path
- Have site caching disabled for the site

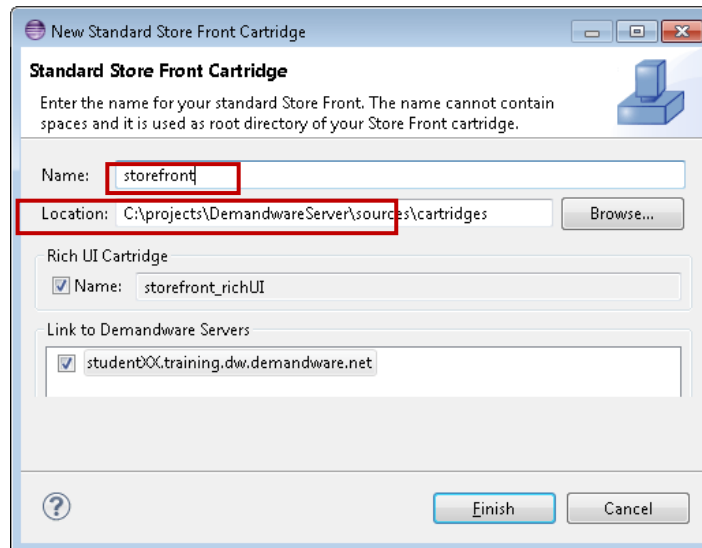
### 3.5. Exercise: Create a New Storefront Cartridge.

To create a new storefront cartridge, follow these steps:

1. From the main menu in UX Studio, click **File**→**New**→**Demandware Storefront Cartridge**.



2. Complete the **New Storefront Cartridge** dialog as follows:



3. Click **Finish**.

## Review & Lab

Question	True	False
A Demandware storefront cartridge is an empty cartridge with empty sub-folders.		
You should create a new cartridge when you need to build generic functionality that can be reused in many sites.		
You can view a list of all files in a cartridge located on a Demandware server from Business Manager		

## 4. Pipelines



---

### Goal

The purpose and goal of this module is to familiarize you with Demandware pipelines and pipeline nodes.



---

### Time

2.5 Hours



---

### Overview

We will discuss what a pipeline is, the pipeline dictionary, as well as the different pipeline elements in a pipeline. We will also demonstrate how to use the following elements: start, interaction, call, jump, and pipelets.



---

### Materials Needed

Solutions Cartridge

---

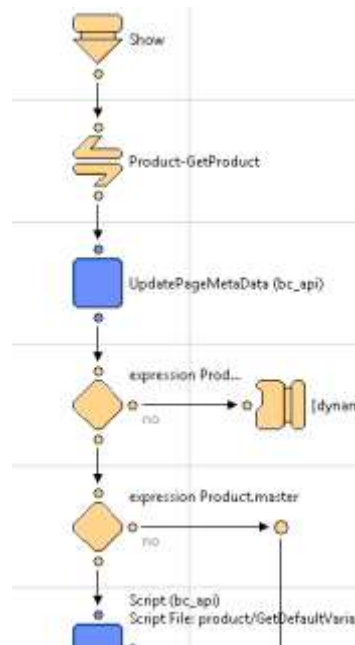


# Pipeline Elements



## 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. Below is 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 on your PC and on the server. Pipelines are defined and stored within the context of a cartridge.

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

There are fifteen different pipeline elements available to you for developing a pipeline. Each node has a specific function. The table below lists all the pipeline elements and their associated function.

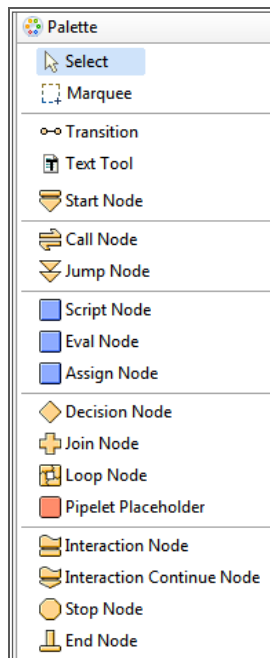
Element	Icon	Description
Start Node		Begins the logical branch of a pipeline.
Interaction Node		Used when a request requires a page as a response.
Transition Node		Defines a path along a pipeline between pipeline nodes.
Call Node		Invokes a specified sub-pipeline. After the sub-pipeline execution the workflow returns to the calling pipeline
End Node		Used to terminate a sub-pipeline, returns to the calling pipeline.
Jump Node		Used when the pipeline forwards the request to another pipeline.
Join Node		Provides a convergence point for multiple branches in workflow.
Interaction Continue Node		Processes a template based on user action via a browser.
Script Node		Used to execute Demandware scripts.
Eval Node		Evaluates an expression.
Assign Node		Used to assign values to new or existing Pipeline Dictionary entries, using up to 10 configured pairs of dictionary-input and dictionary-output values
Stop Node		Used to terminate a sub-pipeline and calling pipelines, stops execution immediately. Used in pipelines that executes a batch job.
Loop Node		Used to loop through an iterator.
Pipelet Placeholder		Placeholder for a script node.
Decision Node		Evaluates a condition and navigates to a different branch in the pipeline.

# Creating a Pipeline

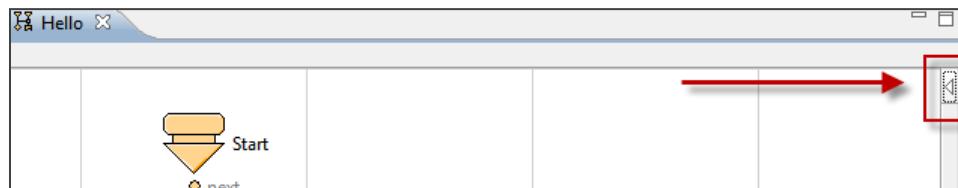


## Introduction

In order to create a new pipeline, you will need at least one Start node and one Interaction Node. When creating a new pipeline in Studio, the pipeline palette will be available in the Pipeline Editor, as shown below.



If you do not see the palette when you create a new pipeline, be sure to click the button on the upper-right corner of the editor to open it up.



### Start Nodes

A pipeline may have multiple Start nodes but each node must have a unique name. Every Start node is the beginning of a different logical branch within the pipeline.

Configuration properties include:

1. Name: Used in pipeline calls
2. Call mode: Specifies the accessibility of the pipeline from a browser.
  - a. Public: Can be called from the browser or from another pipeline
  - b. Private: Can be called from another pipeline via Call or Jump Nodes
3. Secure Connection Required:
  - a. False: Pipeline can be invoked with HTTP and HTTPS protocols
  - b. True: Start node can only be accessed via secure (HTTPS) protocol.

### Interaction Node

This node specifies the template to display in the browser. If Dynamic Template is:

1. true: Template Expression must be a variable containing a template name. The template to be called by an interaction node is not always hard-coded in the node. Instead, the name can be determined dynamically during runtime from the Pipeline Dictionary.
2. false: The template expression contains a path to the template under the `templates/default` folder

### Transition Node

The transition node creates a transition between two nodes. You can easily create a transition between two nodes by clicking and dragging your mouse between two nodes in a pipeline.



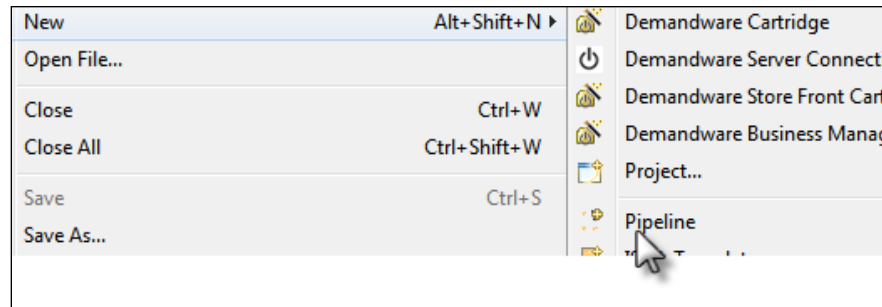
Run the **Creating a Simple Pipeline** activity.

Walk the students through creating the 'Hello' pipeline.



To create a simple pipeline using a Start node and Interaction node, follow these steps:

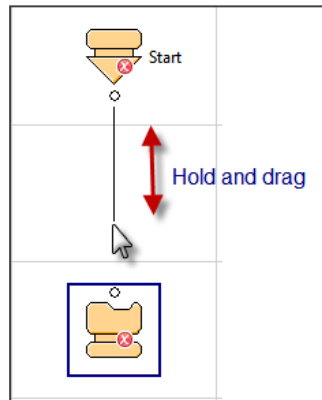
1. From UX Studio, click **File→New→Pipeline**



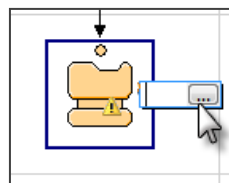
2. In the Create Pipeline window, give your pipeline a name. Be sure to make the name meaningful to the business process the pipeline will be fulfilling.
3. Click **Finish**.
4. From the Palette, click and drag a Start node to the work area.



5. Click and drag an Interaction Node to the work area.
6. Hold your mouse down over the white dot at the bottom of the Start Node. Drag your mouse over to the white dot at the top of the Interaction Node. Let go of your mouse. A transition node will connect the two elements.



7. Click the Interaction node twice (not double-click) to get the ellipsis button next to the node.



8. Click the ellipsis button to select the template you wish to display with the Interaction node.
9. Select the template then click **Ok**.
10. Save the pipeline: **CTRL+S**.

## Creating a Pipeline, continued



### 4.1. Exercise: Create a Pipeline

1. In Studio, select **File** ⇒ **New** ⇒ **Pipeline** in **training** cartridge.
2. Name the pipeline **Hello**, do not specify a group, and keep **View** as the pipeline type.
3. Using the palette, drag a **start node** onto the pipeline editor. The name defaults to *Start*.
4. Drag an **Interaction Node** below the start node, and connect them with a **Transition**.
5. Create a template **hello.isml** that renders a simple HTML page with a “Hello World!” greeting:

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    Hello World!
  </body>
</html>
```

6. Specify template name **hello.isml** in the properties of the Interaction Node in the pipeline.
7. Double-click the interaction node to verify that it opens the hello.isml template.
8. Save both the pipeline and the template.

# Executing a Pipeline



## Introduction

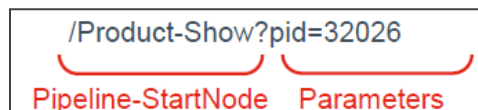
Pipelines can be executed from the browser via an HTTP(S) request or via Call or Jump Nodes. If the pipeline Start node is set as 'Private' it can only be called via a Call or Jump node.

Calling a pipeline via a HTTP request requires the pipeline name and start node at the end of the storefront URL:

`http://instance.realm.client.demandware.net/on/demandware.store/Sites-YourSite-Site/default`



You can also pass parameters via HTTP requests using the following syntax:



## Run the Pipeline Execution activity.

Walk the students through executing a pipeline through a browser.



**To execute a public pipeline from the storefront, follow these steps:**

1. Open your storefront in a browser window.
2. At the end of the url and after the `default/` directory, type in the name of your pipeline and start node using the following syntax:

`/Sites-SiteGenesis-Site/default/Hello-Start`

3. To pass parameters, add a query string after the pipeline invocation:

`/Sites-SiteGenesis-Site/default/Product-Show?pid=ETOTE`





### 4.2. Exercise: Executing a Pipeline Exercise

1. Test your pipeline in the storefront:

<http://student1.training.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/Hello-Start>

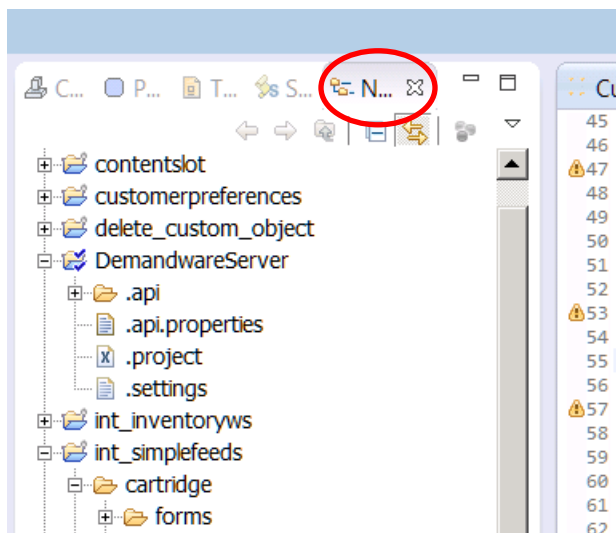
2. Close the Storefront Toolkit on the upper-left corner of the page so that you can view the output.
3. Bookmark this URL so you can use it for future pipelines invocations during this class.



### 4.3. Exercise: Checklist when first pipeline does not work

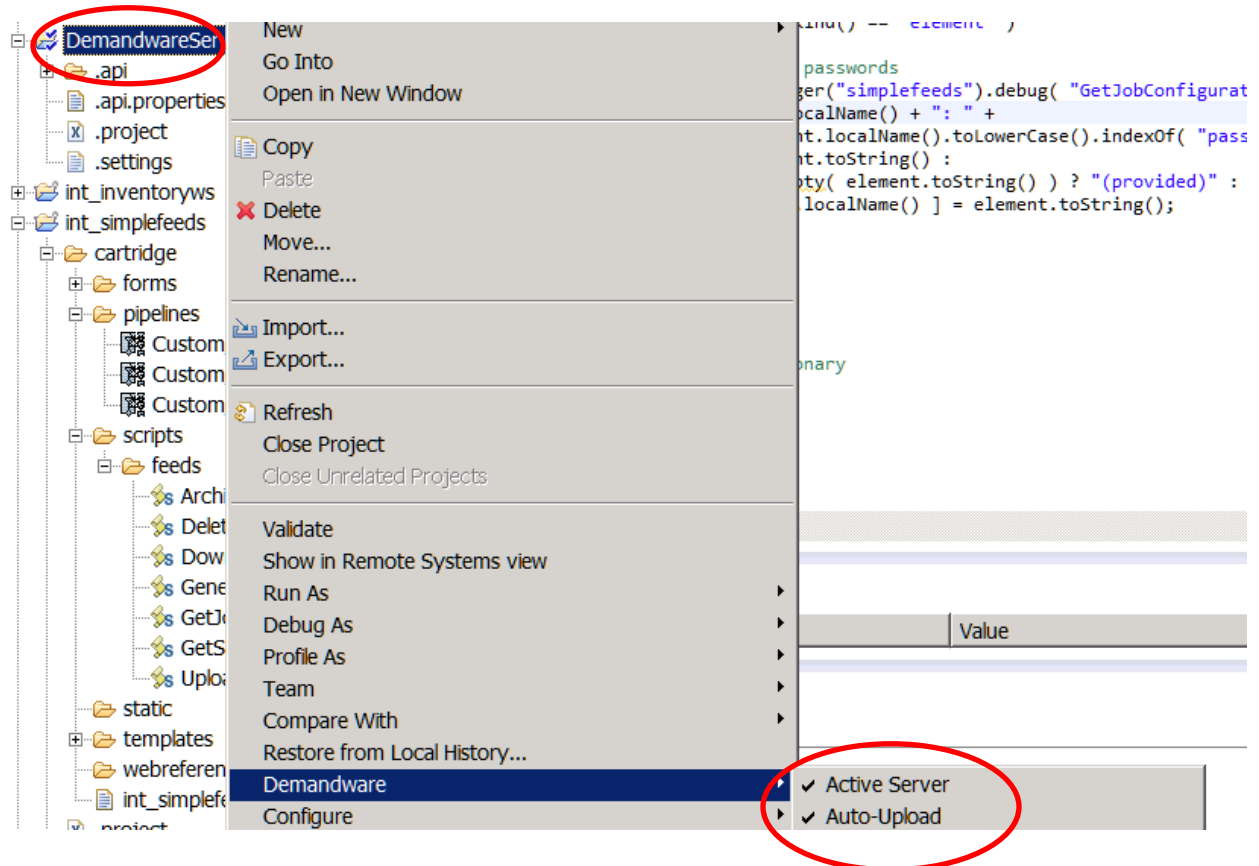
When your first pipeline does not work, it is a good idea first to go through the check list to see if all settings are ok. Even if it does work, it is good to go through the check list to see the settings.

1. Make sure that you are in Navigator view before the next steps in this exercise.

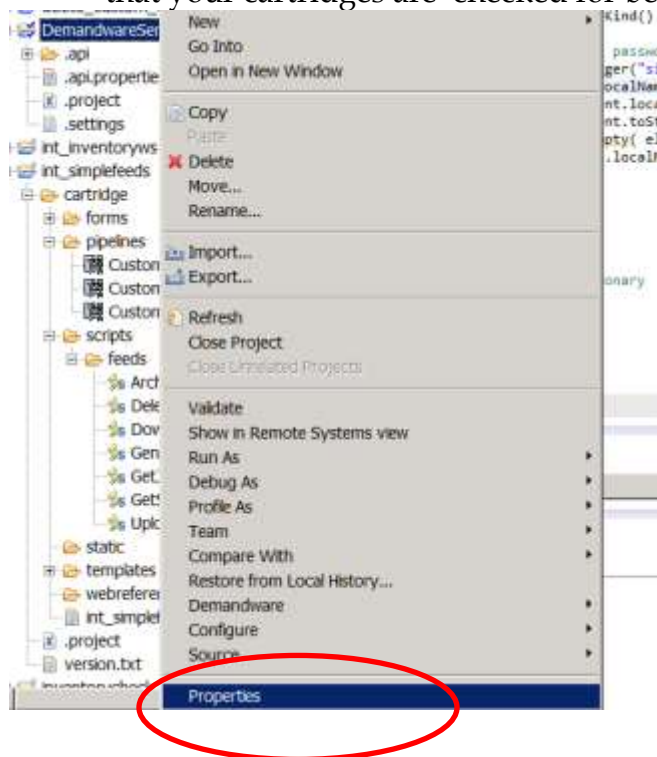


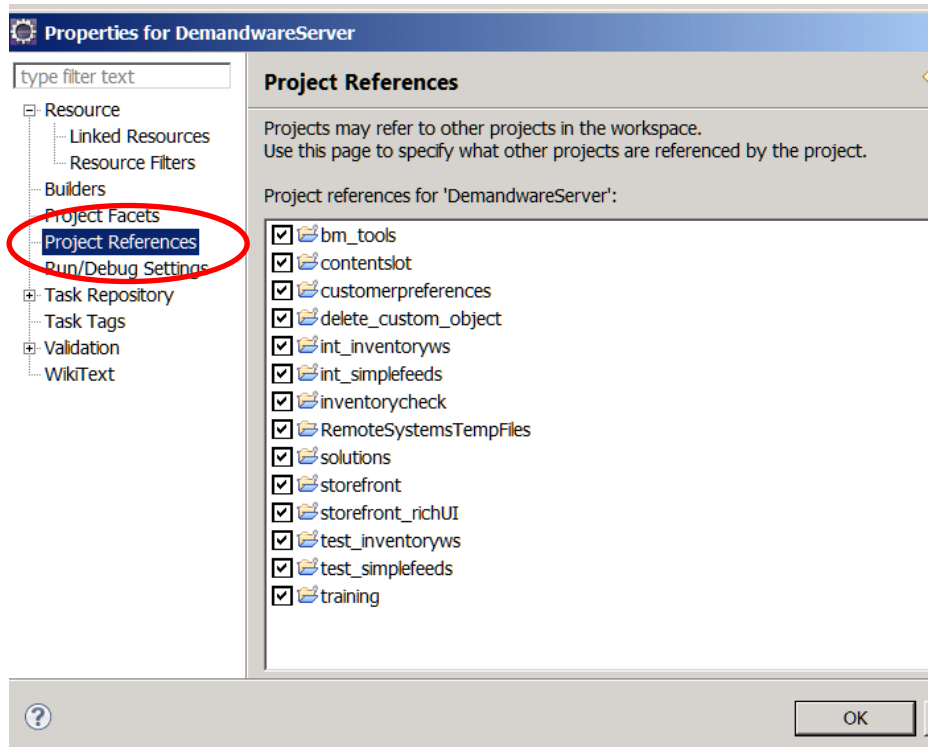
2. Right-click **DemandwareServer** and then hover your mouse over **Demandware**. You should be able to see Active Server and Auto-Upload checked. If you do not, then check them.

3.

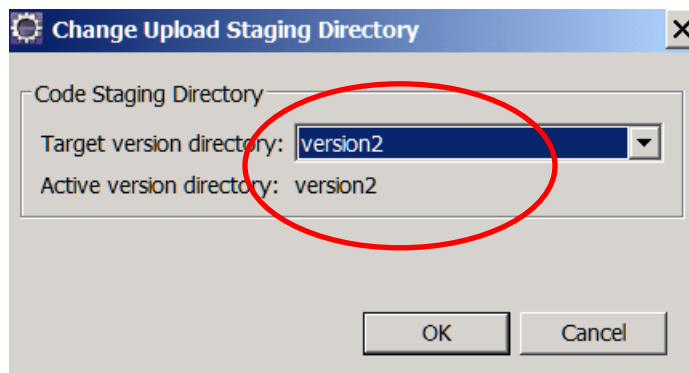


4. Navigate to **DemandwareServer > Properties > Project references**. Check that your cartridges are checked for being uploaded to the server.

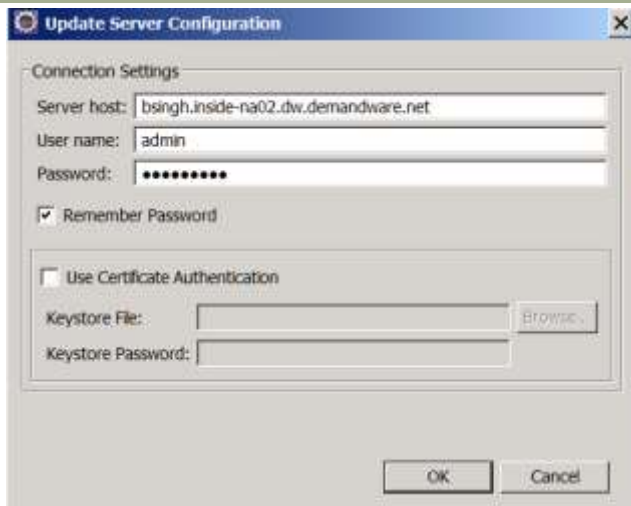




5. Navigate to **DemandwareServer > Demandware > Change Upload Staging Directory** and click there. Make sure that the Target version directory and Active version directory matches.



6. Navigate to **DemandwareServer > Demandware > Update Server Configuration** and click there. Make sure that the configuration is correct.



5. Check your cartridge path. Navigate in your browser to Business Manager. Further Navigate to **Administration > Sites > Manage Sites > Site Genesis > Settings tab**. Check your Cartridge path. It should have the exact same name as the cartridges as in eclipse. There should be no spaces in the path. Make sure that there are no semicolons in place of colons. Also be aware that the names are case sensitive.

[Sites > Manage Sites > Site Genesis - Settings](#)

General	Settings	Cache	Security	Robots
<b>Site Genesis - Settings</b>				
Click <b>Apply</b> to save the details. Click <b>Reset</b> to revert to the last saved state.				
<b>Instance Type:</b> <span>Sandbox / Development</span>				
<b>Deprecated!</b> The preferred way of configuring HTTP and HTTPS host names is by using new features of the site aliases configuration in this section will be used if no hostnames are defined by aliases configuration and intended only to support older configuration				
<b>HTTP Hostname:</b>				
<b>HTTPS Hostname:</b>				
<b>Instance Type: All</b>				
<b>Cartridges:</b>		training:storefront_richUI:storefront		

7. Navigate to **Administration > Sites > Manage Sites > Site Genesis > Cache Tab**. Check if **Time to live** is 0 and **Enable Page Caching** is disabled.

## Instructor Guide

General Settings **Cache** Security Robots

### Site Genesis - Cache

Web content is cached by the web server to improve response times.  
The system uses 2 caches for different kinds of content. These are:

1. **Static Content Cache**  
This caches images, icons, CSS-stylesheets and other non-dynamic assets.
2. **Page Cache**  
This caches HTML pages generated from ISML templates when <iscach>

Both caches can be configured and invalidated separately.

**Instance Type:** Sandbox / Development

#### Static Content and Page Caches

The 'Time to live (TTL) of static content' field defines the time span (in seconds) default is 86,400 seconds, or 24 hours. To disable the caching of static files set the value to 0.

Click 'Invalidate' to start invalidation of the static content and page cache (will clear the cache).

**Time to live (TTL) of static content:**  **Last invalidated:** 11/12/12 8:23:02 am

#### Page Cache Only

Check the 'Enable page caching' box to allow the web server to cache selected storefront pages.

Click 'Invalidate' to start invalidation of cached dynamic storefront pages (will clear the cache).

**Enable page caching:** ☐ **Last invalidated:** 11/12/12 8:23:02 am

8. If you have forgotten before, index your site now.  
Navigate to **Site > Site Genesis > Search > Search Indexes**.  
Check all checkboxes and click on Reindex.

Search

- Online Marketing
- Customers
- Custom Objects
- Ordering
- Analytics
- Site URLs
- Site Preferences

Administration

- Replication
- Organization
- Sites
- Site Development
- Global Preferences
- Operations

All index updates are performed asynchronously in the background.

This page also provides access to the settings for the search indexes. The checkboxes can be used to enable or disable the indexes.

\* Instances of the type Sandbox/Cloudbox are not supported. If ran manually, the Search Index Update Job will run successfully.

The index rebuild schedule settings for this site can be configured in the Site Settings page.

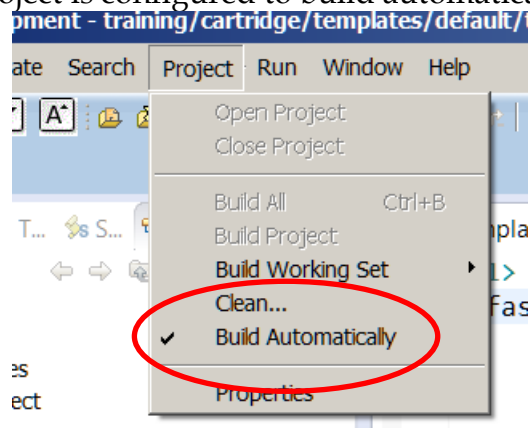
Index Type / Locale
<input checked="" type="checkbox"/> Product / Spelling Indexes
<input checked="" type="checkbox"/> Product Index
<input checked="" type="checkbox"/> Default
<input checked="" type="checkbox"/> Chinese
<input checked="" type="checkbox"/> French

☒ **Activedata Index**

☒ **Non-Localized**

That is all. You have gone through the checklist and should be able to run your pipelines. Just remember to save your project before executing the pipeline and type the url correctly.

9. Check if your project is configured to build automatically in eclipse as below.



# Troubleshooting with the Request Log Tool



## Introduction

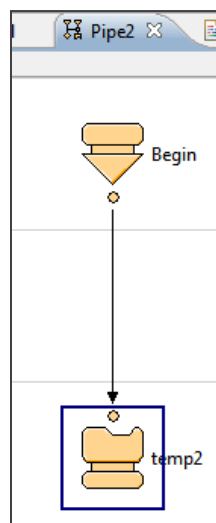
The first tool you can use for troubleshooting error messages on your storefront is the Request Log tool which is part of the Storefront Toolkit that is available in all instances except for Production.

The Request Log tool displays the log for the last request and any request prior during the current session. You can see debug messages as well as error messages.

The most common errors you will make as a new Demandware developer are typographical. In the example below, a call was made to a pipeline named 'Pipe2' with a start node called Start.

```
are.store/Sites-SiteGenesis-Site/default/Pipe2-Start
```

However, the actual name of the start node is 'Begin'.



When looking at the request log, you can see the error is "Start node not found (Start)"

```
4 [2011-03-17 14:42:26.822 GMT] ERROR system.core - Sites-SiteGenesis-Site core Storefront Wo21AzddVOKxbHrfQ-oJVvF
E2_ASU2CHdK8VcYK-0-00 "Exception occurred during request processing: Start node not found (Start) for pipeline (Pipe2)
----- RequestID: E2_ASU2CHdK8VcYK-0-00 SessionType: STOREFRONT SessionID: Wo21AzddVOKxbHrfQ-oJVvF
could not be determined ServerName: domU-12-31-39-10-56-32.compute-1.internal ServerPort: 10052 Request Information -
/Beehive/Sites-SiteGenesis-Site/default/Pipe2-Start Method: GET PathInfo: /Sites-SiteGenesis-Site/default/Pipe2-Start Remote
```



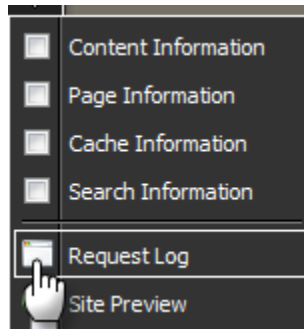
### 4.4. Exercise: Run the Request Log

To access the Request Log tool, follow these steps:

1. From Business Manager, open your sandbox storefront by clicking the **storefront** link
2. Click on the Storefront Toolkit drop-down button that appears on the upper-left hand corner of the site



3. Click the **Request Log** checkbox.



4. The Request Log window will open. You may see a login screen instead of the actual request log. If so, type in your Business Manager login credentials, close the window, and repeat steps 2-3 here.



#### Request Log Exercise

1. Open your sandbox storefront.
2. Invoke a `Test-Start` pipeline (which has not been created).
3. Open the **Request Log** to view the error message.

## Call Nodes & End Nodes



### Introduction

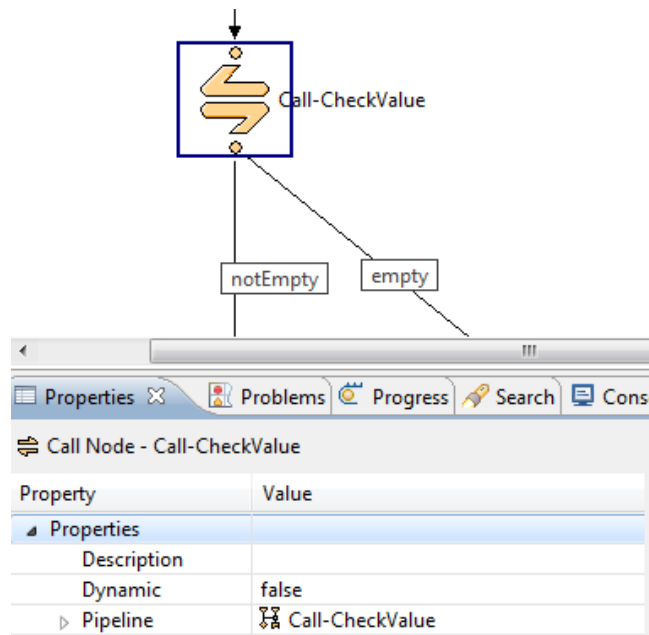
Call nodes and End nodes work together to process specific functionality in a pipeline.

### Call Nodes

A Call node invokes a specified sub-pipeline. A sub-pipeline is a pipeline that is designed for reusability and typically is defined as **private**, meaning that it cannot be invoked from a URL.

After the sub-pipeline execution the workflow returns to the calling pipeline by means of an End node. It basically behaves like a function call where the function might return one or multiple values.

A Call node requires only a Pipeline-Start node to invoke. This information can be provided as a fixed configuration value or from a pipeline dictionary key. Pipeline dictionary will be covered later in this module.

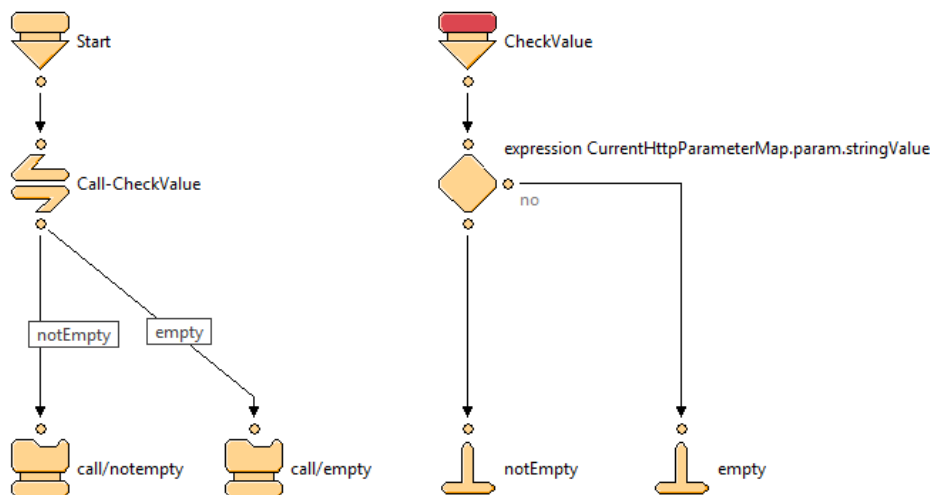


### End Nodes

An End node finishes the execution of the called sub- pipeline and returns a value equal to the End node name. This name must be unique within the sub-pipeline and it may be used by the calling pipeline to control flow after the call.

After the call, a transition from the Call node with the same name as the returned value is followed. In the picture below, if the CheckValue sub-pipeline returns a **notEmpty** value, then that transition is followed on the Start pipeline.

The figure below is an example of a Call node being used to invoke another pipeline. At the end of the execution of the called pipeline, which is using a Decision node to check whether there is a value called **param** being passed in a URL string, the End nodes return control back to the Call node.





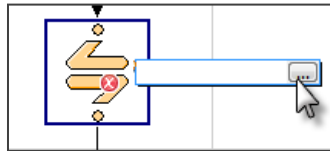
### Run the Call Node activity.

Create a pipeline in front of the students that utilizes a Call node and End nodes. You can use the example in the Process Steps.

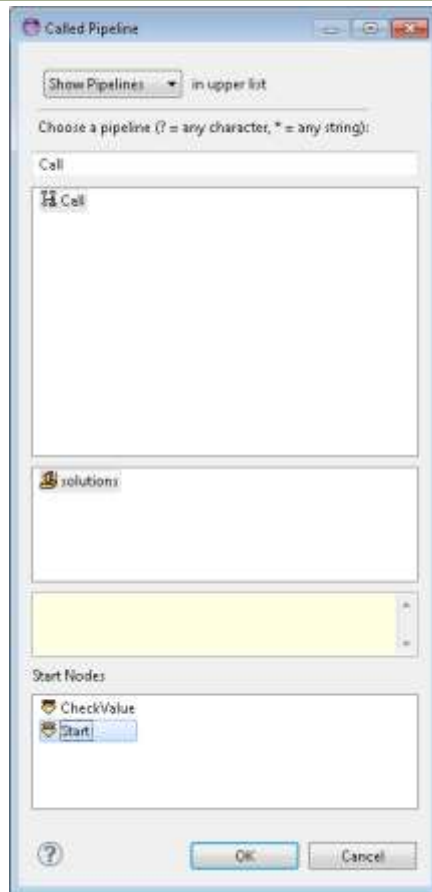


### To use a call node, follow these steps:

1. In Studio, open the pipeline you wish to add the **Call** node to.
2. Select the **Call** node from the palette and drag it over the transition node where you want it. Be sure the transition node turns red before you let go of your mouse.
3. Click the ellipsis next to the **Call** node.



4. Select the pipeline and start node you want to call using the **Call** node.
5. From the **Called Pipeline** window, select **Show Pipelines** from the top drop-down. Then click in the first top white box and type the name of the pipeline you are looking for. The lower boxes will populate with available pipelines and Start nodes.



6. Click **Ok**.
7. You will need to add Interaction nodes that will display the proper isml template depending on which value is returned by the called pipeline.
8. Name the **transition** nodes from the **Call** node according to the values returned by the **End** nodes in the called pipeline.
9. Save your files then test your work by calling the Pipeline-Start node from your storefront.

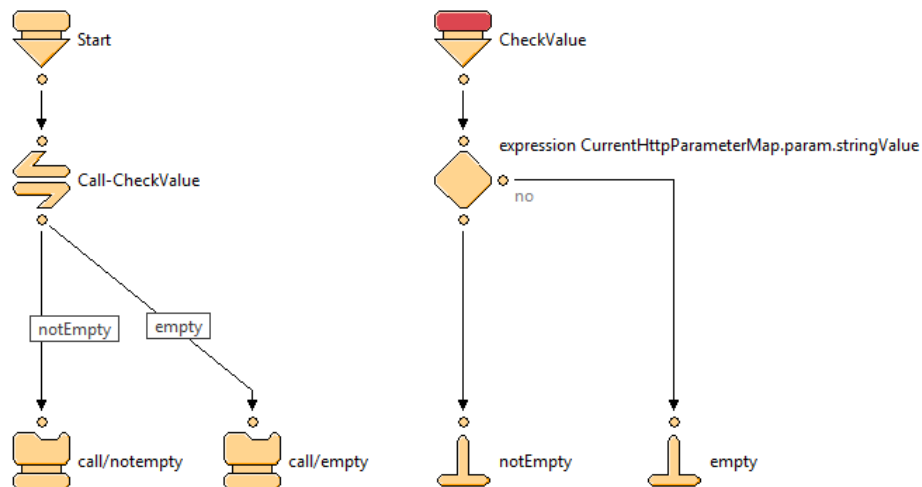


#### 4.5. Exercise: Using a Call Node in a Pipeline

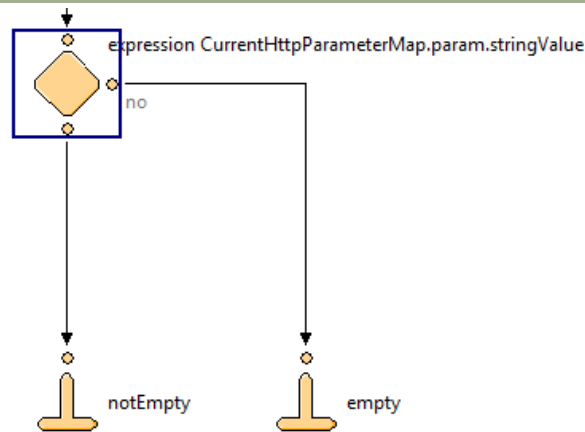
1. Create a pipeline named `Call` to execute a simple call to a sub-pipeline that will determine whether a value called `param` is present in the pipeline URL query string:

`/Sites-SiteGenesis-Site/default/Call-Start?param=1234`

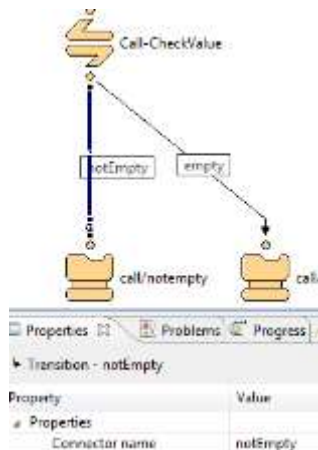
2. If the key is present, the execution of the pipeline will return a message stating so; otherwise the message states the value is missing.
3. Your pipelines should look something like this:



4. Build the `CheckValue` sub-pipeline first:
  - a. Make the call mode **private**: it is a best practice to make sub-pipelines private so that they can only be called from other pipelines.
  - b. Drag a **Decision** node under the **Start** node and set these values in the properties window:
    - i. Comparison operator = `expression`
    - ii. Decision Key = `CurrentHttpParameterMap.param.stringValue`
  - c. Add an **End** node under the Decision node. Name it `notEmpty`
  - d. On the other exit of the **Decision** node add an `empty` node:



5. Create the calling pipeline Call-Start.
6. Add a **Call** node that invokes the Call-CheckValue sub-pipeline.
7. Create two **Interaction** nodes below the **Call** node.
8. Drag a **transition** from the **Call** node to the first **Interaction** node: change its connector name to `notEmpty`:



9. Name the other Transition node `empty`.
10. Create two templates that display different messages and assign them to the Interaction nodes. The successful path should have an ISML code as below:

```
Got the parameter ${pdict.CurrentHttpParameterMap.param.value}
```

The path which does not have parameter should have ISML code as below:

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

11. Save your work.
12. Invoke the Call-Start pipeline from your SiteGenesis site using the URL query string `param=1234`.

<http://student1.training.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/Call-Start?param=1234>

# Jump Nodes



## Introduction

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.

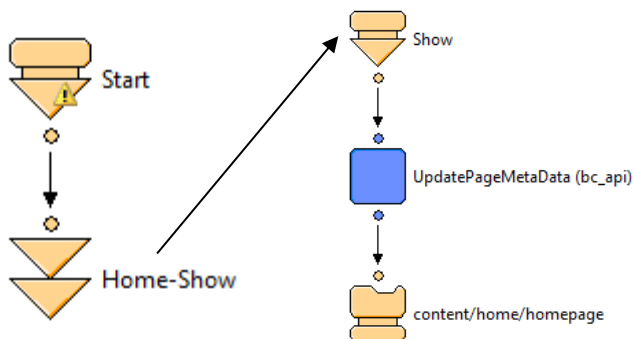
A jump node requires:

- The name of the pipeline to be jumped to and
- The name of the pipeline start node to be used

This information can be provided:

- Either as a fixed configuration value OR
- From a pipeline dictionary key (covered later)

An example of using Jump nodes is the `Default` pipeline. This is a special pipeline that is called by the system if no pipeline name was provided in the URL. In SiteGenesis the `Default-Start` pipeline jumps to the `Home-Show` pipeline, which shows the homepage.



## Run the Jump Node activity.

Demonstrate how to use a **Jump** node: show the `Default-Start` pipeline and the jump to the homepage.



# The Pipeline Dictionary



## Introduction

The pipeline dictionary or `pdict` is the main data container for each pipeline execution. It is created and initialized when a pipeline is invoked and remains in memory as long as the pipeline executes.

The structure of the pipeline dictionary is a hash table with key/value pairs.

The default keys in the `pdict` are:

- `CurrentDomain`
- `CurrentOrganization`
- `CurrentPageMetadata`
- `CurrentSession`
- `CurrentRequest`
- `CurrentUser`
- `CurrentHttpParameterMap`
- `CurrentForms`
- `CurrentCustomer`
- `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.

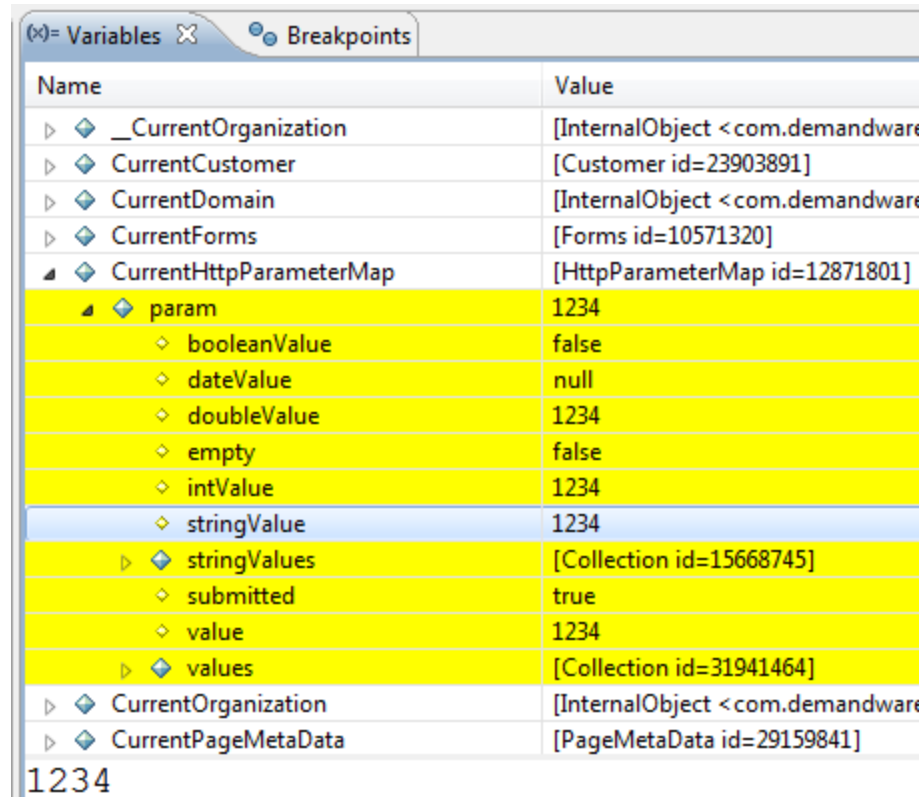
To view the values stored in the pipeline dictionary at run-time, run a pipeline from the storefront while in a debug session (covered next).

## Passing Parameters

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

[/default/Call-Start?param=1234](#)

The parameters will get added to the `CurrentHttpParameterMap` object inside the `pdict`. You can see the values stored in the pipeline dictionary when you run the pipeline debugger (covered next):



Name	Value
▶ ◆ <code>_CurrentOrganization</code>	[InternalObject < com.demandware
▶ ◆ <code>CurrentCustomer</code>	[Customer id=23903891]
▶ ◆ <code>CurrentDomain</code>	[InternalObject < com.demandware
▶ ◆ <code>CurrentForms</code>	[Forms id=10571320]
▶ ◆ <code>CurrentHttpParameterMap</code>	[HttpParameterMap id=12871801]
▶ ◆ <code>param</code>	1234
◆ <code>booleanValue</code>	false
◆ <code>dateValue</code>	null
◆ <code>doubleValue</code>	1234
◆ <code>empty</code>	false
◆ <code>intValue</code>	1234
◆ <code>stringValue</code>	1234
▶ ◆ <code>stringValues</code>	[Collection id=15668745]
◆ <code>submitted</code>	true
◆ <code>value</code>	1234
▶ ◆ <code>values</code>	[Collection id=31941464]
▶ ◆ <code>CurrentOrganization</code>	[InternalObject < com.demandware
▶ ◆ <code>CurrentPageMetaData</code>	[PageMetaData id=29159841]

1234

As you can see, the `CurrentHttpParameterMap` is an object of type `HttpParameterMap`. It contains (on this invocation) a single key called `param` which in turn contains multiple values. One of them is the `stringValue`, which contains the string '1234'. You could also use the `intValue` if you wanted to use the integer value 1234 on a calculation.

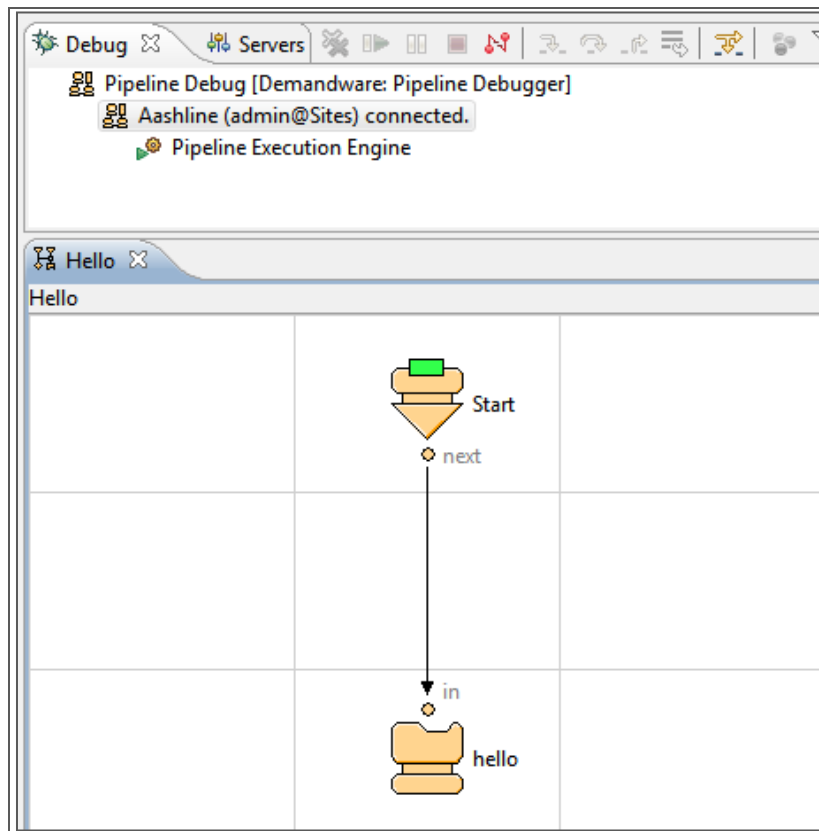
For more information on the different classes mentioned here, please consult the Demandware Script & Pipeline API in the Help menu. We will cover the Script API in more detail in later modules.

# Troubleshooting with the Pipeline Debugger



## Introduction

When you are testing your pipelines in the storefront and receive an error on execution, you can use the Request Log tool as well as the Studio Pipeline Debugger to help you troubleshoot your pipeline.

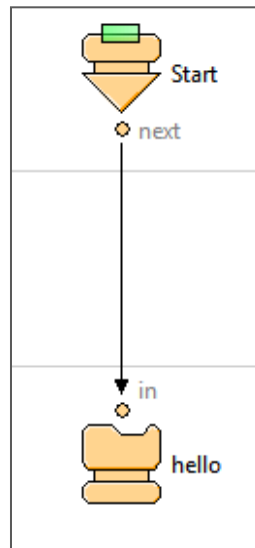


## Pipeline Debugger

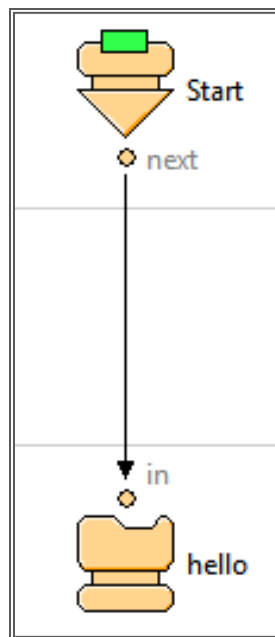
The Pipeline Debugger operates on the pipeline level, not at source code level. The Debugger allows for step-by-step tracking of pipeline execution and for examining the pipeline dictionary at runtime.

The Debugger requires a running Demandware system and a properly configured Remote Server connection. You will also need to create a debug configuration before running the Debugger.

In order to execute the pipeline debugger properly, you will need a pipeline with breakpoints set on at least one node of the pipeline.



When a pipeline debugger is launched, the breakpoint color changes from a semi-transparent green (shown above) to a solid green:





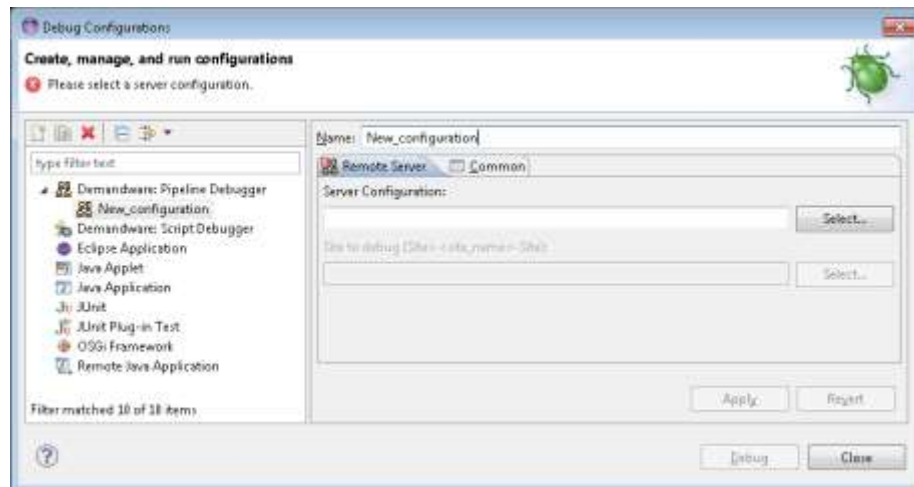
## 4.6. Exercise: Create a Debug Configuration

To create a debug configuration, follow these steps:

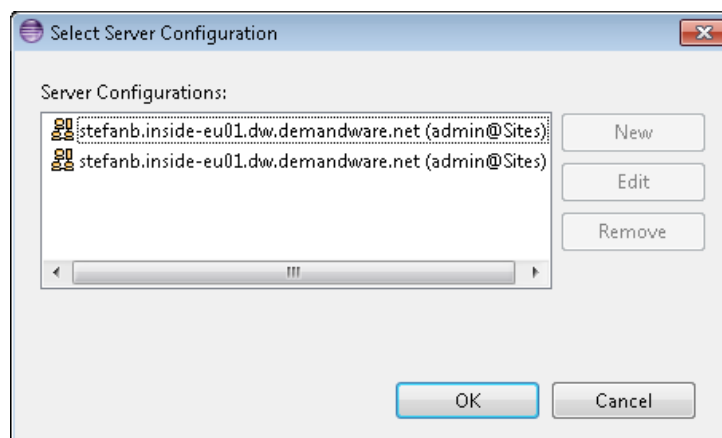
1. In Studio, select **Run→ Debug Configurations...** from the main menu or select **Debug Configurations...** from the drop-down menu under the green bug icon:



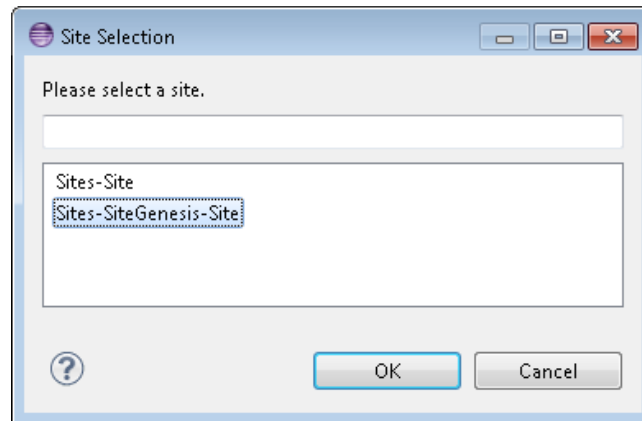
2. Double-click on **Demandware: Pipeline Debugger** to create a new configuration:



3. Give the configuration a name: **Pipeline Debug**.
4. Click the **Select** button from the **Server Configuration** section. Select the server connection you wish to debug then click the **OK** button:



- Click the **Select** button from the **Site to Debug** section. Select the site you wish to debug then click the **OK** button:



- Click **Apply** to save the configuration.
- Finally click **Debug** to start the debugger.



### Debugging a Pipeline Using the Debugger

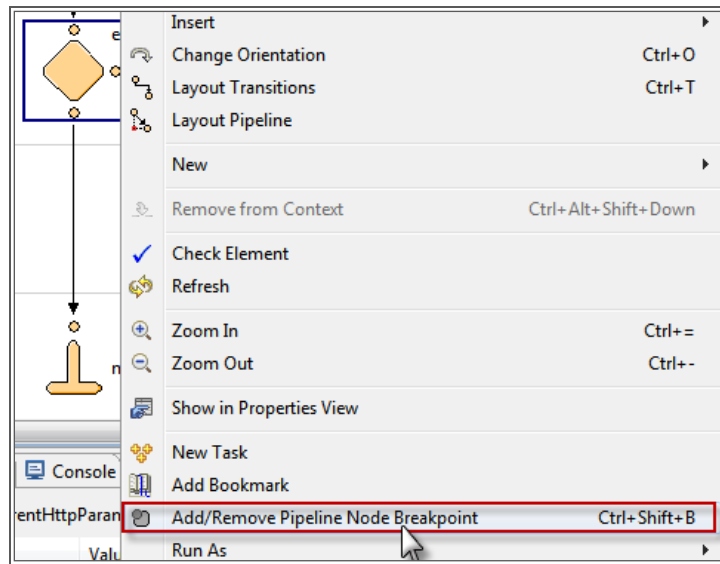
Once you have created a debug configuration, you still need one more step to use the Debugger. To execute a Pipeline Debugging Session, you will also need to set breakpoints in the pipeline where the Debugger will pause.

## 4.7. Exercise: Debug a Pipeline

Demonstrate how to create a break point on a pipeline, then execute the pipeline through the storefront. Show them how to step through the pipeline execution.

To set breakpoints on a pipeline for debugging, follow these steps:

1. Open the pipeline you wish to debug in Studio.
2. Click on a pipeline node you want the Debugger to pause at on execution.
3. Right-click your mouse and select '**Add/Remove Pipeline Node Breakpoint**'.

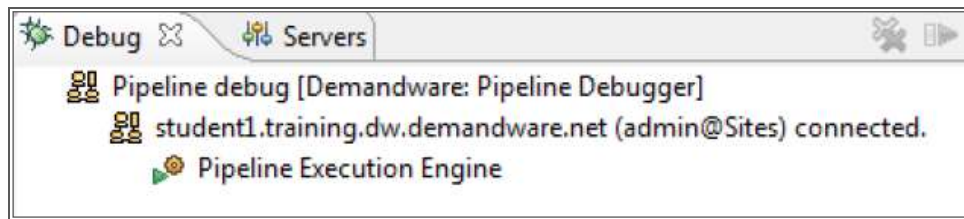


## Troubleshooting with the Pipeline Debugger, continued

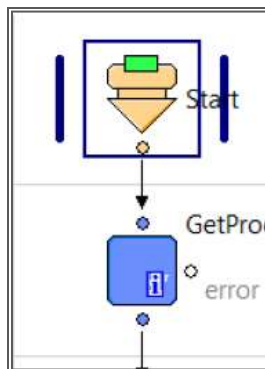


To debug a pipeline using a debug configuration, follow these steps:

1. In Studio, open the Debug perspective so that you can view a debugging session. Use one of the following methods:
  - a. **Window ⇒ Open Perspective ⇒ Other... ⇒ Debug** or
  - b. Click the **Open Perspective** icon on the upper-right corner and select **Other... ⇒ Debug**.
2. Start a debugging session:
  - a. From the Debug icon, select **Debug Configurations...** and double click the **Pipeline Debug** configuration.
  - b. Verify that the Pipeline Execution Engine is running.



3. In a browser, launch the pipeline you want to debug.
4. Click on the Demandware Studio icon on the taskbar, which should be blinking since the breakpoint was triggered. Sometimes the OS will switch context to Studio automatically, but this is rare.
5. In Studio, execution stops at the breakpoint, as indicated by the vertical bars:



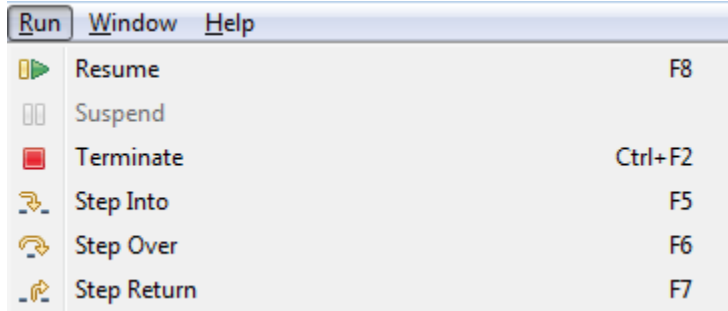


# Troubleshooting with the Pipeline Debugger, continued

4. The following **Debug Perspective** menu allows you to step thru the pipeline:



5. Or you can use the corresponding keyboard shortcuts:





### 4.8. Exercise: Use the Debugger

1. Create a pipeline debug configuration called **Pipeline Debug**.
2. Add a breakpoint on the **Call-Start** pipeline.
3. From the storefront, invoke the pipeline.
4. View the variables window when the debugger stops at the break point.
5. Using the F5 key, step next through the debugger. What are the values stored in the pdict?
6. Rerun the **Call-Start** pipeline but this time add a parameter at the end of the string:

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

7. Check the values of the **CurrentHttpParameterMap** after the Start node,
8. Continue stepping thru the pipeline and observe changes in the pdict.

# Pipelets

## Introduction

Now that we have introduced you to the pipeline dictionary, let's start using it to print information to a page using Pipelets.

A pipelet executes an individual business function within a Demandware pipeline. Pipelets are pre-coded pieces of functionality provided by Demandware but you can also use other types of pipelets from the palette such as:

- Script: use to invoke a custom Demandware script file
- Eval: use to evaluate data in the Pipeline Dictionary
- Assign: use to assign values to specific keys on the pdict

Demandware Pipelets are available in Studio via the Pipelets view. They belong to the **bc\_api** cartridge which is always downloaded as part of the Demandware API the first time you connect to a server. There is a published API available under Studio Help menus or in the Demandware documentation site.

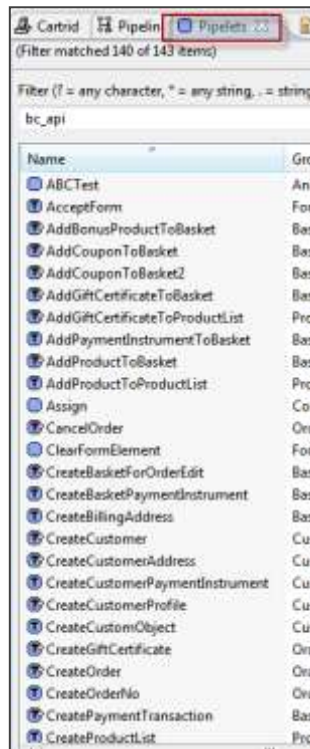
Each Demandware pipelet has documentation on its functionality, input and output parameters. You can see this information in the Properties view when the pipelet is selected on the pipeline.



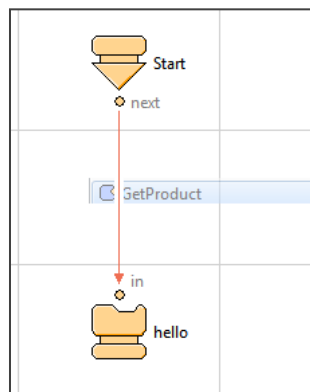
## Pipelets, continued

To access and use a Demandware API pipelet, follow these steps:

1. In Studio, open or create the pipeline you wish to add a pipelet to.
2. Open the pipelet view tab.

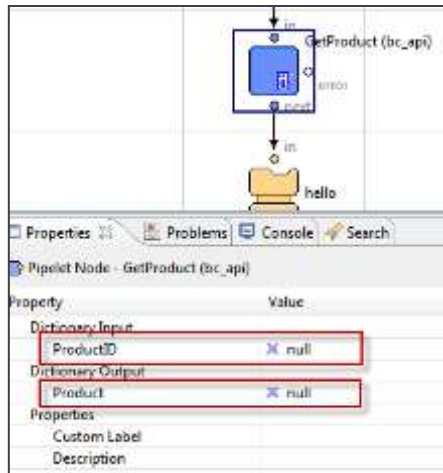


3. From the pipelet view tab, drag and drop the pipelet onto the transition node between the two nodes you want the pipelet to execute. Be sure the transition node turns red when you hover your mouse over it. Otherwise the pipelet will not be connected to the node.

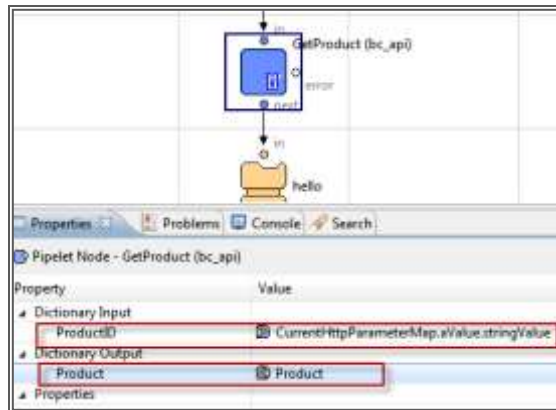


## Pipelets, continued

- Depending on the pipelet you are using, you will need to configure the pipelet for execution in the properties tab of the pipelet. In the example below, a GetProduct pipelet is used to create a product object by using a ProductID value. The value for the input will come from a stored variable in the pipeline dictionary. The product object output will need a name value.



Undeclared values



Declared values

- Save your pipeline.



#### 4.9. Exercise: Use a Pipelet in a Pipeline

1. Create a new pipeline called `ShowProduct`. Add a **Start** node connected to an **Interaction** node.
2. In the Pipelet API (use Studio Help), study the usage, input and output parameters for the `Catalog ⇒ GetProduct` pipelet.
3. Drag and drop the `GetProduct` pipelet between the **Start** node and **Interaction** node. Save your pipeline.
4. In Studio, verify that your debugging session is still active, and start a new one if necessary.
5. Put a breakpoint on the start node of the pipeline.
6. From your storefront, request the `ShowProduct-Start` pipeline appending a URL query string containing the product id:  
`ShowProduct-Start?pid=P0048`.
7. In the debugger, locate the `CurrentHttpParameterMap` in the Variables window.
8. Expand the `CurrentHttpParameterMap` variable to find the product ID parameter.
9. Look at the value of the `CurrentHttpParameterMap.pid.stringValue`.
10. Execute through the end of the pipeline (press F5).
11. In the debugger, right-click the `GetProduct` pipelet and right-click to select the **Show in Properties View** menu option.
12. Enter the pipelet input and output values so that the product ID gets passed to the pipelet:

Property	Value
Dictionary Input	
ProductID	<code>CurrentHttpParameterMap.pid.stringValue</code>
Dictionary Output	
Product	<code>myProduct</code>
Properties	
Custom Label	
Description	

13. Save your pipeline.

14. Debug the pipeline again to see when the `Product` object gets added to the pipeline dictionary.

Save the `hello.isml` template as a new template `product.isml` in your cartridge.

15. Change the `ShowProduct-Start` pipeline to display the new product template.
16. For the error exit case of the pipelet connect it to another new **interaction** node calling a new template `productnotfound.isml` printing the `pdict` log info :

```
<h1> The product ID ${pdict.CurrentHttpParameterMap.pid.stringValue}  
does not exist</h1>
```

17. In the template for the successful case, show the **product name** using the `myProduct` object found in the pipeline dictionary:

```
<h1>The product name is ${pdict.myProduct.name}</h1>
```

18. Debug the pipeline again to see the product name.

## Review & Lab

Pipeline Review Questions	Answer
<p>1. Each pipeline can have:</p> <ul style="list-style-type: none"> <li>a) At least a Join node and a Transition node.</li> <li>b) At least one Start node and a Jump node.</li> <li>c) At least a Start node and an Interaction node.</li> <li>d) B or C</li> </ul>	
<p>2. The Pipeline Dictionary holds the following default objects:</p> <ul style="list-style-type: none"> <li>a) GlobalDefault</li> <li>b) CurrentHttpParameterTable, CurrentScope, CurrentRequest, CurrentForms, CurrentPipeline</li> <li>c) CurrentHttpParameterMap, CurrentSession, CurrentRequest, CurrentForms, CurrentCustomer</li> <li>d) CurrentCustomer, CurrentSession, CurrentRequestor, CurrentFormValue, CurrentHTTPParameterMap</li> </ul>	



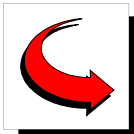
#### 4.10. Exercise: Test your knowledge of pipelines

1. Create a new pipeline called `Basket` and save it in your cartridge.
2. Add a **Start** node and **Interaction** node, which renders a `basket.isml` template (you will add code to this template later).
3. Add a `GetBasket` pipelet to the transition between the previous nodes.
4. In the `GetBasket` pipelet properties, add a `Basket` object as the output of the pipelet:

Pipelet Node - GetBasket (bc\_api)

Property	Value
Configuration	
Create	false
Dictionary Output	
Basket	Basket
StoredBasket	null
Properties	
Custom Label	
Description	

5. In one browser tab, open SiteGenesis and add at least 3 products to your cart.
6. Start the pipeline debugger and put a breakpoint after the `GetBasket` pipelet.
7. Invoke the `Basket-Start` pipeline to trigger the breakpoint.
8. Inspect the values in the `pdict`:
  - a. What values are stored in the `Basket` object in the `pdict`?
  - b. How would you access the product name for each product in the basket?
9. Do not worry if your pipeline does not display anything for now: you will write a loop later to display all the products.



#### Transition to Internet Store Markup Language (ISML)

## 5. Internet Store Markup Language (ISML)



---

### Goal

The purpose and goal of this module is to teach you the most commonly used Internet Store Markup Language (ISML) tags and expressions.

---



---

### Time

4 Hours

---



---

### Overview

We will define and give examples of using the following ISML tags: `<isset>`, `<isinclude>`, `<isdecorate>`, `<isloop>`, `<ismodule>` and `<isif>` tags.

---



---

### Materials Needed

Solutions cartridge

---

# ISML Templates



## Introduction

Internet Store Markup Language or ISML templates are files with an extension of `.ismml` which 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 Internet Store Markup Language (ISML) tags and expressions.

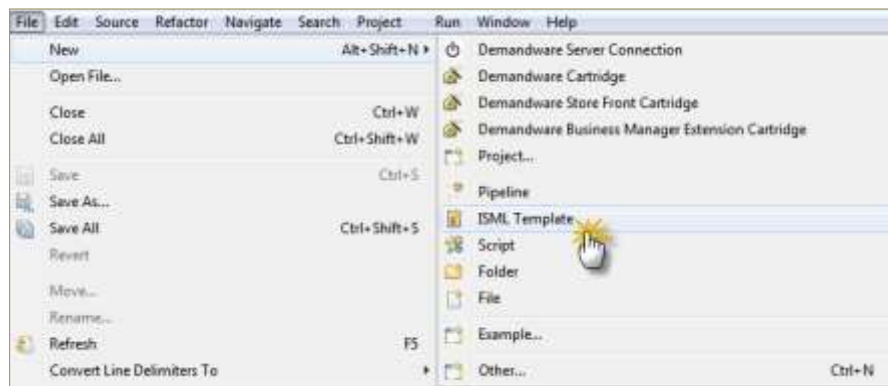
When describing a Demandware application using the Model-View-Controller (MVC) pattern, templates are the “view”, pipelines are the “controller” and the DW Script API is the “model”.

Run the Create an ISML Template activity..

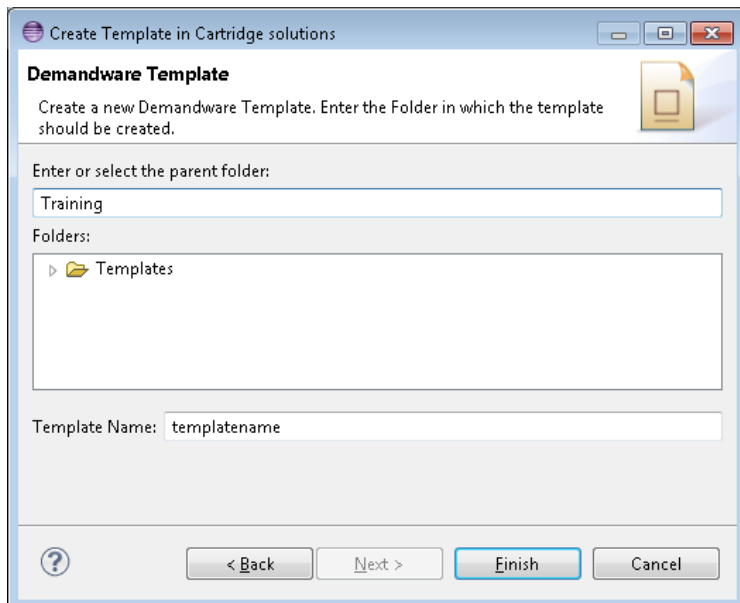


To create an ISML template, follow these steps:

1. Open Studio UX
2. With a cartridge selected in Navigator View, click **File→New→ISML Template**

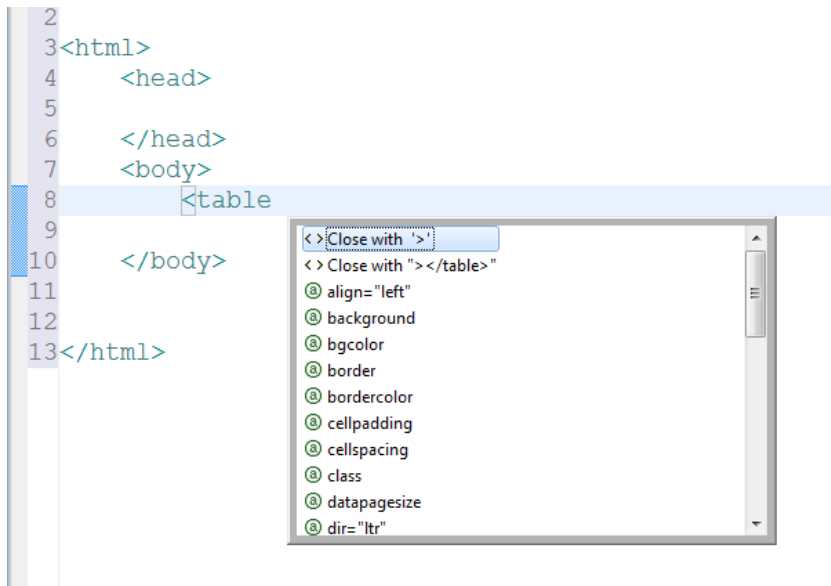


3. The **Create Template** window opens.
4. Specify what folder you want to store your template in the **Parent Folder** box. If the folder does not exist it will be created.

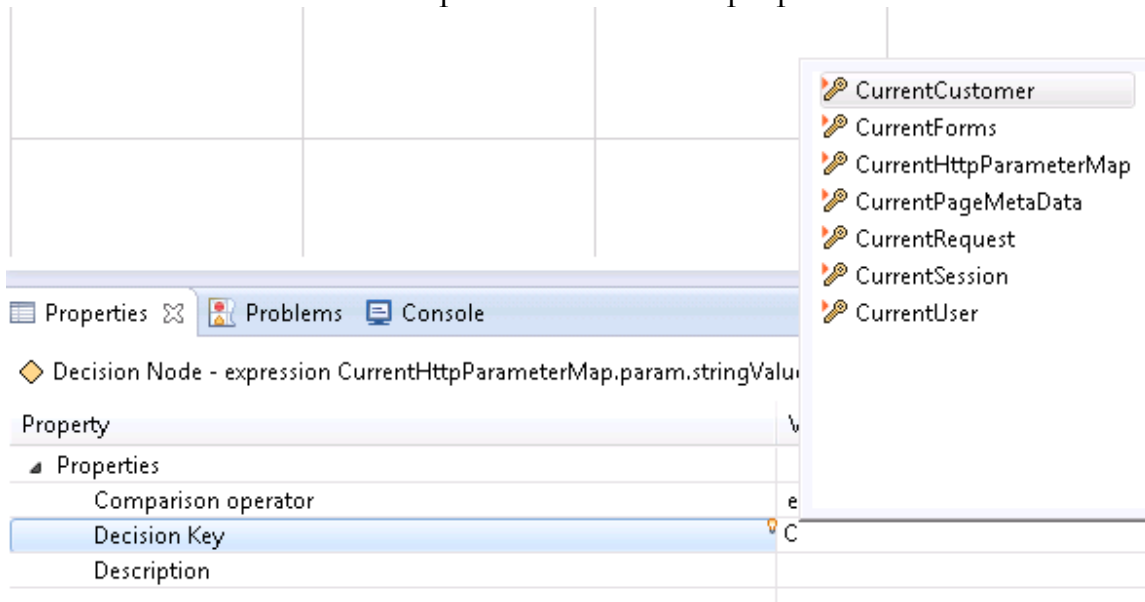


5. Enter a name for your template in the **Template Name** box. There is no need to type the .isml extension.

- Click the **Finish** button.
- Your new template opens in the ISML editor in UX Studio:



This editor supports HTML and ISML system tag auto-completions as shown above. It features as well an auto-complete for some node properties as below:



# ISML Tags and Expressions



## Introduction

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	<code>&lt;iscookie&gt;</code>	Sets cookies in the browser
	<code>&lt;iscontent&gt;</code>	Sets the MIME type
	<code>&lt;isredirect&gt;</code>	Redirects browsers to specific URLs
	<code>&lt;isstatus&gt;</code>	Define status codes
Flow Control	<code>&lt;isif&gt;</code>	Evaluates a condition
	<code>&lt;iselse&gt;</code> <code>&lt;iselseif&gt;</code>	Specifying alternative logic when an <code>&lt;isif&gt;</code> condition does not evaluate to true.
	<code>&lt;isloop&gt;</code>	Creates a loop statement
	<code>&lt;isnext&gt;</code>	Jumps to the next iteration in a loop statement
	<code>&lt;isbreak&gt;</code>	Terminates loops
Variable-related	<code>&lt;isset&gt;</code>	Creates a variable
	<code>&lt;isremove&gt;</code>	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 allow 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. Here is an example of accessing a property of the `Product` object in the pipeline dictionary:

```
${pdict.Product.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. Some packages are available implicitly, so classes do not need to be fully qualified:

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

Other access to classes and methods must be fully qualified:

```
dw.system.Site.getCurrent()
```

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

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

With some HTML experience, most ISML tags and expressions don't require much example for you to understand how to use them. For this module, we will cover the most frequently used tags: `<isset>`, `<isinclude>`, `<isdecorate>`, `<isloop>` and the conditional tags `<isif>`, `<iselseif>`, and `<iselse>`

We will cover more tags in subsequent modules.

### IMPORTANT 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.



# Creating and Accessing Variables



## Introduction

You can create and access your own custom variables in an ISML template by using the `<isset>` tag (to create and access variables in Demandware Script, see the [Demandware Script](#) module)

When using the `<isset>` tag, the required attributes that must be assigned are name and value. 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"
>
```

The *value* attribute can be either a hardcoded string or number or an ISML expression accessing another variable or object:

Value Type	Example
string	value="hardcoded text"
expression	value="{pdict.Product.name}"

The scope attribute of a variable refers to the level of accessibility of the variable. The three available levels to you as a developer are `session`, `request`, and `page`.

## Variable Scope

Demandware offers the following scopes, listed here from widest to narrowest access. It is important to understand the different scopes of a variable and what objects can access a variable at each level.

- Global Preferences: accessible via the `dw.system.OrganizationPreferences` class. Available to any site within an organization.
- Site Preferences: accessible via the `dw.system.SitePreferences` class. Available to any pipeline executing as part of a site.

- `session`: available through the whole session.
- `pdict`: available while a pipeline executes. It might encompass multiple requests, like when using **Interaction Continue** Nodes (covered in the Forms Framework module).
- `request`: available through a request-response cycle. Typically the same as the pipeline scope.
- `page`: available only for a specific page, and its locally included pages.
- `slotcontent`: available only in the rendering template for a content slot.
- `<isloop>` variable: available only inside the loop.

Session variables are available across multiple requests within a customer session. 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}
```

Request variables are available via the `request` scope. A request variable is available only for a single browser request-response cycle: it does not persist in memory for a subsequent request. Similar to session variables, you must prefix request variables with a qualifier `request.custom` when accessing them:

```
${request.custom.myRequestVar}
```

Page variables are available on the current ISML page, and their scope is limited to the current template, and any locally included templates (this will be covered later). They are accessed without a prefix:

```
${pageVar}
```



### 5.1. Exercise: Setting and retrieving variables

1. Create a new pipeline called `VarTest`. Add a **Start** node and an **Interaction** node to the pipeline.
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:
5. The value of the `sessionVar` variable is:  
`${session.custom.sessionVar}<br/>`
6. Open a web browser and test the pipeline.
7. Add similar examples of request and page variables to the `varTest` template and display them.
8. Modify the examples using boolean and string values, i.e. `"${false}"` and `"Hello"`.
9. Test the pipeline again to see the new variable values.
10. Increment the variables by using syntax like  
`value="${request.custom.requestVar + 1}"`
11. Explain your findings.

# Reusing Code in Templates

## Introduction

There are four tags available to you as a developer for reusing code in ISML templates:

- `<isinclude>`
- `<isdecorate>`
- `<ismodule>`
- `<iscomponent>`

In this course we will cover `<isinclude>`, `<isdecorate>`, and `<ismodule>`.

The tags above allow you to write reusable code in an ISML template. Reusable code:

- Saves time
- Reduces the likelihood of errors
- Minimizes the chore of updating pages
- Helps to ensure a consistent look and feel

## The `<isinclude>` Tag

The `<isinclude>` tag allows you to embed an ISML template inside the invoking template. There are two types of includes:

- **Local Include** – allows you to include the code of one ISML template inside of another while generating the page
- **Remote Include** – allows you to include the output of another pipeline inside of an ISML template. The primary purpose for using a remote include is for partial page caching. We will discuss caching later in this book.

## Local Includes

Local includes are heavily used throughout SiteGenesis. The syntax for using a local include is:

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

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

### Local Includes Example

*Template 1:*

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

*Template 2 (calendar.isml)*

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

When the template gets rendered by the browser, the user will see:

**My Template**

**Included template**

### IMPORTANT NOTE

- **Local Include** -All variables from the including template are available in the included template, including page variables.
- **Remote Include** -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.
- Includes from another server are not supported.

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:

```
1<!-- TEMPLATENAME: hello.isml --->
2<html>
3<head>Hello Pipeline</head>
4<H1>
5<isinclude template="account/newslettersignup"/>
6</html>
```

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



### 5.2. Exercise: Local Include

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 `ShowProduct` pipeline from the last chapter.
  - b. Look up the output of the `GetProduct` pipelet and compare it to the expected `pdict` variable in the `producttile` template: your pipelet outputs a `pdict.myProduct` but the template expects `pdict.product`. These are not the same variables!
  - c. Open the `product.isml` template from the `ShowProduct` 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:  
`ShowProduct-Start?pid=P0048.`



### Remote Includes

The syntax for a remote include is

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

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

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

In the example above, 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 `URLUtils` methods instead.

Here is an example of passing URL parameters:

```
<isinclude url="${URLUtils.url('BrowseCatalog-Hotdeals',
'catalogCategoryID', 'Storefront')}" />
```

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

There is a new tag that implements a remote include

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

This tag allows you to pass as many attributes as you want without having to use the `URLUtils` methods:

```
<iscomponent pipeline = "Product-Show" productid = "1234"
name = "Wide-screen television" />
```





### Run the Remote Include activity.

Demonstrate how to use a Remote Include by creating two pipelines where 1 is calling the other using the Remote Include tag.



### 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.



### 5.3. Exercise: Remote Include

1. Study the Demandware Script API help for the `dw.web.URLUtils.url()` method.
2. Locate and study the `Product-IncludeLastVisited` pipeline in the storefront cartridge.
3. Study the `lastvisited.isml` template, specifically:
  - a. The use of the `<isloop>` tag.
  - b. The use of the `pdict.LastVisitedProducts`.
4. Open the `ShowProduct` pipeline and the `product.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:

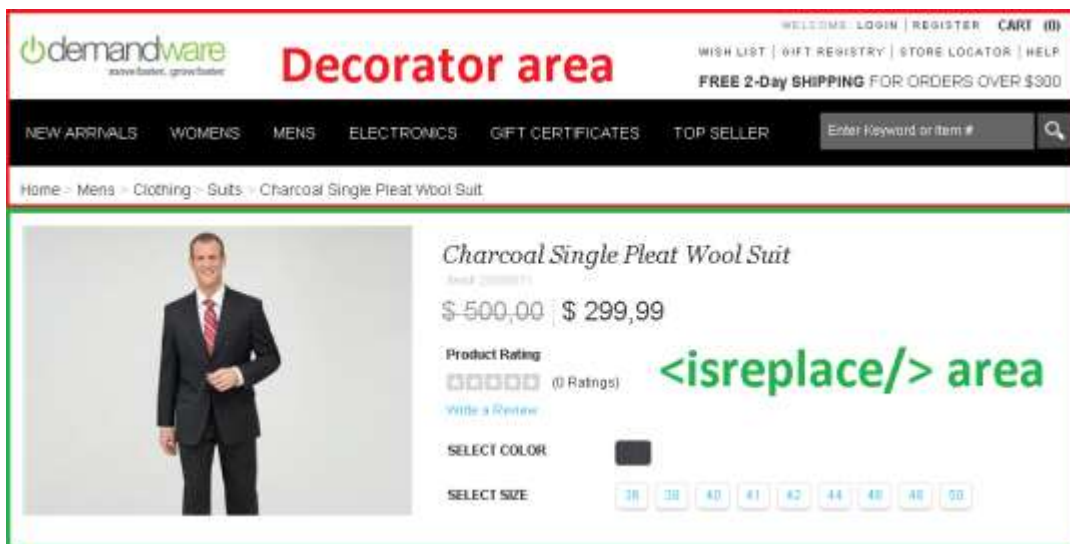
```
<isinclude url="${URLUtils.url('Product-IncludeLastVisited')}"/>
```
6. Test the pipeline with an existing product: `ShowProduct-Start?pid=P0048`.
7. On a different browser window, visit at least 3 other products in the storefront.
8. Retest the pipeline: all the visited products should appear.

# Reusing Code in Templates



## Using the <isdecorate> Tag

The <isdecorate> tag lets you decorate the enclosed content with the contents of the specified (decorator) template. A decorator is an ISML template that has html, css, and overall design of a page. The decorator template has the tag <isreplace/> identifying where the decorated content shall be included. The screenshot below shows an example of a decorator and the area where the code is being replaced.



Typically, only one tag (<isreplace/>) is used in the decorator template. However, multiple tags can also be used. If the decorator template has 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="[Decorator template name]">
...My content...
</isdecorate>
```

+	=	<i>Final generated page</i>
Decorator template		<html>
<html>		<head>...</head>
<head>...</head>		<body>
<body>		...My content embedded...
<isreplace/>		</body>
</body>		<html>
<html>		

## Reusing Code in Templates, continued



Run the use `<isdecorate>` tag activity.

Demonstrate how to use the `<isdecorate>` tag by decorating one template with another.



1. To use the `<isdecorate>` tag, follow these steps:
2. Open the ISML template that has the code you wish to replace in a decorator. Add the `<isdecorate>` tag around the code you wish to include in a decorator.
3. `<isdecorate template="[directory/]decoratorname">`  
     Your code goes here.  
     `</isdecorate>`
4. Save the template.
5. Open the decorator template. If you are using a SiteGenesis template, the decorator templates names start with "pt\_".
6. Find the location in the code where you want to use the `<isreplace/>` tag. Add the tag to the template.
7. Test the page by calling the pipeline that uses the decorator template. For example, if the decorator template is used by the 'Account-Show' pipeline/start node, type in the URL that will execute the 'Account-Show' pipeline.

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



### 5.4. Exercise: Using a Decorator

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

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

7. Test the pipeline with an existing product: `ShowProduct-Start?product=P0048`.

# Creating Custom Tags with <ismodule>



The <ismodule> tag allows you to define your own ISML tags which can be used like any standard tags.

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. See the example below in `util/modules.isml`:

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

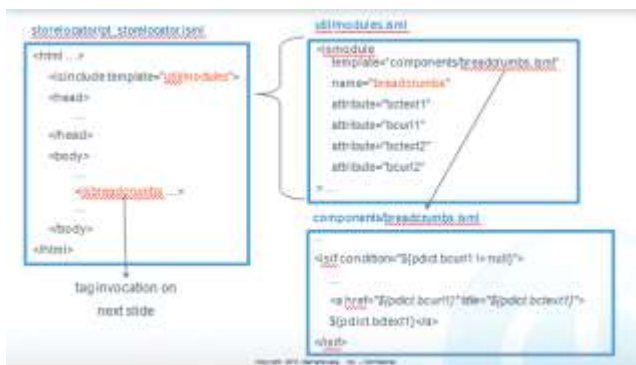
2. The isml file which specifies what happens when the attributes are passed. See the code snippet from inside `breadcrumbs.isml`:

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

3. Lastly, you need to invoke the custom tag inside an isml template:

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

Putting it all together would look as such:





### 5.5. Exercise: Using Custom Tag

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

1. Open the `util/modules.isml` template.
2. Locate the `producttile` custom tag definition.
3. Note the different inputs defined for the `producttile` custom tag.
4. Locate the template that implements this custom tag, and study it: `producttile.isml`.
5. In the `ShowProduct-Start` pipeline, open the `product.isml` template.
6. Remove the remote include.
7. Change the existing local include to include the template that contains all custom tag definitions:

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

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

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

9. Test the pipeline with an existing product: `ShowProduct-Start?product=P0048`.
10. 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}" />
```

11. Test the `ShowProduct-Start` pipeline again.

# Conditional Statements and Loops



## Introduction

Every programming language gives a programmer the ability to evaluate a condition in order to determine what logical path the program should take. Most every language uses the keywords *if*, *else if*, and *else*. Demandware uses similar keywords as well but adds “*is*” to the beginning of the syntax:

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



## Run the Conditional Statements activity.

Demonstrate how to use the `<isif>` tag in a template then show the results in the storefront.



**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.
3. Here is an example:

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





### Loops

With `<isloop>` you can loop through the elements of a specified collection or array. As an example, you can list data like categories, products, shipping and payment methods. An `<isloop>` statement can be nested in one another.

### Supporting tags

The `<isbreak>` tag can be used within a loop (defined by an `<isloop>` tag) to terminate a loop unconditionally. If `<isbreak>` is used in a nested loop, it terminates only the inner loop.

Use `<isnext>` to jump forward in a loop. Jumping 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>
```

## Instructor Guide

The attributes have the following usage:

Attribute	Description
<code>items</code> ( <code>iterator</code> )	Expression returning an object to iterate over. Attributes <i>iterator</i> and <i>items</i> can be used interchangeably.
<code>var</code> (alias)	Name of the variable referencing the object in the iterative collection referenced in the current iteration.
<code>status</code>	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.
<code>begin</code>	Expression specifying a begin index for the loop. If the begin is greater than 0, the <code>&lt;isloop&gt;</code> skips the first x items and starts looping at the begin index. If begin is smaller than 0, the <code>&lt;isloop&gt;</code> is skipped.
<code>end</code>	Expression specifying an end index (inclusive). If end is smaller than begin, the <code>&lt;isloop&gt;</code> is skipped.
<code>step</code>	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
<code>count</code>	The number of iterations, starting with 1.
<code>index</code>	The current index into the set of items, while iterating.
<code>first</code>	True, if this is the first item while iterating ( <code>count == 1</code> ).
<code>last</code>	True, if this is the last item while iterating.
<code>odd</code>	True, if count is an odd value.
<code>even</code>	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">`



### 5.6. Exercise: Using Loops

1. Open the `Basket` pipeline and the `basket` template.
2. Implement the following code to display the products in the cart:

Basket product names are:

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

3. Open a browser to your storefront. Add products to the cart first, including a product with an option (like a TV warranty).
4. Open another browser tab and invoke the `Basket-Start` pipeline.
5. Add a `status` attribute in the `<isloop>` tag so that you can see what the `count` and `index` parameters return.
6. Replace the `allProductLineItems` property of the `basket` with the method `getProductLineItems()`.

## Review & Lab

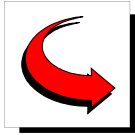
Question	Answer
1. What ISML tag is used to create a variable?	
2. What ISML tag is used to format output to a page?	
3. What ISML tag is used to include a local template?	
4. What ISML tag is used to create a loop statement?	
5. What Storefront Toolkit tool helps you to troubleshoot pipeline execution errors?	



### 5.7. Exercise: Complete Basket Pipeline

Modify the `basket` template so it distinguishes that the `Basket` holds regular products or option products.

1. Find out which key in the `pdict.Basket` object can be identified as option product name of all items inside the basket.
2. Enhance the `Basket` loop with a check of product or option product based on the key you have identified. Finally print out product names and option product names.



**Transition to Content Slots**

## 6.Content Slots



---

### Goal

The purpose is to utilize content slots to improve the appearance of your site and to better showcase products and promotions.



---

### Time

2 Hrs



---

### Overview

We will demonstrate how to create content slots, their corresponding rendering templates, and the use of content link functions within content slots and HTML attributes in general.



---

### Materials Needed

contentslot cartridge

---

# 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.

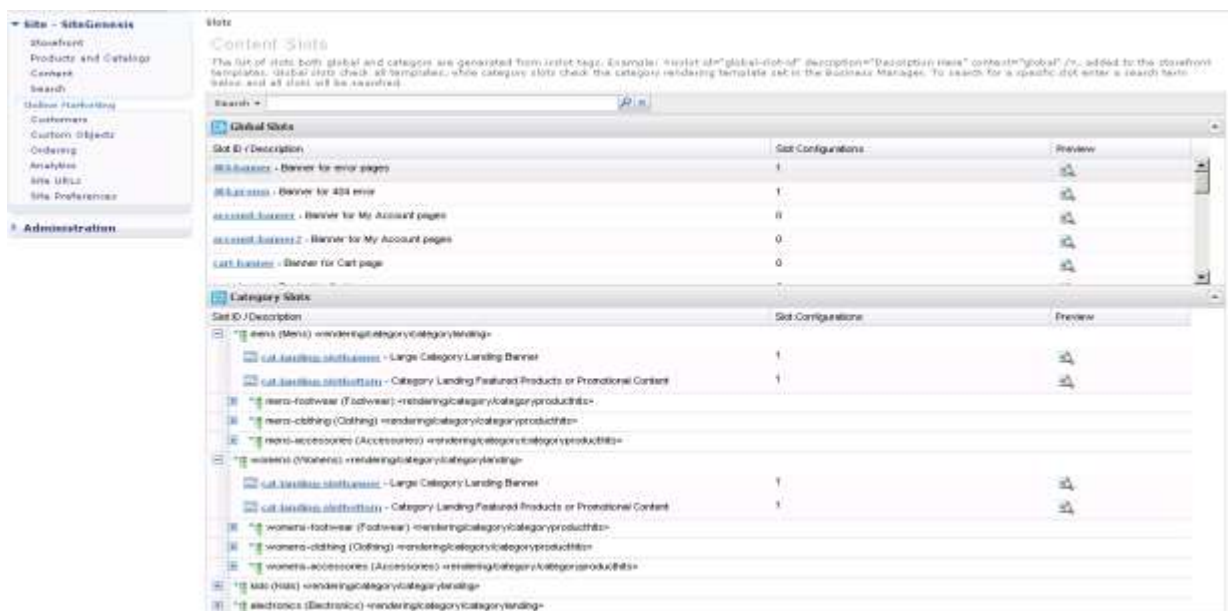
Slots can easily be found on a page by using the **Storefront Toolkit > Content Information** tool, as shown below. Hover the mouse around the page to reveal where content slots are used, and to get a link to the slot's configuration page in Business Manager:



## Content slots vs. Content Assets

Slots are different from content assets in several aspects:

- Slots are considered part of marketing. They are located under **Site > Online Marketing > Content Slots**. Content Assets appear under the Content section.



- 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 do not have such qualifiers.

### Creating 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.

Slots can be of two kinds:

- Global slots: can appear on any page.
- Category slots: appear on category-specific pages since they depend on the category id.

A content slot will display content of one of the following types:

- 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 AB testing are out of the scope of this course.



# Creating Content Slots - Developer Tasks



## Creating a Content Slot

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:

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

- Category slot:

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

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

As stated in the previous section, 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 below 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 (next section) 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"/>
```

While we have not covered the `<isprint>` tag in detail, there is extensive documentation and usage examples for it in SiteGenesis.

Using the `encoding="off"` setting allows the HTML snippet to be generated without encoding, so that the browser renders it correctly. The static text and styling all render as follows:

**FREE 2-Day SHIPPING** FOR ORDERS OVER \$300



### 6.1. Exercise: Creating a Slot

In this exercise we will 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 doesn't exist.
2. Investigate which template is been 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 Slots - Merchant Tasks



## Creating a Content Slot Configuration

The merchant creates a content slot configuration by navigating to **Site > Online Marketing > Content Slots**, then finding the specific slot the developer created, e.g. header-banner.

Slots > Slot 'header-banner' - (Global)

Slot Configuration(s)	
Header banner within the header, between the logo and search bar, width 1000px, by height 100px.	
Click <b>New</b> to create a new slot configuration, and click on a specific slot configuration to <b>Edit</b> that slot configuration. To <b>Delete</b> specific slot configuration, click on the <b>Delete</b> icon.	
ID	Description
free2dayshiporderover300 (default)	HTML slot for Free 2 Day Shipping

The merchant selects an existing configuration or clicks **New** to create a new one:

Slot Configuration - New Slot Configuration - 2012-07-23 14:14:43

The **Description** field is for an internal description, the **Callout** field is for a storefront message. The **Default** checkbox will be the slot configuration that is used if multiple slot configurations are running on the same schedule. If multiple slot configurations are scheduled and have the same **Rand** category products, **Content Asset** is for a chosen content asset, and **HTML** is for plain text or HTML code.

Fields with a red asterisk (\*) are mandatory.

Select language: Default

ID: New Slot Configuration - 2012-07-23 14:14:43

Enabled: No

Default: ☐

Description:

Slot Content

Content Type: Product

Product:

Template: slots/product/product\_1x2.isml

Callout:

HTML Editor

Schedule/Qualifiers/Compatibility

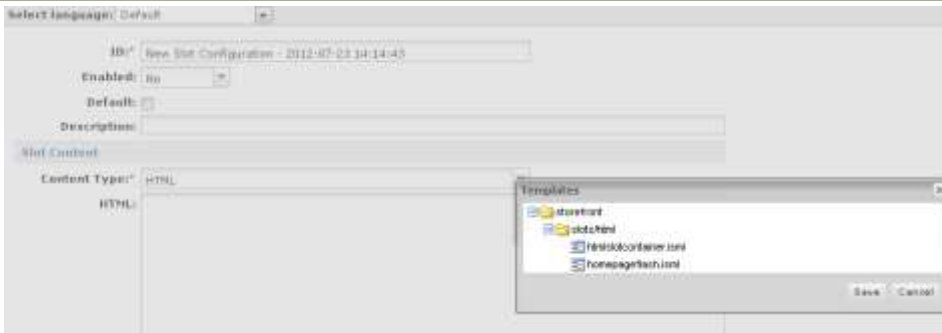
Add Schedule \*

ID

Click Add Schedule to create a schedule for this Slot Configuration. You can create a single "default" schedule or schedule the slot configuration to a Campaign.

Here the merchant enters an ID, enables the slot and provides other relevant information. The slot content that follows is the most important piece for the programmer since it defines what data becomes available for the rendering template.

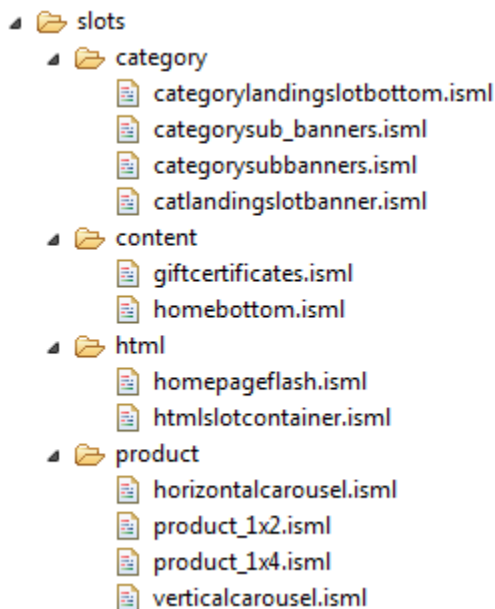
For example, if the slot is to show HTML, the merchant might configure the slot content as follows:



The merchant gave the banner an ID and enabled the slot. The content type HTML was selected, which opens the HTML text area (you can use the HTML Editor if you want to compose the HTML from scratch). Then the ellipsis on the Template area was clicked: this opens the Templates dialog shown above, and all the possible rendering templates for this type discovered in all cartridges in the cartridge path.

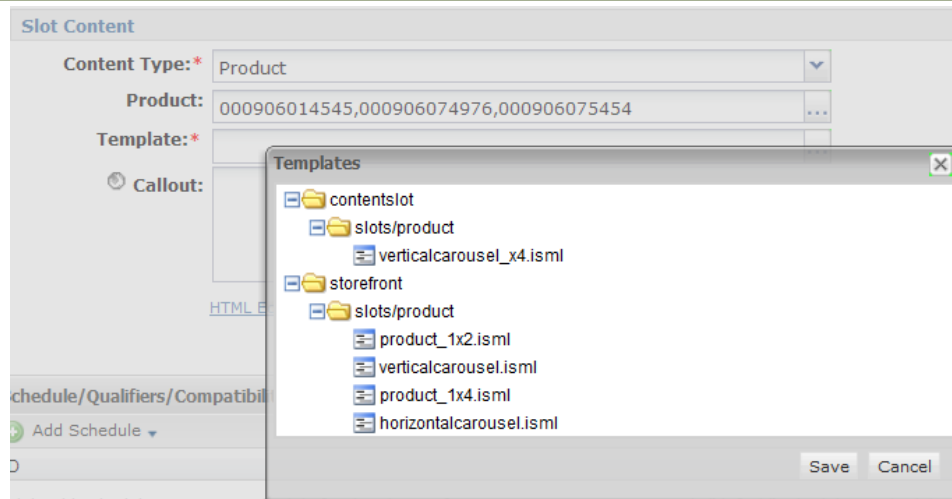
The SiteGenesis storefront cartridge comes with some default templates for every type of content. Furthermore, the templates are located in especially named folders that Business Manager discovers by default: `slots/html` for the HTML type in the example above.

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



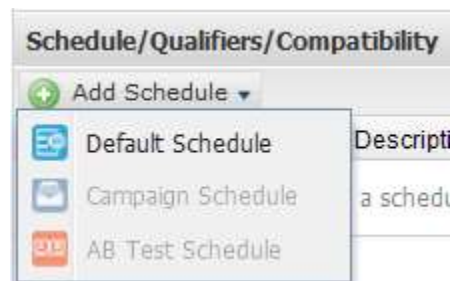
The merchant may choose to reuse an existing rendering template, or use a new one as instructed by the programmer. This is why the collaboration between merchant and programmer is important: without a rendering template there is no way to visualize the slot.

Another example with a different content type might look like this:



Here the merchant selects the Product content type, selects some product IDs, and then has to choose from all the available product-specific rendering templates. Notice that any slots/product templates available in any cartridge in the path become visible.

Finally, the merchant needs to provide a schedule for the slot. This is either the default schedule or based on a marketing campaign:



For this course we choose the Default Schedule. A discussion of the other choices is out of the scope of this course, but it is covered in the Managing the Storefront course.

### 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` which was discussed previously.

When using HTML in assets or slots it might be tempting to hardcode hyperlinks to pages or images in the storefront: this is a bad idea since those links are instance-specific (i.e. Staging) and would have to be changed every time after a replication.

To solve this problem, Demandware offers special Content Link Functions that are used only in attributes of type HTML. The content link functions are:

- `$staticlink$` - 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, but with the http protocol.
- `$httpsUrl()` - Creates an absolute URL, but with the https protocol.
- `$include()` - Can be used to make a remote include call (relevant for caching purposes).

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

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

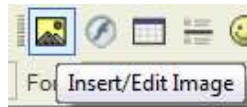


## 6.2. Exercise: Creating a Slot Configuration

In this exercise you 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.
3. Create a new configuration for the slot:
  - a. Give it an ID: banner-for-everyone.
  - b. Enable it.
  - c. Make it the default.
  - d. Select HTML for the content type.
  - e. Using the HTML editor:

- Click on 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></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 weird string: the `nohits` page should display with the new slot visible.

## Review & Lab

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



### Exercise 6.3: Create Content slot with Rendering Template

In this lab you will 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 `storefront/cartridge/templates/default/slots/product/verticalcarousel.isml` to the `training/cartridge/templates/default/slots/product` folder. 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 it to match the following code. The highlighted things have to be changed. You can copy paste from this place only if time is short but do it with understanding.

```
<iscache type="relative" minute="30" varyby="price_promotion"/>
<isinclude template="util/modules"/>

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

7. Create a template named `pt_productsearchresult_UI.isml` at 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. Navigate to the contentslot cartridge and access the `/static/default/css/verticalcarouselx4.css` file.



Copy the `verticalcarouselx4.css` from the `contentslot` cartridge to **the same place in your cartridge**. (Create the directory structure if it is not there)

10. Navigate to the **BM → Site → SiteGenesis → Online Marketing → Content Slots**.
11. 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).
12. Navigate to the **storefront** from **BM** in the browser. Search for some **non-existent** product like **“MyBestPants”**
13. Verify that the `nohits.isml` shows both the previous banner and multiple products in a vertical carousel.



---

### Transition to Demandware Script (DS)

---

## 7.Demandware Script (DS)



---

### Goal

The purpose and goal of this module is to familiarize you with using Demandware Script in ISML templates and script files.



---

### Time

2 Hrs



---

### Overview

We will discuss the Demandware Script syntax, API, as well as how to create a new script pipelet.



---

### Materials Needed

Solutions

---

# Overview



## Introduction

The server-side language used for coding in the Demandware Platform is Demandware Script (DWScript). Demandware Script is based on JavaScript, which is standardized as ECMAScript. Demandware Script implements the standards ECMA-262 (=ECMAScript 3rd Edition) and the ECMA-357 standard, also known as ECMA for XML or E4X.

In addition, Demandware supports all JavaScript language extensions by Mozilla known as JavaScript 1.7. Demandware also supports optional type specification (from JavaScript 2.0/ECMA 4th edition proposal and ActionScript).

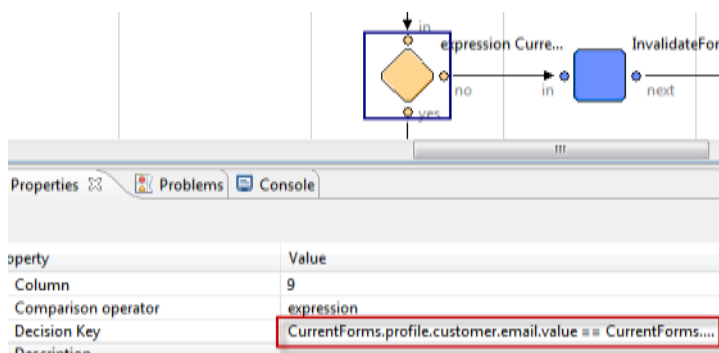
You use Demandware Script anywhere you need to access data about the system, such as products, catalogs, prices, etc.

You write DWScript in pipelines inside Decision nodes, and in ISML templates for expressions or inside `<isscript>` tags. But by far the most extensive use of Demandware Script is 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



## Overview, continued

### Demandware Script API

There is a well-documented API that gets published with every new Demandware update. The Script and pipelet APIs are available under the Studio Help menus. The ISML documentation (and more) is available in our documentation portal:

<https://info.demandware.com/DOC2/index.jsp>.

Demandware continually updates clients to the latest version by using a mechanism called global deployment. Deployments happen 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*:

Site Genesis Time Zone: Etc/UTC Time | Instance Time Zone: US/Eastern Time | Version: 14.1 (Compatibility Mode: 13.6)

Demandware will provide access to preview releases (14.1 as of this writing) by updating sandboxes prior to updating the P.I.G. instances. This will give your organization a chance to test any new API updates in 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 our 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



## Introduction

The Demandware Script API is organized in packages just like Java.

However, you cannot inherit from these classes or packages when you create a script, which is indeed different from Java. You can only use the properties and methods of these classes in your scripts.

## TopLevel package

This is the default package in Demandware Script, 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`.

Of interest here is the `TopLevel.global` class, which contains many of the common variables and constants used in pipelines and scripts, such as:

- Constants: `PIPELET_NEXT` and `PIPELET_ERROR`

These are used to indicate the result of a script pipelet and determine which exit the pipeline takes after pipeline execution

- Properties: `customer`, `request` and `session`

These are commonly used in scripts to get access to the current customer and the current session

## API Packages, continued

### IMPORTANT NOTE

In the following packages you will find many classes that end with the word `Mgr`, for example: `dw.catalog.ProductMgr`. The role of these classes is **to retrieve instances of business objects related to the package they belong to**: you 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 Packages	
<b>dw.campaign</b>	Campaign and promotion related APIs  Classes: <code>PromotionMgr</code> , <code>Campaign</code> , <code>Promotion</code> , <code>SourceCodeGroup</code> , etc.
<b>dw.catalog</b>	Catalog, product, price book related APIs  Classes: <code>CatalogMgr</code> , <code>Category</code> , <code>Product</code> , <code>Recommendation</code> , <code>PriceBook</code> , etc.
<b>dw.content</b>	Non-product content management related APIs  Classes: <code>ContentMgr</code> , <code>Content</code> , <code>Folder</code> , <code>Library</code> , etc.
<b>dw.customer</b>	Customer profile and account related APIs  Classes: <code>CustomerMgr</code> , <code>Customer</code> , <code>Profile</code> , <code>ProductList</code> , <code>OrderHistory</code> , etc.
<b>dw.order</b>	Order related APIs including basket, coupons, line items, payment, shipment  Classes: <code>Basket</code> , <code>Order</code> , <code>ProductLineItem</code> , <code>ShippingMgr</code> , <code>TaxMgr</code> , etc.



Generic Packages	
<b>dw.crypto</b>	Encryption services using JCA; DES, Triple-DES, AES, RSA, etc. Classes: Cipher, MessageDigest
<b>dw.io</b>	Input and output related APIs Classes: File, FileReader, CSVStreamReader, XMLStreamReader, etc.
<b>dw.net</b>	networking related APIs Classes: FTPClient, HTTPClient
<b>dw.object</b>	System base classes and custom object related APIs Classes: PersistentObject, ExtensibleObject, CustomObjectMgr, etc.
<b>dw.rpc</b>	Web services related APIs Classes: WebReference, Stub
<b>dw.system</b>	System functions Classes: Site, Request, Session, Logger
<b>dw.util</b>	Similar to the java.util API: collections, maps and calendar classes
<b>dw.value</b>	Immutable value objects Classes: Money, Quantity
<b>dw.web</b>	Web processing related APIs Classes: URLUtils, Forms, Cookie, HttpParameterMap, etc.

# Using Demandware Script in ISML



## Introduction

Demandware Script can be embedded into ISML by using the `<isscript>` tag. The example code below shows how to get the root category of a current site's navigation catalog as well as getting the category named 'sale' using Demandware script.

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

<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 choose to import any packages at the top of the script:

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

Run the Demandware Script in ISML activity.

Demonstrate how to use Demandware Script inside an ISML template.



### 7.1. Exercise: Using Demandware Script in ISML

1. Create a new pipeline called `DScript`.
2. Add a Start node and Interaction node.
3. Create a new ISML template named `dscript` and use it in the **Interaction** Node.
4. Using the `dw.customer.CustomerMgr` class, print the registered customer count.
5. Test your pipeline in the storefront.

# Script Pipelets



## Introduction

So far we have used Demandware pipelets to implement common functionality in a Demandware storefront: the `GetProduct` pipelet. In this lesson you will learn how to create your own script pipelets by writing custom Demandware Script files.

Demandware Script files have an extension of `.ds` and are stored in the `/cartridge/scripts` directory.

Script files like Pipelets, can have input and output parameters for data manipulation. The example below shows input/output parameters from a 'GetProduct' script:

Configuration	
OnError	PIPELET_ERROR
ScriptFile	solutions:product/GetProduct.ds
Timeout	
Transactional	false
Dictionary Input	
ProductID	CurrentHttpParameterMap.aValue.st
Dictionary Output	
Product	Product

When you create a new script file, the file will be preconfigured for scripting. The example below shows a brand new script file:

```

1/*
2 * Demandware Script File
3 * To define input and output parameters, create entries of the form:
4 *
5 * <paramUseType> <paramName> : <paramDataType> [<paramComment>]
6 *
7 * where
8 *   <paramUseType> can be either 'input' or 'output'
9 *   <paramName> can be any valid parameter name
10 *   <paramDataType> identifies the type of the parameter
11 *   <paramComment> is an optional comment
12 *
13 * For example:
14 *
15 *   @input ExampleIn : String This is a sample comment.
16 *   @output ExampleOut : Number
17 *
18 */
19
20importPackage(dw.system);
21
22function execute( args : PipelineDictionary ) : Number
23{
24    // read pipeline dictionary input parameter
25    // ... = args.ExampleIn;
26
27    // insert business logic here
28
29    // write pipeline dictionary output parameter
30
31    // args.ExampleOut = ...
32
33    return PIPELET_NEXT;
34}

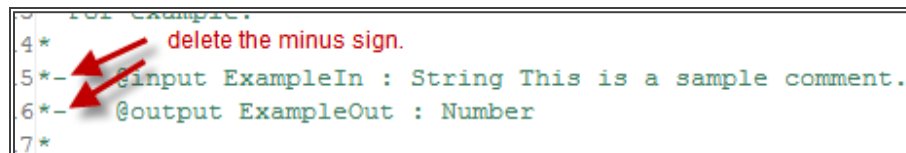
```

### Input & Output Parameters

Input and output parameters are configured within the script text. The script text below is an example of an input parameter that is called 'ProductID' which takes in a string value while the output parameter is called 'Product' and returns a product object:

```
6* @input    ProductID : String The product id coming from t
7* @output   Product : Object The product found
```

When you first create a script file, the input and output parameters are commented out with a \*- so you will need to delete the minus sign for the Demandware application to read the parameters:



The screenshot shows a code editor with the following lines: 4\* (commented), 5\*-\* @input ExampleIn : String This is a sample comment., 6\*-\* @output ExampleOut : Number, and 7\* (commented). Two red arrows point to the minus signs in lines 5 and 6, with a red text annotation 'delete the minus sign.' above them.

```
4*
5*-* @input ExampleIn : String This is a sample comment.
6*-* @output ExampleOut : Number
7*
```

For input/output data types, you can use `TopLevel` package classes such as `String`, `Number`, `Object`, etc. You can also specify any other data type as long as you fully qualify it: `dw.catalog.Product`. In the following snippet the `TopLevel.Object` data type allows you to avoid having to qualify the object you are returning:

```
@output Subscription : Object
```

```
@output Product : dw.catalog.Product Must fully qualify this output type
```

### Importing

When working with script files, if you access Demandware Script packages or classes other than `TopLevel`, you will need to import them in the script using the following syntax:

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

You can import a single class, if you wish:

```
importClass( dw.system.Logger );
```

You can import a custom script from the same cartridge as the current script you are writing:

```
importScript( "common/libJson.ds" );
```

## Instructor Guide

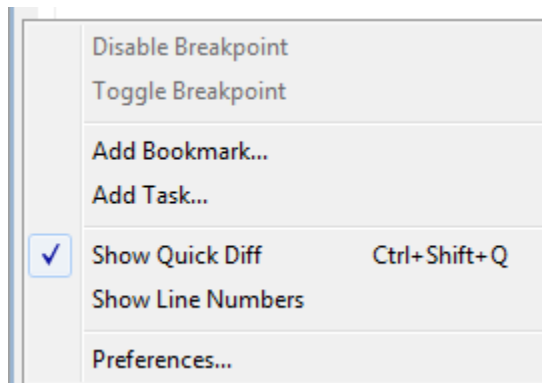
If you are accessing a script in another cartridge, make sure you specify the cartridge prior to the script name as the system will not look for further scripts with the same name in the cartridge path:

```
importScript( "<cartridge name>:[folder/]utilities.ds" );
```

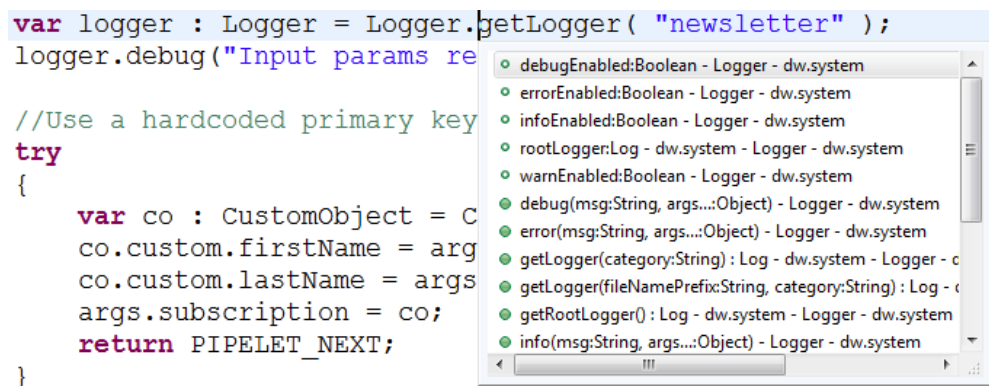
Of course, you can always fully qualify the access to a specific class and avoid importing that package/class.

### Using the Script Editor

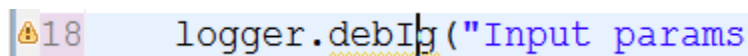
You can turn on line numbers on the script by right-clicking the gray column on the left side of the editor and selecting the **Show Line Numbers** checkbox:



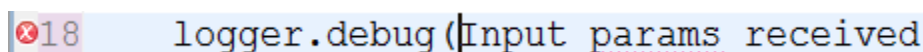
The editor offers auto-complete capabilities and syntax checking: use these tools to minimize coding errors. Studio will automatically generate code hints when you click **Ctrl+Spacebar**:



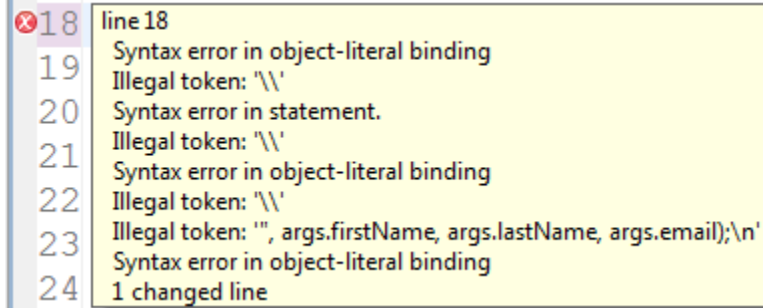
Warnings will appear for unknown methods after you save the file:



Syntax errors will be indicated as follows after you save:

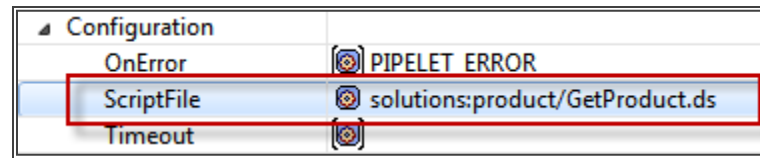


The Script Editor allows you to hover over the warning or error icon to get more precise information about the offending usage:



### Scripts and Cartridge Path Relationship

Although script pipelets can reference scripts in the same cartridge or from dependent cartridges, scripts are NOT searched using the cartridge path, unlike pipelines. A script is expected to be in the cartridge of the current executing pipeline unless the cartridge is explicitly stated in the `ScriptFile` configuration property. In the figure below, the `solutions` cartridge, `product` directory (under `scripts`), `GetProduct.ds` script will be used:



Of course, this means that you will need to add the `solutions` cartridge to the cartridge path so the script is found. In which order the `solutions` cartridge appears in the path is not relevant from the point of view of script access. However, the order of cartridges is relevant for executing pipelines and templates, in which case the first matching file found is used.

### The ScriptLog Output

Every script pipelet used in a pipeline comes with a default Dictionary Output property called the `ScriptLog`, of type `String`:

Configuration	
OnError	PIPELET_ERROR
ScriptFile	test/test.ds
Timeout	
Transactional	false
Dictionary Output	
ScriptLog	null

You can write to the `ScriptLog` output within your script by using the `TopLevel.global.trace(msg : String , params : Object ...)` method which uses `Java MessageFormat`. While this capability exists in the platform as a way to write debug or error messages to the pipeline, its use is not

recommended at this time. We suggest you use the `dw.system.Logger` API to write to log files instead. This API is covered later.



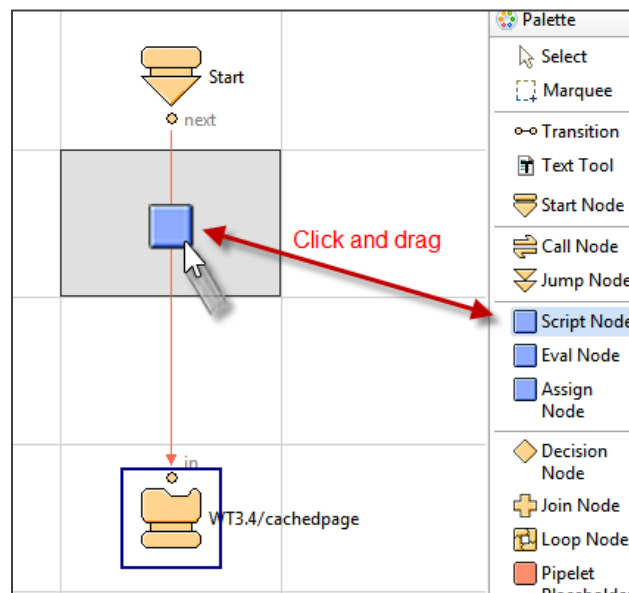
### Run the Create a Script Pipelet activity.

Demonstrate how to create a new script pipelet. Explain the commenting-out of input and output parameters and the `PIPELET_NEXT` and `PIPELET_ERROR` commands.



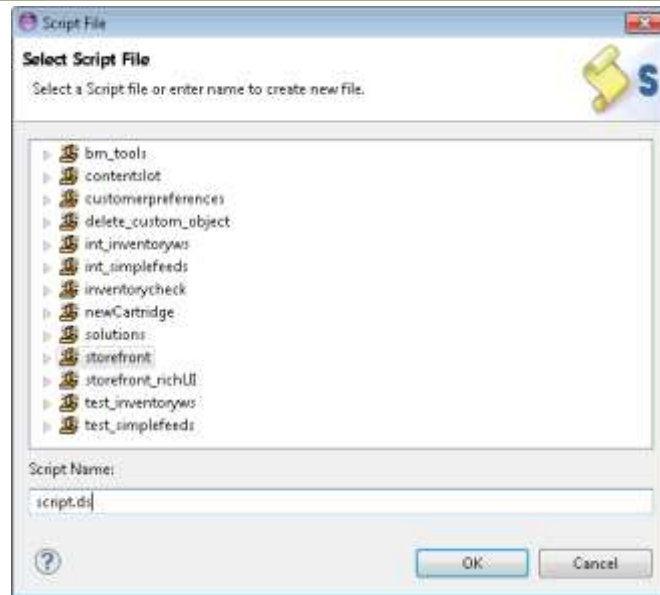
### To create a new script pipelet, follow these steps:

1. Open the pipeline you wish to add the script pipelet to.
2. Drag a new script node onto the workspace over the transition node where you want your script to execute. Be sure the transition node turns red before releasing your mouse over the transition node.



3. In the **Script File** window, select the cartridge in which you want to store your new script file.
4. Type `<scriptname.ds>` or `<directory/scriptname.ds>` in the **Script Name** field:





5. Click OK.
6. To begin editing the script file, double-click on the new script node. The script file will open up in the workspace using a default script code template:

```
/**
 * Demandware Script File
 * To define input and output parameters, create entries of the form:
 *
 * @<paramUsageType> <paramName> : <paramDataType> [<paramComment>]
 *
 * where
 * <paramUsageType> can be either 'input' or 'output'
 * <paramName> can be any valid parameter name
 * <paramDataType> identifies the type of the parameter
 * <paramComment> is an optional comment
 *
 * For example:
 *
 *-   @input ExampleIn : String This is a sample comment.
 *-   @output ExampleOut : Number
 *
 */
importPackage( dw.system );

function execute( args : PipelineDictionary ) : Number
{
    // read pipeline dictionary input parameter
    // ... = args.ExampleIn;

    // insert business logic here

    // write pipeline dictionary output parameter

    // args.ExampleOut = ...

    return PIPELET_NEXT;
}
```

This script code template can be found in UX Studio under **Window → Preferences → Demandware UX Studio → Generation Templates**. Here you can customize both script and ISML generation templates to suit coding guidelines at your project.



## 7.2. Exercise: Creating a Script Pipelet

1. In the ShowProduct pipeline, remove the GetProduct Demandware pipelet.
2. Drag a Script Node from the palette onto the same location as the removed pipelet, and complete the dialog as follows:
  - a. Select your cartridge to store the script.
  - b. For Script Name, enter `product/GetProduct.ds`.
3. Connect the new script pipelet to the start node and the interaction nodes the same as before.
4. Double-click the **script** node (a.k.a. script pipelet) to enter the Script Editor.
5. Modify the generated script code as follows or copy the code from the `GetProduct.ds` file in the solutions cartridge:
  - a. **Input** parameter `ProductID` of type `String`
  - b. **Output** parameter `Product` of type `Object`
  - c. Import the `dw.system` and `dw.catalog` classes.
  - d. Use the `ProductMgr.getProduct(args.ProductID)` method to obtain the product, and store it in the `args.Product` variable.
  - e. If the product is not found (`args.Product` is null), write a message to the `ScriptLog`:
    - i. `trace("The product {0} was not found", args.ProductID);`
    - ii. `Return PIPELET_ERROR.`
  - f. If the product exists, return `PIPELET_NEXT`.
6. Connect the input and outputs of the script pipelet to pipeline dictionary variables by using the pipelet properties view as shown:

Timeout	
Transactional	false
<input checked="" type="checkbox"/> Dictionary Input	
ProductID	CurrentHttpParameterMap.pid.stringValue
<input checked="" type="checkbox"/> Dictionary Output	
Product	myProduct
ScriptLog	Log
<input checked="" type="checkbox"/> Properties	
Custom Label	

7. Run the pipeline with a valid product in the query string: `ShowProduct-Start?pid=P0048`.
8. Verify that the product name appears as before.
9. Modify your `productnotfound` template so that it displays the contents of the `pdict.Log`.

# Script Debugging

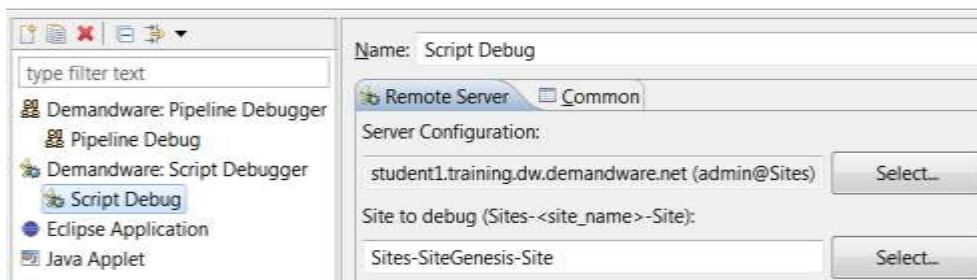


## Introduction

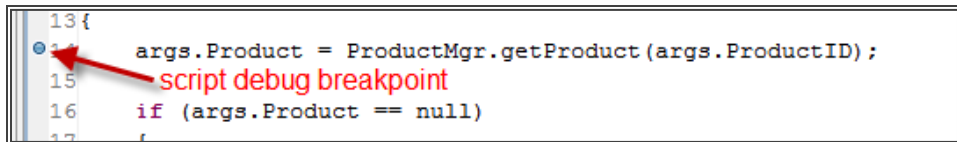
Studio gives you the ability to debug scripts as well as pipelines. In order to use a script debugger you must first create a script debug configuration. The process for creating a script debug configuration is identical to the pipeline debug configuration setup:

Create, manage, and run configurations

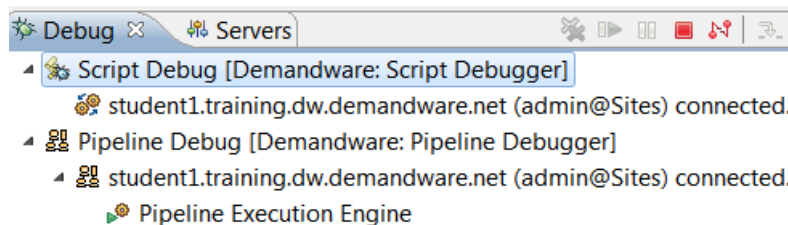
Create a configuration that will launch the Script Debugger.



In order to use the debug configuration you will need to add breakpoints in your script files:



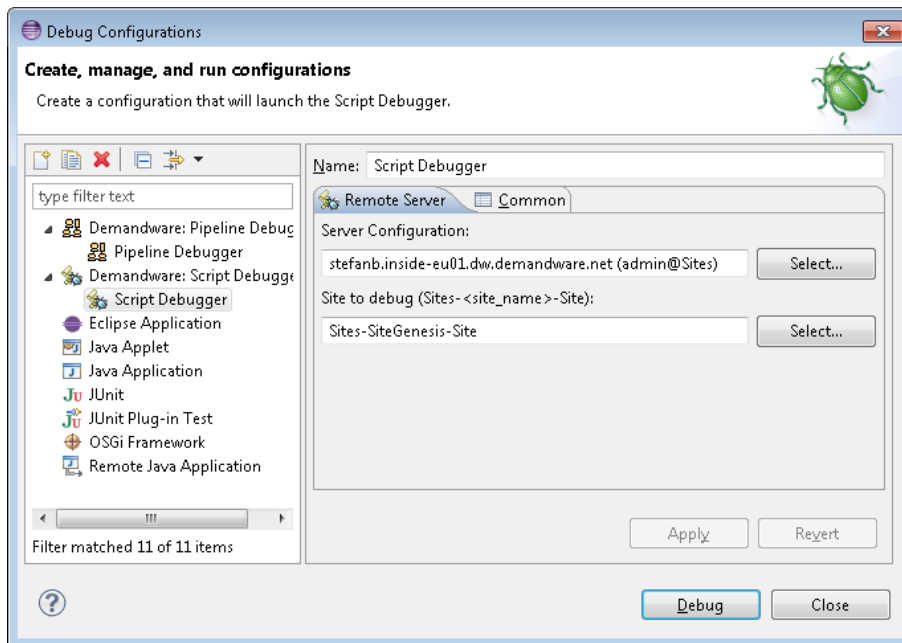
When debugging, it is possible to run the script debugger along with the pipeline debugger.





### 7.3. Exercise: Creating a Script Debug Configuration

1. In 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 will show as 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:



The screenshot shows the 'Variables' window in Visual Studio Code. It has two tabs: 'Variables' and 'Breakpoints'. The 'Variables' tab is active. It displays a table with two columns: 'Name' and 'Value'. The 'args' variable is expanded, showing its contents. The 'args' variable itself has a value of '[PipelineDictionary id=30646159]'. It contains two properties: 'Product' with a value of 'null' and 'ProductID' with a value of '54399'.

Name	Value
args	[PipelineDictionary id=30646159]
Product	null
ProductID	54399

- b. Use **F5** to execute the line.
  - c. Study the `args.Product` output variable: it should not be `null`:
- 
- The screenshot shows the 'Variables' window in Visual Studio Code. It has two tabs: 'Variables' and 'Breakpoints'. The 'Variables' tab is active. It displays a table with two columns: 'Name' and 'Value'. The 'args' variable is expanded, showing its contents. The 'args' variable itself has a value of '[PipelineDictionary id=30646159]'. It contains two properties: 'Product' with a value of '[Product sku=54399]' and 'ProductID' with a value of '54399'.
- | Name      | Value                            |
|-----------|----------------------------------|
| args      | [PipelineDictionary id=30646159] |
| Product   | [Product sku=54399]              |
| ProductID | 54399                            |
- d. Execute through the end of the pipeline (**F8**): the product name should appear on 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.

# Resource API and Resource Bundles



## Introduction

In storefront code you want to 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.). Also, 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.

## IMPORTANT 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"`).

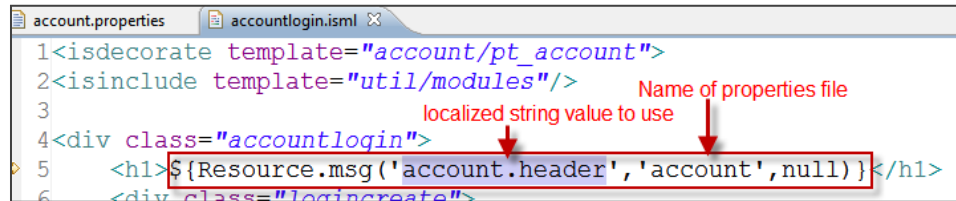
```

1account.header=My Account
2account.sendfriend=Send to a Friend
3account.sendlisttofriend=Send my List to a Friend
4
5#####
6# ISML Directory: account/
7#####
8accountoverview.welcome=Hello
9accountoverview.welcome2=, Welcome Back
10accountoverview.not=Are you not
11
12accountredirect.title=Redirect
13accountredirect.permanent=This account has been
14accountredirect.accountlogin=login page.
15account.forbidden=This page has timed out. You
16
17#####
18# ISML Directory: account/addressbook/
19#####
20editaddress.editaddress=Edit Address
21editaddress.addaddress=Add Address
22editaddress.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:



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 on 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('singleshopping.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**:

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



## Review & Lab



### 7.4. Exercise: Run the Demandware Script

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

1. Open `GetProduct.ds`
2. Inside the `trace` method use the Resource API to externalize the hardcoded message when the product is not found.
3. Add the corresponding externalized string in a resource bundle.

Demandware Script Review Questions	True	False
You do not need to fully qualify a folder path when importing a script into a script file		
You can modify the functions of a Demandware pipelet		
A script pipelet has full access to the Demandware Script API		



### Transition to Forms Framework

## 8.Forms Framework



---

### Goal

The purpose of this module is to describe the concepts and usage of the Demandware Forms framework as well as demonstrate how to implement a form in a pipeline.



---

### Time

2.5 Hrs



---

### Overview

We will study the following components of the forms framework: xml metadata file, isml templates that display a form, specific pipeline elements required for forms, and we will also demonstrate how to create a new form.



---

### Materials Needed

Solutions cartridge

---

# Overview



## Introduction

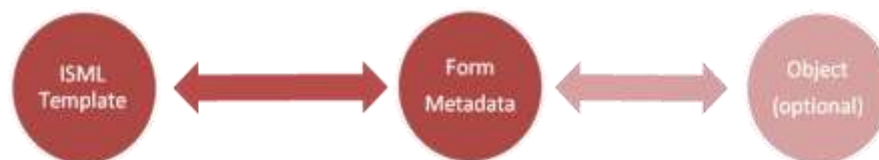
The Demandware platform provides a set of tools that help simplify form display and processing. Using the Demandware Forms framework, you can control how consumer-entered values are validated by the application, rendered on a browser and possibly stored on a server.

To utilize the Demandware forms framework, you need the following four files:

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

There are 3 objects that interact when working with Demandware forms:

1. 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.
2. ISML template: it uses the form metadata fields and actions to show an HTML form to the user.
3. 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.custom" />
  <field formid="address1" label="forms.custom" />
  <field formid="city" label="forms.city.cities" />
  <field formid="state" label="forms.state.states" />
  <field formid="zip" label="forms.customeraddress" />
  <field formid="country" label="forms.country" />
  <field formid="phone" label="forms.customeraddress" />
  <action formid="cancel" valid-form="false"/>
  <action formid="confirm" valid-form="true"/>
</form>
```

It is very easy to create this ISML template whose fields depend on the data from the form metadata:

First Name:	Patricia
Address Line 1:	120 Presidential Way Street Address, P.O. Box
City:	Woburn
State:	Massachusetts
Country:	United States
Phone:	781-756-3700 Format: XXX-XXX-XXXX
Indicates required field	
<a href="#">go back</a>	

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:

Manage Customers > 00000001 - Addresses > Manage Address

Create new Address

Fields with a red asterisk (\*) are mandatory.

Standard Address

Address ID:	H-Add	?
Title:		?
Company:	Demandware	?
Salutation:	Dear	?
First Name:	Patricia	?
Second Name:		?
Last Name:	Miller	?
Suffix:		?
Address 1:	10 Presidential Way	?
Address 2:		?
Suite No:		?
Post Box:		?
City:	Woburn	?
Postal Code:	01801	?
Country:	US (United States)	?
State:	MA	?
Contact Phone:	781-756-3700	?

Apply Reset Cancel

# XML Metadata File



## Introduction

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

The `field` element may use the following attributes:

Attributes	Description
<b>formid</b>	Required: unique ID to identify the field for ISML templates and pipelines.
<b>type</b>	Required: data type for field (see table below).
<b>label</b>	Usually a key to an externalized string in the <code>forms.properties</code> resource bundle.
<b>description</b>	Description for field, might be used in tooltips.
<b>min-length, max-length</b>	Restricts the field length for data entry.

<b>min,max</b>	Valid range for integer, number and dates.
<b>range-error</b>	Message shown if value provided does not fall within the specified range.
<b>regexp</b>	Regular expression for string fields: email, phone, zipcode, etc.
<b>parse-error</b>	Message shown when the data entered does not match the regex. Usually a key to an externalized string.
<b>mandatory</b>	Field is required via server side validation when true.
<b>missing-error</b>	Message shown if the primary key validation error is generated in a pipeline.
<b>value-error</b>	shown if a primary key validation error is generated in a pipeline.
<b>binding</b>	Used to match <code>field</code> to a persistent object attribute.
<b>masked</b>	Specify # of characters to mask.
<b>format</b>	Format for display of dates, numbers, etc.
<b>whitespace</b>	Specify whitespace handling (none or remove).
<b>timezoned</b>	Optional flag for date objects (true or false).
<b>default-value</b>	Pre-defines a value for a field.
<b>checked-value</b>	Value when field is checked in a form.
<b>unchecked-value</b>	Value when field is unchecked in form.

Field types can be as follows:

Field type	Description
<b>string</b>	Use for text data.
<b>integer</b>	Use for numeric data like days, months.
<b>number</b>	Use for quantity fields.
<b>boolean</b>	Use with multiple-choice fields.
<b>date</b>	Use this when <code>timezoned</code> or <code>format</code> 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 the example above, 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.

Finally, 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.

### IMPORTANT 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.

# ISML Form Template



## Introduction

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

```
<form>...</form>
```

You can choose to 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"
name="SendToFriendForm" id="SendToFriendForm">
```

The method `dw.web.URLUtils.continueURL()` ensures that the form gets submitted back to the Interaction Continue Node 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, you 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 below:

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.

# Form Pipeline Elements



## Overview

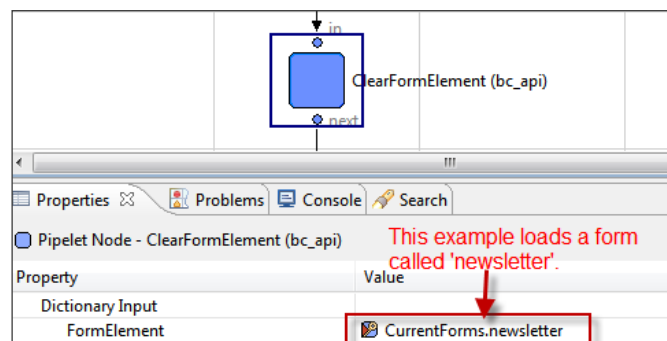
A pipeline that uses the Demandware forms framework has a very distinctive pattern because it uses the following elements:

- ClearFormElement pipelet to create a form object in the pdict from the 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.

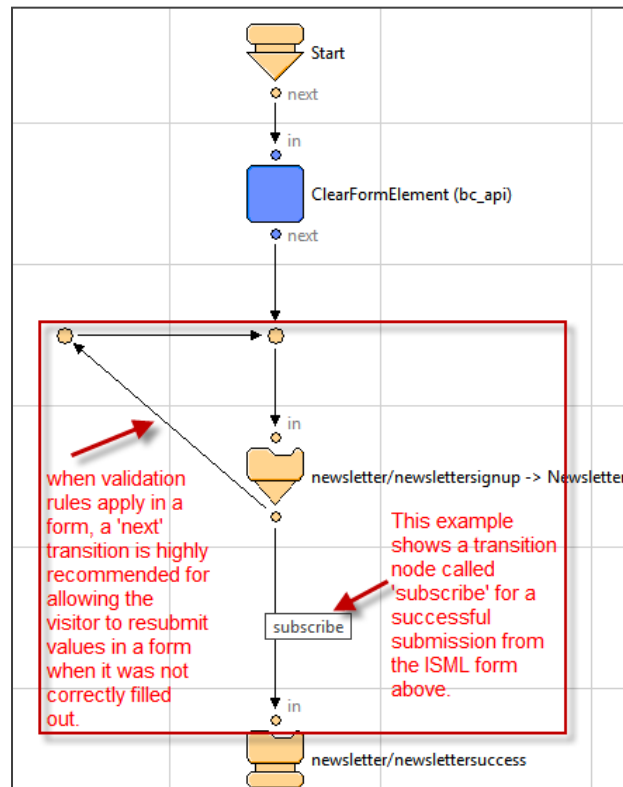


To create a form using the form framework, follow these steps:

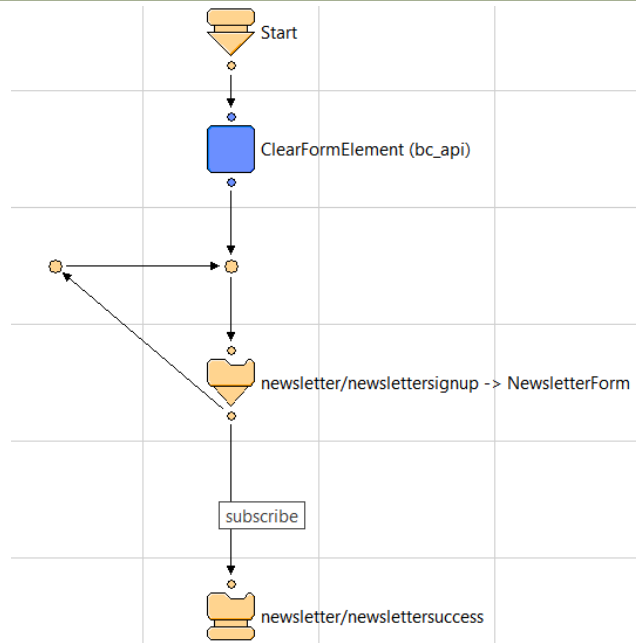
1. Create an xml metadata file that will hold your form data.
2. Create an ISML template that will display a form to a visitor.
3. Create a pipeline that includes at minimum:
  - a. **Start** node
  - b. ClearFormElement pipelet that loads the xml form you created in step 1 into the pdict.



- c. Interaction Continue node that links to the ISML template you created in step 2.
- d. Transition nodes to handle the following scenarios:
  - Continues the pipeline after the ISML form has been successfully submitted by the visitor
  - Sends the pipeline back to the form to be resubmitted if there are validation rules which fail the successful submission.



- e. The pipeline at minimum should look similar to the example below:

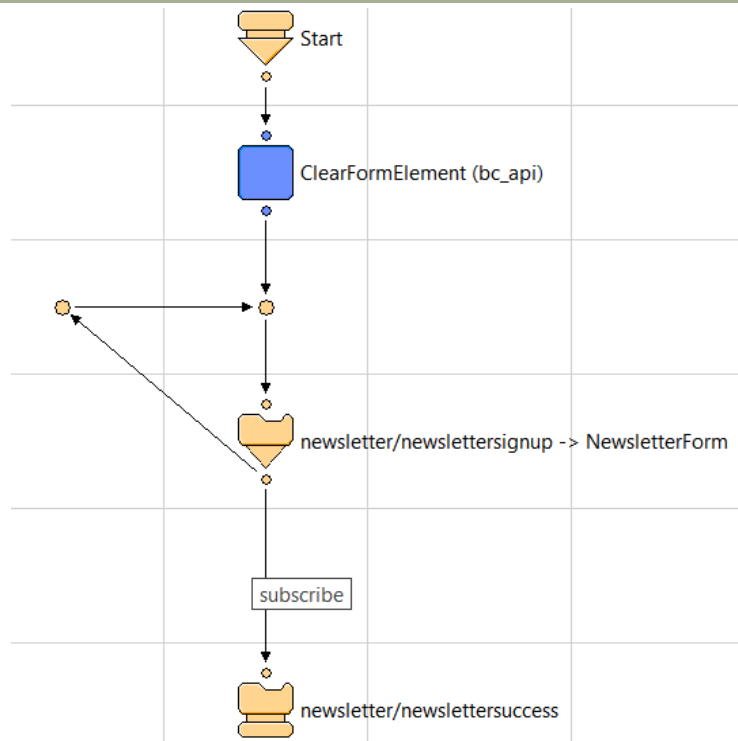


## 8.1. Exercise: Using Forms Framework

The goal of this lab is to create a form to capture newsletter subscription information. In this exercise the data will not be saved to the database.

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.
  - c. Study `forms.properties` in the storefront cartridge: this resource file contains the localized strings used in the form metadata file.
2. 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.

3. 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”.
  - b. Save `locale.properties` from the solutions cartridge into your cartridge: this file contains the externalized strings used in `newslettersignup.isml` and `newslettersuccess.isml`
4. Create a pipeline that displays the Newsletter Subscription form:
  - a. Create a new pipeline called `Newsletter- Start`.
  - b. After the start node, drag a `ClearFormElement` pipelet that clears the `CurrentForms.newsletter` form and loads it into the `pdict`. Check its properties!
  - c. Next, create a transition from the pipelet to a new interaction continue node.
  - d. Give the ICN (Interaction Continue Node) these properties:
    - Start Name: `NewsletterForm`.
    - Template: `newslettersignup.isml`.
  - e. Create a transition from the interaction continue node to a new interaction node:
    - Name the connector between the two nodes as `subscribe`.
    - Interaction node displays `newslettersuccess.isml`
5. Test the `Newsletter-Start` pipeline:
  - a. Test with correct data.
  - b. Test with a malformed email (missing the “@” sign): *An error occurred*.
6. Handle validation errors in the form submission:
  - a. Drag a join node to the left of the interaction continue node.
  - b. Pull a transition from the join node to the transition between the pipelet and ICN.
  - c. Check the name of this transition: it should be named `next`.



## Review & Lab



### 8.2. Exercise: Create form meta-data by yourself

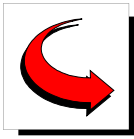
In this lab you need to capture customer interests related to categories and products in the site. For now you will just create the form interaction, later you will store this data in the profile system object.

1. Copy `cartridges/forms/default/newsletter.xml` from the training cartridge to `preferences.xml` in the same location. You will be modifying this form metadata file that captures marketing and personal information from a registered visitor. The modification has to be done as per the following table:

formid	label	Data type	binding	Externalized Strings
interestApparel	forms.interestedinApparel	boolean	custom.interestApparel	Are you interested in Apparel ?
interestElectronics	forms.interestedinElectronics	boolean	custom.interestElectronics	Are you interested in Electronics ?
newsletter	forms.interestedinNewsletter	boolean	custom.newsletter	Are you interested in Newsletter ?

- a. None of the choices is mandatory
  - b. Add an `apply` action that does not require validation.
2. We will not do anything with this metadata until a later exercise.

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



---

### Transition to Custom Objects

---



## 9. Custom Objects



---

### Goal

The goal of this module is to use custom objects and transactional pipelets to store data that does not naturally belong in any other Demandware system object. An additional goal is the use of defensive coding and custom logging to improve the quality of the code.

---



---

### Time

2 Hrs

---



---

### Overview

We will define custom objects and create instances programmatically. We will use a transactional pipelet to save the custom object in the database, and implement custom logging to allow debugging and error messages to be written to logs.

---



---

### Materials Needed

The following cartridges are needed to run this module: storefront and solutions

---

# Defining Custom Objects



## Introduction

In the previous lesson, you created a simple form using an Interaction Continue Node. You validated the data being submitted, but did not store the data permanently. In this lesson, you will learn how you can store data in a custom object so it can be persistent.

Custom objects (COs) 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.

## IMPORTANT NOTE

You should always consider first if a Demandware System object (Product, Catalog, etc) can be used instead of creating a custom object. Please do not abuse the fact that 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 COs. Demandware Platform Governance has quotas around custom object API usage and data size which will be enforced in the future.

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

- Custom Object Definitions: allows naming, primary key and column specification. Located under **Administration ⇒ Site Development**
- Custom Object Editor: allows instance creation and editing. Located under **Site - <site> ⇒ Custom Objects ⇒ Custom Object Editor**.

When defining the CO you specify the storage scope of the instances: site or organization. Organization custom objects can be used by any site, whereas site custom objects are created by one site and cannot be read by another. The CO type itself is always available to the entire organization. Also, you can specify if you want CO instances to be replicable: this means they can be copied from Staging to Production as part of the replication process.

An example of CO usage that we will cover in this course is a newsletter that customers can sign up for and which the platform does not have a system table for. 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. In order to allow anyone to get a newsletter we need to define a CO. This CO should not be replicable, since subscriptions created in Staging should not be copied to Production. You also need to consider how to clean up COs 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.

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

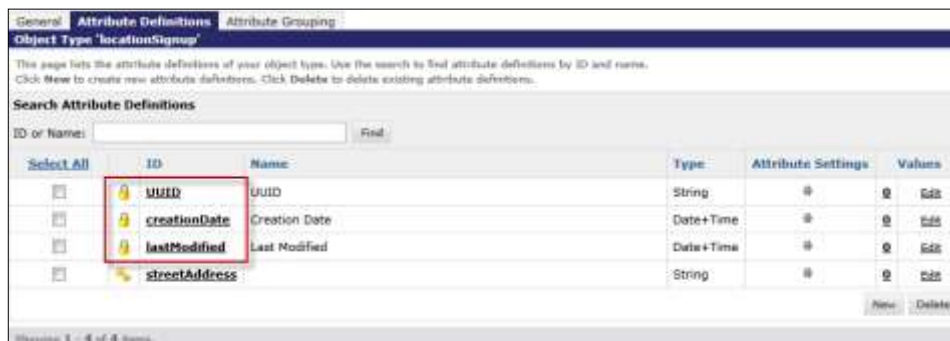
You can create your custom objects in Business Manager, or you can do it programmatically. Before you can create a custom object instance you must first define the custom object data type in Business Manager.



## 9.1. How-To : Create Custom Object Types

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. Spaces in the name are not allowed.
  - 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. You will notice default values created with your custom object type. These values cannot be changed once they are created.



7. Create the attributes (values you wish to capture in the table) by clicking the **'New'** button.
8. Specify a unique name in the ID field and then select the type of data being entered in the attribute from the **Value Type** drop-down menu:

9. Click the **Apply** button.
10. When you are finished, click the '**Back**' button to add another attribute.
11. When you are finished adding attribute definitions, you will need to create an Attribute Group. Click the '**Attribute Grouping**' tab.

12. Enter a name for your grouping in the **ID** field and a name in the **Name:** field.

13. Click the **Add** button. You will need to add the field attributes next.
14. Click the **Edit** link to add field attributes to the group.

15. Click the ellipse next to the **ID:** field to select field attributes.

16. Select the attributes you wish to add from the list by clicking in the checkbox next to each one. Then click the 'Select' button.

**Select Object Type Attribute**

Select the attributes you want from the list below by clicking the attribute ID or name. You can close this window without selecting an attribute by clicking the Cancel button.

**Search Attribute Definitions**

ID or Name:  Find

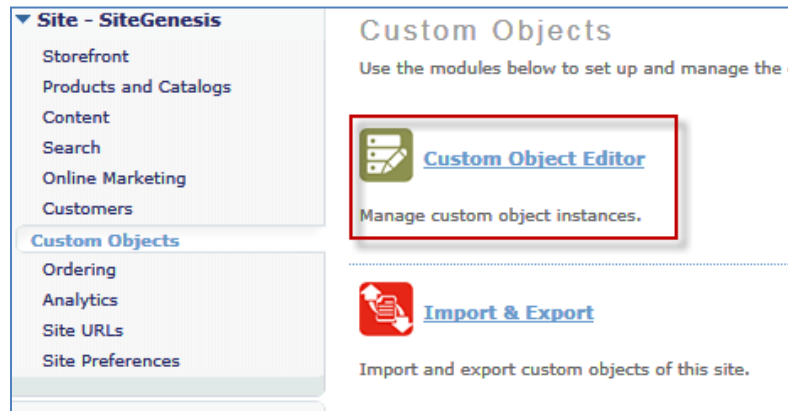
Select All	ID	Attribute Name	Type	Attribute Settings
<input type="checkbox"/>	city	City	String	
<input type="checkbox"/>	creationDate	Creation Date	Date+Time	#
<input type="checkbox"/>	lastModified	Last Modified	Date+Time	#
<input type="checkbox"/>	state	State	String	
<input type="checkbox"/>	streetAddress		String	#

Select Cancel

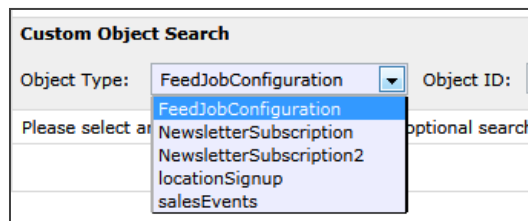
17. You are now ready to view, add, and edit new instances of the custom object type you just created in the **Custom Object Editor** section. We will next give you the steps for creating new custom objects manually.

### To create a custom object instance manually in Business Manager:

1. Log into Business Manager.
2. Click the site you wish to manage the custom objects for.
3. Click **Custom Objects**→**Custom Object Editor**.



4. At the next window, select the custom object type you wish to manage from the drop-down list.



5. To create a new custom object click the **New** button.



6. Enter data in each of the required fields and then click the **Apply** button.

A screenshot of the 'Local Signup' form. It contains three input fields: 'streetAddress:' (with a red asterisk indicating it is required), 'State:', and 'City:'. Each field has a corresponding text input box.

7. You have now created a custom object. Click the **Back** button to exit the custom object editor.



### 9.2. Exercise: Create a Custom Object Definition

In this exercise, you will define a custom object that will store the data that your Newsletter form gathers from the customer.

1. In Business Manager navigate to **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 is up to 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 CO:
  - a. Name it Presentation.
  - b. Includes attributes firstName, lastName and email.
5. Using **Site - SiteGenesis ⇒ Custom Objects ⇒ Custom Object Editor**, find the new NewsletterSubscription type and manually enter a new subscription.



# Using Script to create Custom Object Instances



## Using Demandware Script to Create Custom Objects

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

- **CustomAttributes**: attributes defined by a user in the Business Manager to extend a system object or CO. Accessible via the syntax:  
`co_instance.custom.attribute`
- **CustomObject**: represents an instance of a CO
- **CustomObjectMgr**: allows the creation of CO instances
- **PersistentObject**: allows persistent storage
- **ExtensibleObject**: allows 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 COs are persisted in the database and can have custom attributes added by an administrator or programmer 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 allows creation of an instance of a CO by providing the CO 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 CO is created. Otherwise, a duplicate PK error will occur.



## Implicit Database Transaction Handling

Database transaction handling in Demandware is handled one of two ways:

- 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 (covered in the next section).

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:

The screenshot shows a pipelet node in a pipeline, labeled "Script (bc\_api)" with the script file "newsletter/SignUpNewsletterLab5.1.ds". The node has a black "T" on it, indicating it is transactional. Below the node is the "Properties" window for the "Pipelet Node - Script (bc\_api)".

Property	Value
Configuration	
OnError	PIPELET_ERROR
ScriptFile	newsletter/SignUpNewsletterLab5.1.ds
Timeout	
Transactional	true

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.



### Run the Create Custom Object via Script activity.

Demonstrate how to use the Demandware Script API to create a custom object instance.



### 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 : CustomObject =
        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`.



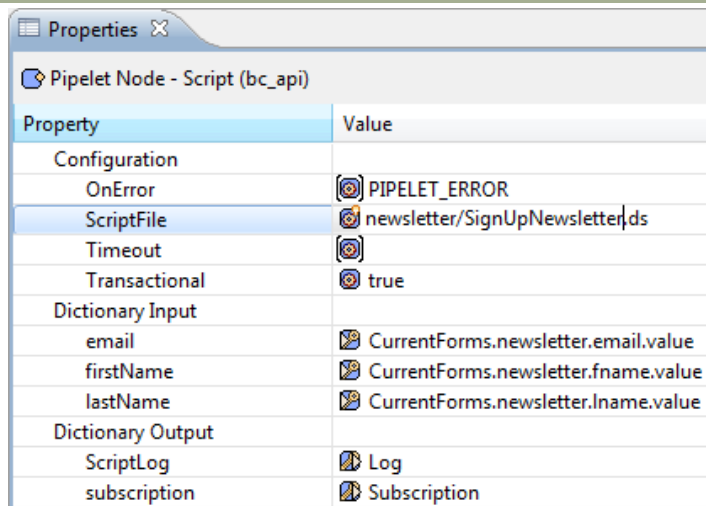
### 9.3. Exercise: Create a Custom Object Instance programmatically

In this exercise, you will use the form you created in the previous module to create a new custom object programmatically.

#### Create a Script Pipelet to Create Newsletter Subscription

1. Open the `Newsletter-Start` pipeline.
2. Drop a new script node after the interaction continue node (connected to the `subscribe` transition).
3. The node uses a new `newsletter/SignUpNewsletter.ds` script. In the script, do the following:
  - a. Declare input parameters `firstName`, `lastName`, `email` of type `String`.
  - b. Declare output parameter `subscription` of type `Object`.
  - c. Create a new `CustomObject` instance of type `NewsletterSubscription` using `args.email` as the primary key.
  - d. Store input parameters into `CustomObject` instance fields. Hint: use the `custom` keyword to assign values to custom attributes:
 

```
co.custom.firstName = args.firstName;
```
  - e. Return the CO in the output parameter `subscription`.
  - f. Return `PIPELET_NEXT`.
4. Edit the properties of the pipelet:
  - a. Make it **Transactional** so it commits to the database.
  - b. Assign all the corresponding form values to the pipelet inputs.
  - c. Assign the subscription output to a `Subscription` object.
  - d. Verify that all properties are correctly defined:



Property	Value
Configuration	
OnError	PIPELET_ERROR
ScriptFile	newsletter/SignUpNewsletter.ds
Timeout	
Transactional	true
Dictionary Input	
email	CurrentForms.newsletter.email.value
firstName	CurrentForms.newsletter.fname.value
lastName	CurrentForms.newsletter.lname.value
Dictionary Output	
ScriptLog	Log
subscription	Subscription

- For the case when the email subscription already exists, connect the error exit of the script node to an interaction node holding a template stating :

```
<h4>An error occurred in your subscription. Maybe the email
${pdict.CurrentForms.newsletter.email.value} already exists ?.</h4>
Back to <a href="${URLUtils.url('Newsletter-Start')}">Newsletter
Subscription</a>.
```

### Show Confirmation to the User

- Modify the subscription confirmation template to use the Subscription object from the pdict:

```
${pdict.Subscription.custom.firstName}
```

- Test the Newsletter-Start pipeline in the storefront.
- Verify that the CO instance is created in Business Manager.

### Troubleshooting:

- Check to see if you imported `dw.object` package
- Check to see if you have marked the script node as Transactional
- Re-check the configuration of the script in Step 4d.
- Check if the NewsletterSubscription Custom Object type exists in the Business Manager ('N' capital, 'l' small, 'S' capital).
- Also check if string attributes "firstName", "lastName" and "email" exist in it and are a part of an Attribute group.

# Custom Logging



## Introduction

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.

As for categories, the programmer can define as many levels of categories and subcategories as needed. Demandware does not impose a certain categorization; you decide how you want to organize your logging categories. For example:

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

If logging is enabled for a category (say product), all its subcategories will also be enabled.

## Examples

If Warn logging is enabled for "product": Warn, Error and Fatal errors are logged for "product" and all its sub-categories.

If Warn logging is enabled for "product" and Debug for "product.import":

- 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 will need to use the `dw.system.Logger.getLogger()` factory method. This method creates a Logger object for a specified category:

## Instructor Guide

```
var logger : Logger = Logger.getLogger( "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;
}
```

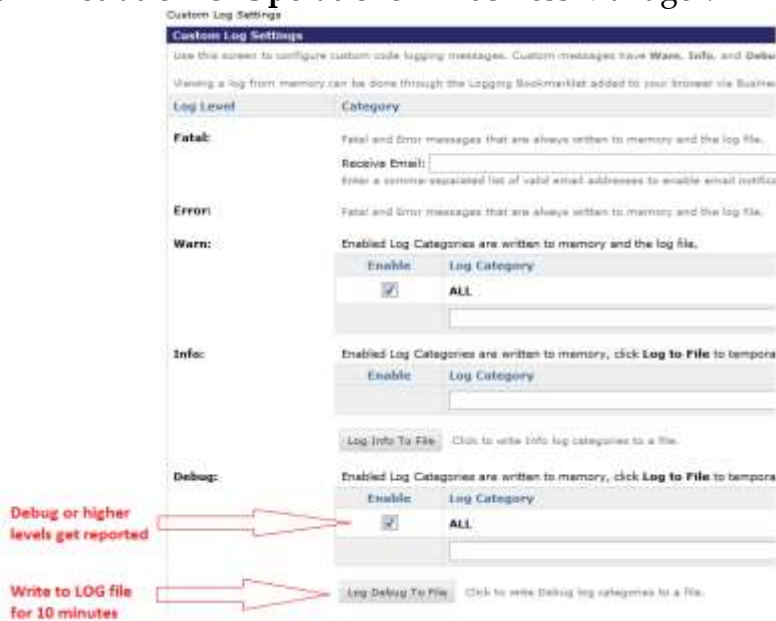
You can use the Logger object to write a message for a specific severity level:

`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 will need to enable **Custom Log Settings** under **Administration → Operations** in Business Manager:





### Run the Enable Custom Logging activity.

Demonstrate how to enable custom logging in Business Manager. Enable 'Log to File'. Show the students where to view their custom logs.



To enable Custom Logging in Business Manager, follow these steps:

1. Log into Business Manager.
2. Click **Administration**→**Operations**→**Custom Log Settings**.
3. Create a log category: type it on the field under a given severity, then click **Add**.



4. **Enable** the checkbox next to the log category you wish to write to.
5. Click the **Apply** button.
6. Click **Log Debug to File** to allow debug messages to be written to a log for the next 10 minutes. Usually Debug and Info messages get written to memory only, and are visible via the Request Log tool.
7. Run the pipeline you wish to debug.
8. Review the custom log file in Business Manager by clicking **Administration**→**Site Development**→**Development Setup**→**Log Files**.
9. Open the log file that was just created. Search for the file by date. The custom log file will be named something like **customdebug-177.aaq.demandware.net-appserverxxx.log**.



## 9.4. Exercise: Custom Logging

1. You will 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.
2. Modify Pipelet to Write to Debug Log
  - a. Open the newsletter/SignUpNewsletter.ds script.
  - b. Use the `dw.system.Logger` API to write debug messages.
  - c. Use the following code as a guide to create the try-catch block around the code you wrote in the previous lab:

```
//Write messages to the log
var logger : Logger = Logger.getLogger( "newsletter" );
logger.debug( "Input params firstName: {0} lastName: {1} email: {2}",
args.firstName, args.lastName, args.email);

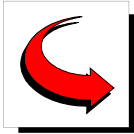
//Instantiate the NewsletterSubscription custom object
try
{
    ... code from previous lab ...
}
catch (e)
{
    logger.error("A newsletter subscription for this email address",
    " already exists: {0}", e.causeMessage );
    return PIPELET_ERROR;
}
```

3. Enable Logging for Debug Messages
  - a. In Business Manager, navigate to **Administration ⇒ Operations ⇒ Custom Log Settings**.
  - b. Enter **All** in the Log Category field.
  - c. Click **Add** to enable debugging for **All** debug messages.
  - d. Click **Log Debug to File**.
  - e. Click **Apply** to save these changes.
4. Test your pipeline with a duplicate email address and verify the **latest customwarn** log files.
5. Navigate to **Administration ⇒ Site Development ⇒ Development Setup ⇒ Log Files**.
6. Verify the messages on the **customdebug** and **customerror** log files that appear with the most recent timestamp.

7. Verify the debug messages also appear on the request log.

## Review

Custom Object Questions	True	False
Custom objects are the only way to store custom data in Demandware		
The “custom” keyword is required to access attributes of a custom object		
Custom objects need primary keys		
Custom object instances can only be created in Business Manager		
Implicit transaction means that the pipelet always commits		



---

**Transition to Data Binding and Explicit Transactions**

---

## 10. Data Binding and Explicit Transactions



---

### Goal

The goal of this module is to use data binding to update persistent objects with form data and vice versa. Since this task often requires multiple steps, we will also cover the use of explicit transactions to accomplish a multi-step process.

---



---

### Time

2 Hrs

---



---

### Overview

We will discuss how to use `UpdateFormWithObject` and `UpdateObjectWithForm` pipelets to pre-populate forms and update the database after the form is submitted. Also, we will use an explicit transaction to commit changes to the database on a modified version of the Newsletter exercise.

---



---

### Materials Needed

The following cartridges are needed to run this module:  
`solutions`, `customerpreferences`

---

# Data Binding with Forms and Objects



## Introduction

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. In the following form metadata we are using `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>
```

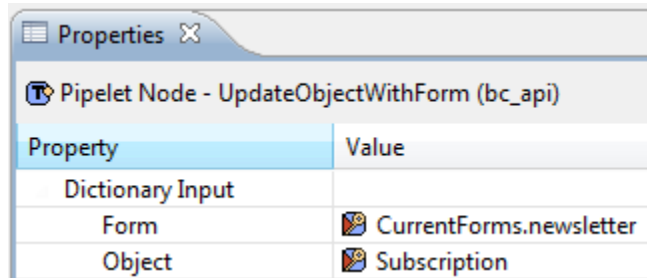
This is because the NewsletterSubscription CO we want to bind this form to have `firstName`, `lastName` and `email` fields which are all custom attributes, as seen below since the fields don't have a lock icon (they were added by you as custom attributes of the CO):

	ID	Name
	<u>UUID</u>	UUID
	<u>creationDate</u>	Creation Date
	<u>email</u>	
	<u>firstName</u>	First Name
	<u>lastModified</u>	Last Modified
	<u>lastName</u>	Last Name

### Using the UpdateObjectWithForm Pipelet

The `UpdateObjectWithForm` pipelet updates an existing persistent object with data from the form. It requires the object to update and the form object both available on the `pdict`. It is transactional by default since the object to be updated must be a persistent object.

Here is an example of how you define the properties of the pipelet using the newsletter form and `NewsletterSubscription` object we saw above:



The pipelet will inspect the `CurrentForms.newsletter` form in the `pdict`, and try to match every field with a binding attribute to a column in the object called `Subscription`. This object must be an instance of `NewsletterSubscription` that was placed in the `pdict` by either creating a new instance (using `CreateCustomObject` pipelet) or by retrieving an existing instance (using `SearchCustomObject` pipelet).

If the `Subscription` object is null, or not an instance of `NewsletterSubscription` CO, or the form is not in the `pdict`, the pipelet will fail and the transaction will be rolled back. If the pipelet is successful the transaction will commit. This pipelet is an excellent way to enter new data or update existing data on an object.



### 10.1. Exercise: Using `UpdateObjectWithForm` Pipelet

You will modify the `Newsletter` pipeline to remove the script node that creates the `NewsletterSubscription` CO and use Demandware pipelets instead to achieve the same behavior.

1. Create a CO using a Demandware pipelet:
  - a. Open the `Newsletter` pipeline.
  - b. Remove the Script pipelet that uses the `SignUpNewsletter.ds` script.
  - c. Add a new `CreateCustomObject` pipelet in the same place.
  - d. Specify the following properties:

Property	Value
Configuration	
CustomObjectType	NewsletterSubscription
Dictionary Input	
Key	CurrentForms.newsletter.email.value
Dictionary Output	
CustomObject	Subscription

2. Update the CO with data from the form:
  - a. Drag an `UpdateObjectWithForm` pipelet below the previous pipelet and connect it with a transition.
  - b. Specify the properties of the pipelet so the `CurrentForms.newsletter` form fields populate the fields in the `Subscription` object (see the screen shot in the previous page).
  - c. Make sure the `newsletter.xml` metadata has bindings that match the attribute keys in the CO.



### Using UpdateFormWithObject Pipelet

The UpdateFormWithObject pipelet updates a form with data from an object. It requires the form to update and the object to be both available on the `pdict`. It is not transactional since the updated form lives in the `pdict` scope, not in the database.

Notice that a form group may be updated with an object: as long as the bindings match, just that part of the form will be updated.

Properties	
Pipelet Node - UpdateFormWithObject (bc_api)	
Property	Value
Configuration	
Clear	false
Dictionary Input	
Form	CurrentForms.profile.customer
Object	CurrentCustomer.profile

In the example above, the `profile.xml` form has a customer group that will be updated with the existing profile data from the logged in customer:

```

Account  profile.xml
<?xml version="1.0"?>
<form xmlns="http://www.demandware.com/"
  <group formid="customer">
    <field formid="firstname" label=
    <field formid="lastname" label=
    <field formid="email" label="pr
    <field formid="emailconfirm" la

    <field formid="birthday" label=
    <field formid="phone" label="pr
    <field formid="addtoemaillist"
    <action formid="editprofile" va
  </group>

```



### 10.2. Exercise: Using `UpdateFormWithObject` Pipelet

In this exercise you will review the `Account-EditProfile` pipeline to verify how the `profile` form is populated with the existing values from the database before the form is shown to the user for editing.

1. Open the `Account-EditProfile`
2. Notice the 3 different usages of the `UpdateFormWithObject` pipelet.
3. Examine the properties of each pipelet and find:
  - a. The form metadata files being referenced
  - b. The system or custom objects that will bind to those fields
4. Create a login in the system, and put a breakpoint on this pipeline to examine what happens when you edit your personal data.



# Explicit Transaction Handling



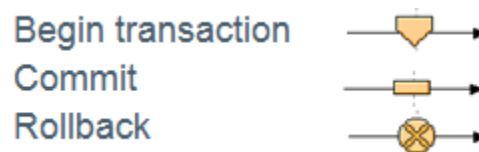
We already saw that transaction handling can be done implicitly by just executing a transactional pipelet: the commit or rollback is controlled by the PIPELET\_NEXT or PIPELET\_ERROR return values.

However, in some circumstances the transaction spans several pipelets or steps: in this case you need to decide where the transaction begins and ends. This mechanism is called Explicit Transaction Handling.

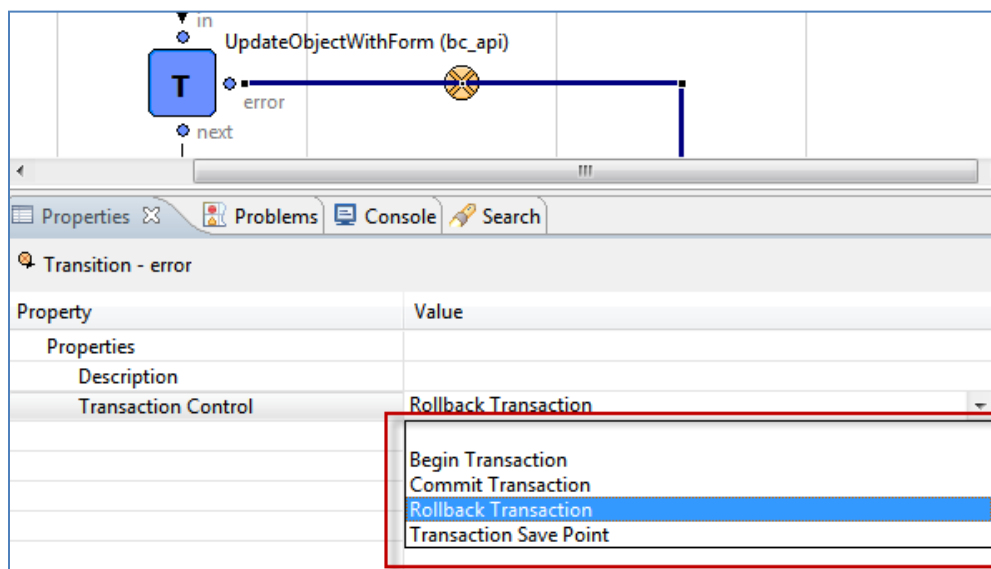
## Explicit Database Transaction Handling

This is implemented at the pipeline level by changing the Transaction Control property of a transition or connector.

You can:



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



This can be used to override the built-in implicit transaction in order to group changes that need to be an atomic transaction.

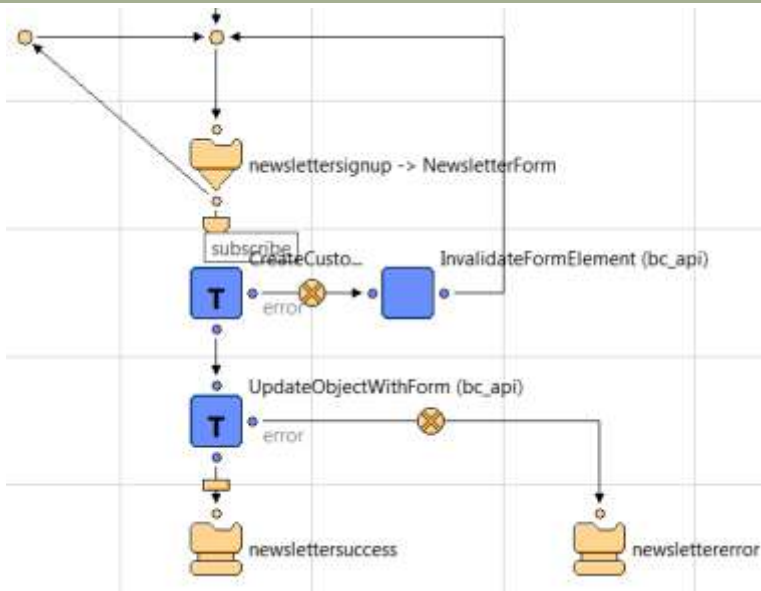


### 10.3. Exercise: Explicit Transaction Handling

You will modify the `Newsletter` pipeline to ensure that the pipelets that handle CO creation and updating are wrapped inside a transaction. Optionally, you will implement an error handler in case a duplicate email address is used to create the CO.

1. Use a transaction surrounding the create and update pipelets:
  - a. Open the `Newsletter` pipeline.
  - b. Change the subscribe transition to use a **Begin Transaction** control before the creation of the `Subscription` custom object.
  - c. Change the transition after the `UpdateObjectWithForm` pipelet to use a **Commit Transaction** control.
  - d. Change the error transition from the `UpdateObjectWithForm` pipelet to use a **Rollback Transaction** control.
2. Create an error handler for duplicate primary keys:
  - a. Add an `InvalidateFormElement` pipelet that invalidates the `CurrentForms.newsletter.email` element at the error exit of the `CreateCustomObject` pipelet.
  - b. Roll back the transaction at the error exit of the `CreateCustomObject` pipelet.
  - c. Connect the `InvalidateFormElement` with the join node above the ICN.
  - d. Make sure the `newsletter.xml` metadata file contains a value error definition:
 

```
value-error="forms.contactus.email.value-error"
```
  - e. Create an appropriate message (i.e. "email already exists") for the `forms.contactus.email.value-error` key in the `forms.properties` file.
  - f. The pipeline snippet looks like this:



3. Test the pipeline with a new subscription:
  - a. Use the debugger to see the contents of the Newsletter Subscription object as it gets created and updated.
  - b. Try creating the subscription with the same email to see the validation error generated.

## Review & Lab

Forms Framework Questions	True	False
To implement Explicit Transactions you use the <code>beginTransaction()</code> method in a script pipelet		
<code>UpdateObjectWithForm</code> can update any object even if it is not persistent		
<code>UpdateFormWithObject</code> will update any form fields where the binding matches the object attributes		



## 10.4. Exercise: Retrieve information from Custom Object and store it

In this lab you create a new `EditPreferences` pipeline 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.

1. Extend the `Profile System Object`
  - a. In the Business Manager, extend the `Profile` system object with the following custom attributes:
    - ♦ `interestApparel : Boolean`
    - ♦ `interestElectronics : Boolean`
    - ♦ `newsletter : Boolean`
  - b. None of the attributes is 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 item that calls the `EditPreferences` pipeline. Use the `$httpsUrl (EditPreferences-Start) $` content link function to invoke your pipeline (this syntax was covered in the Content Slot module):

```
<li>
<a href="$httpsUrl (EditPreferences-Start) $"
    title="View and modify your preferences">

    My Preferences</a>
<p>View and modify your Preferences</p>
</li>
```

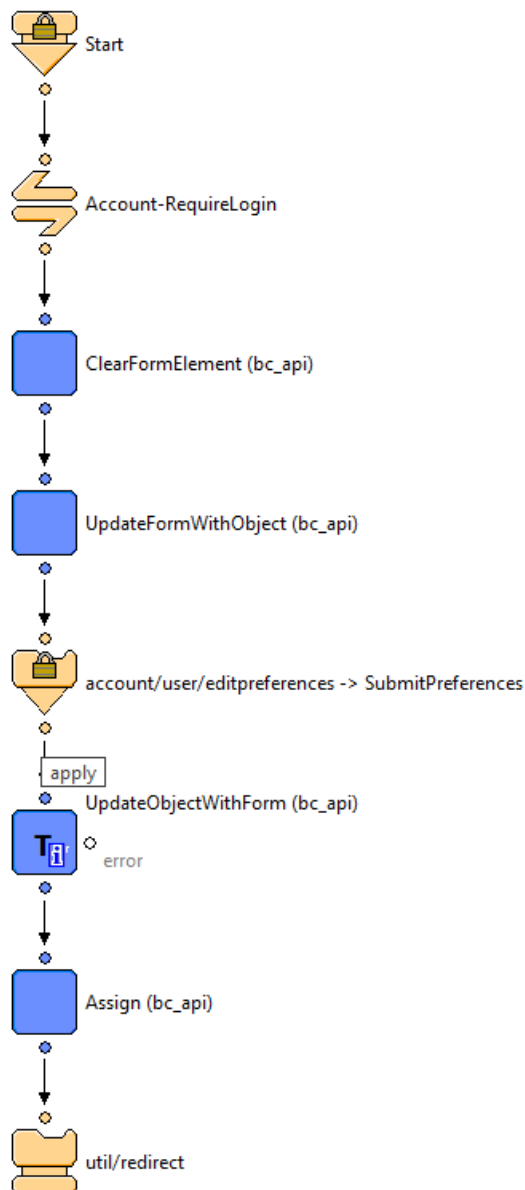
3. Edit the Form Metadata to add bindings
  - a. Open the `preferences.xml` form metadata file.

- b. 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. Create a Pipeline for Entering Preferences

- a. Create an `EditPreferences` pipeline.
- b. The start node should require a secure connection.
- c. Add a **call** node, pointing to the `Account-RequireLogin` pipeline, to make sure the user is forced to login before executing the rest of the pipeline.
- d. Add a `ClearFormElement` pipelet which clears the `CurrentForms.preferences` `FormElement`.
- e. Add an `UpdateFormWithObject` pipelet. Define properties as:
  - ♦ `Form`: `CurrentForms.preferences`
  - ♦ `Object`: `CurrentCustomer.profile`
  - ♦ Optionally you can switch the `Configuration` → `Clear` setting to `true` instead of adding the previous `ClearFormElement` pipelet.
- f. Copy the `editpreferences.isml` from the `customerpreferences` cartridge to your own. **Make sure the formfields are matching the formids of the `preferences.xml` metadata file.**
- g. Add an ICN with secure connection required. Make it point to `editpreferences.isml`. Start name "`SubmitPreferences`".
- h. Add an `UpdateObjectWithForm` pipelet after the `apply` transition with the following properties:
  - ♦ `Form`: `CurrentForms.preferences`
  - ♦ `Object`: `CurrentCustomer.profile`
- i. Add an `Assign` pipelet that maps the first expression (in "`From_0`") `dw.web.URLUtils.https('Account-Show')` to the first variable called `Location` (in "`To_0`"). This will redirect you to the `Account-Show` pipeline after pressing the **apply** button.
- j. End the pipeline with an **interaction** node that invokes the `util/redirect` template (in the `storefront` cartridge).
- k. Verify that the pipeline looks as follows:



5. Test in storefront by logging in first, then entering your preferences. Go back into the form again and check that your original preferences were saved.

# 11. Data Integration



---

## Goal

The purpose and goal of this module is to understand how external data is integrated into the Demandware Platform.

---



---

## Time

3 Hrs

---



---

## Overview

We will discuss how batch feeds (asynchronous integrations) are configured, and how Web Services are used for synchronous integration.

---



---

## Materials Needed

The following cartridges are needed to run this module:  
`delete_custom_object, int_simplefeeds,`  
`test_simplefeeds, solutions`

---



# Data Integration Overview



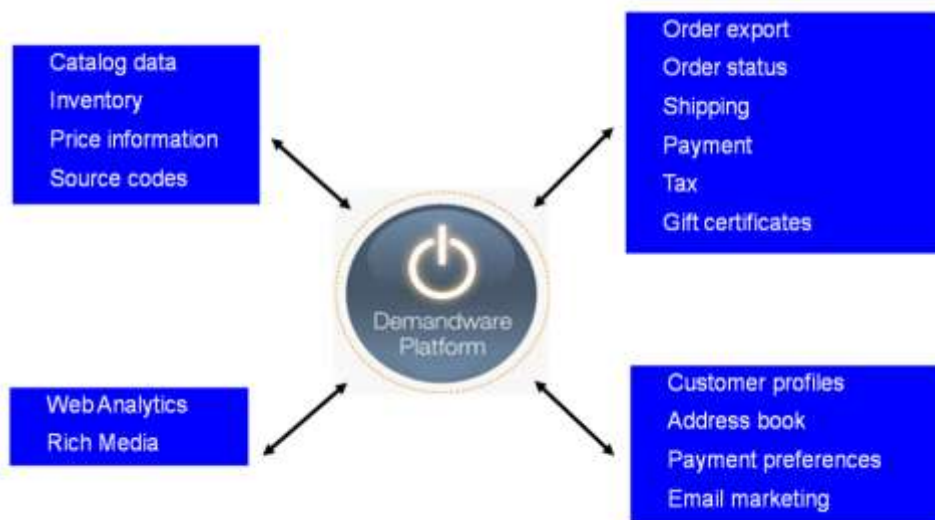
## Introduction

In this lesson we will cover two types of data integration:

- Simple Feed Integration
- Web Services

All new Demandware clients need to integrate data from other data systems. This lesson will cover the best practices for determining what type of integration is most effective depending on the frequency needs of the data.

Typical integration data points in Demandware are:



# Data Integration: Simple Feeds

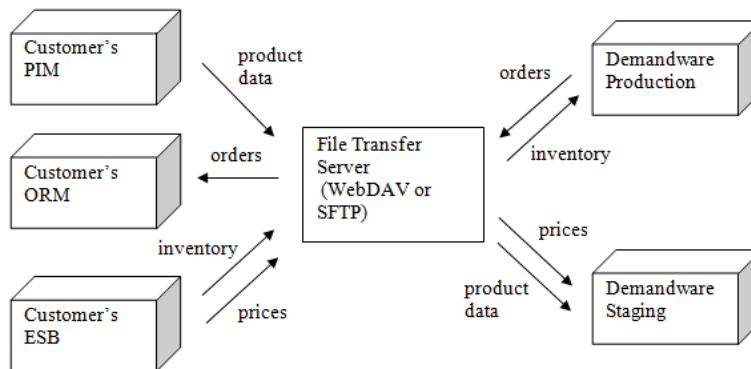


## Introduction

The idea of Simple Feed Integration is to couple Demandware and external systems loosely by exchanging files on a File Transfer Server. Simple Feed Integration supports the protocols WebDAV (HTTP and HTTPS) and SFTP for the transfer with the File Transfer Server. WebDAV HTTP (no network layer encryption) is meant for development/testing purposes only. When considering WebDAV HTTPS, please mind that an SSL-certificate from a Demandware accepted Certificate Authority (CA) needs to be installed at the File Transfer Server. The list of accepted CAs is available at Demandware's Customer Central.

The File Transfer Server needs to be hosted by the customer or another 3rd party. Demandware does not offer hosting File Transfer Servers. The WebDAV access to certain folders on a Demandware instance cannot be used for that purpose.

The simple feed cartridges support the following integration flow:



## File Format

The file format for the incoming and outgoing feeds is the standard Demandware import/export format. For XML files the schema definitions (XSD files) can be downloaded from Customer Central. A good approach to create a sample file is to set up the data as desired in Business Manager, run an export from the Business Manager user interface and use the exported file as a template.

All import files can alternatively be provided in gzip format (not to be confused with zip). In that case provide the file with extensions `.xml.gz` or `.csv.gz` at the File Transfer Server and adjust the file matching rule. The validation and import processes will automatically unzip the files during processing.

### Jobs

To perform an execution of functionality jobs within Demandware come as three types: scheduled, import/export, and batch. You can schedule jobs to run according to a schedule or for one time only. Demandware also provides a means of detecting problems with job execution, notification of job status and recovery from failed jobs. For instance, if a job fails to execute successfully, you can send an e-mail notification to alert someone to the problem. This feature enables job failure rules to be set, and jobs to automatically re-run if execution fails. You can set job failure rules and retry parameters such as number of retries.

You can define jobs that execute on the global (or organization) level and jobs that execute only on a site level through **Administration > Operations**.

### 11.1. Exercise: Exporting custom object NewsletterSubscription using a job

In this exercise you will clean up the NewsletterSubscription Custom Object. It will be first exported to an xml file and then the original one in the system will be removed. We will use the DeleteCustomObjects-Start pipeline to perform the export and removal.

1. Modify the Custom Object Definition for NewsletterSubscription
  - a. Go to **Administration→Site Development→Custom Object Definitions**
  - b. Add a new **attribute** to the Custom Object Definition called “exported”
  - c. Choose its data type as boolean
2. Upload the delete\_custom\_object cartridge to your Sandbox through the UXStudio. **Hint:** Navigate to **DemandwareServer->Properties->Project References** then check the cartridge that you want to upload.
3. Add the delete\_custom\_object cartridge to the required cartridge paths for executing a job in the Site scope.
  - a. Go to **Administration→Sites→Manage Sites→Business Manager**
  - b. Add the delete\_custom\_object cartridge **to the Business Manager site** (makes the pipeline available inside BM)
  - c. Make sure that it appears in the path. You may have to click the **Apply** button twice.
4. Again go to **Administration→Sites→Manage Sites→SiteGenesis→Settings tab**
  - a. Add same cartridge (delete\_custom\_object) to the **SiteGenesis site** (for debugging & email notifications) cartridge path.
  - b. Again make sure that it appears in the path.
5. Setup your job to run the pipeline.
  - a. Navigate in Business Manager to **Administration →Operations →Job Schedules**.
  - b. Create a new job, name it e.g. DeleteCO,
  - c. Set the execution scope to Sites
  - d. Mark it as Enabled using checkbox
  - e. Pipeline is DeleteCustomObjects,
  - f. Start node is Start.
  - g. Click on **Apply** button.
6. On the **Sites tab** select the **SiteGenesis** site and click on **Apply** Button.
7. Set up notifications for success/status of the job.

- a. On the **Notification** tab
  - b. Check all possible thresholds.
  - c. Enable it.
  - d. Configure your email address to receive job execution status emails.
  - e. The sender's email should be **donotreply@demandware.com**
  - f. Click on the Apply button.
8. Run the job.
  - a. Go to the **General** tab and **Run** the job. You should receive an email about the success.
9. Check if the job has removed the previous objects (not object definition) and exported them in the xml file.
  - a. Navigate to **Site Development → Development Setup → Import/Export**
  - b. Check the export directory called `"/src/customization/"`. Do you see the xml file containing objects of type NewsletterSubscription?
  - c. Go to **Site→SiteGenesis→Custom Objects**
  - d. Try to find objects of type NewsletterSubscription. They are gone (deleted).
  - e. In the log directory you'll find the respective export log file.

Check the export directory called `"/src/customization/"` under **Site Development → Development Setup → Import/Export** for the exported CO and in the log directory you'll find the respective export log file.

### Import into Production

Technically all object types can be imported on all Demandware instance types. Importing data directly into the instance Production may have adverse effects as there is no preview or rollback functionality available. Please load all feeds that can be previewed (e.g. catalog data) into instance Staging for approval and use Demandware's data replication feature for promotion to Production. Data replication also comes with rollback functionality (covered later in this course.)

Consider the different Catalog import modes that the catalog/pricebook/inventory imports support:

XML File Objects	Pre-Import Database Objects	Import Mode	Post-Import Database Objects
A <sup>+</sup> a, B <sup>-</sup> a, C, D, F <sup>Δ</sup> a	A, B, C, E, F	MERGE	A <sup>+</sup> a, B, C, D, E, F <sup>Δ</sup> a
		UPDATE	A <sup>+</sup> a, B, C, E, F <sup>Δ</sup> a
		REPLACE	A <sup>+</sup> a, B <sup>-</sup> a, C, D, E, F <sup>Δ</sup> a
		DELETE	E

A<sup>+</sup>a → an object with a new attribute

B<sup>-</sup>a → an object with an attribute removed.

C → an object left unchanged

D → a new object

F<sup>Δ</sup>a → an object with an attribute value that is changed

Some imports support an attribute mode at the import element level. In this case, the only supported mode is DELETE, whereby the import mode for the process can be overwritten for a particular element. This is useful when feeding changed information to a Demandware system, allowing a single import process to create, update and delete objects.

You can overwrite the XML file import mode for a single element, but only to specify DELETE:

```
<product product-id="12345" mode="delete"/>
```

When considering imports directly into Production mind that Demandware imports typically commit changes to the database on a per object basis. This means during the execution of the import some updates may already be visible in the storefront, others not. The catalog import is a two pass import: during the first pass objects are updated, during the second pass relationships between objects are updated. It is not recommended running catalog import directly into Production with Simple Feed Integration.

### Simple Feed Cartridges

The two cartridges included in the Simple Feed integration are:

- `int_simplefeeds`: Implementation of generic logic. Needs to be assigned to the storefront site and the Business Manager site. Modification of the `Custom_FeedJobs` pipeline is necessary to use the feed configuration custom object. Do not modify other code.
- `test_simplefeeds`: Cartridge to help troubleshooting WebDAV or SFTP connection issues and trigger a Simple Feed Integration job from the storefront during development. This cartridge must not be assigned to a site on Production systems. It may be assigned to the storefront site on sandbox instances if the storefront is password protected.

WebDAV is the only protocol that DW supports where DW can access files external to Demandware and where external systems can push files to DW. It is the only protocol that works on both directions.

You cannot use SFTP to access files in DW: there is no SFTP server in DW instances. However, DW can access an external SFTP server.



### How-To : Import objects using simple feed using the Demandware int\_simplefeed cartridge

1. If you have not already done so, download and import the `int_simplefeeds` cartridge into Studio.
2. If not imported already, import Custom Cartridges to your Project Code Base:
  - a. Add `int_simplefeeds` to your code repository (e.g. SVN)
  - b. Import into Studio for uploading to the server.
3. Assign Cartridge to Sites
  - a. The feeds will be processed by Demandware jobs that are executed in the context of a site. Pipelines for jobs are looked up in the cartridges at the Business Manager site, i.e. Sites-Site; templates (they are used for emails) are looked up in the cartridges for the respective storefront site.
  - b. In Business Manager go to **Administration > Sites > Manage Sites** and then to **Sites-Site**, and to each storefront site. Under Cartridges add `int_simplefeeds`.
4. Import Custom Object Type Definitions
  - a. The generic implementation uses a custom object, `FeedJobConfiguration`, to store configuration data. The custom object exists in the context of a site.
  - b. The custom object definition can be found in `metadata/FeedJobConfigurationCO.xml`. It needs to be loaded on all instances that use the feed integration cartridge. You can store, share and distribute the definition using Site Import / Export. To get it loaded in the first place please follow the instructions below:
  - c. In Business Manager navigate to **Administration > Site Development > Import & Export**. There you upload the file and then import.
 

*Important: Business Manager users that have the right to modify custom objects, can view and modify Simple Feed Integration configuration data, including connection endpoints, login/password, and public key for encryption of credit card data in order to export feeds. Make sure access privileges in Business Manager are set so that only authorized users can access respective modules.*
5. Create Job Schedules
  - a. Technically an arbitrary number of schedules and feed configurations can be set up on a Demandware instance. Currently, however, it is not possible to tell the schedule what configuration to use. So the only way to tie a schedule to a configuration is to use a unique pipeline start node. The cartridge



`int_simplefeeds` comes with a pipeline `Custom_FeedJobs`. There is a start node `StartDefault` that reads the feed configuration with the ID “Default”. In this document we will be dealing with that start node and that feed configuration only. If you wish to set up additional configurations, you need to add additional parameters and reference those when setting up the schedules.

- b. In most cases the feed jobs are triggered by a pre-defined schedule (e.g. hourly, daily). It is also possible to keep the schedule disabled and trigger it manually in Business Manager, or to provide code that triggers the job with the `RunJobNow` pipelet.
  - c. To set up a new job schedule, navigate in Business Manager to **Administration > Operations > Job Schedules**. Create a new job, name it e.g. `CatalogUpdate`, make sure the execution scope is `Sites`, Pipeline is `Custom_FeedJobs`, **Start** node `StartDefault`.
  - d. On the **Sites** tab you select all storefront sites you want to execute the job for. Please mind that you can provide site specific feed job configurations – see next step.
  - e. On the **Resources** tab you specify all object types your feed job is going to modify. The job framework will then try to acquire locks for these resources before the job is started, effectively avoiding parallel execution with other import processes or data replication.
  - f. On the **Notification** tab you can configure email addresses to receive job execution status emails. Additional, more granular success and error emails can be sent by feed tasks. They are configured with the task. Feed tasks can report temporary errors such as communication errors with external systems as job failures to the job framework (on-temporary-error: FAIL). Here you can define how to handle them: Continue as Scheduled, Retry, or Stop on Error.
  - g. In the **Parameters** tab you can define a parameter for theoretically any resource you want to import into a site.
6. Create Feed Job Configuration
    - a. A feed job configuration is stored in a site specific custom object and may contain multiple tasks (e.g. `DownloadAndImportCatalog` and `DownloadAndImportPriceBooks`).
    - b. To create a feed job configuration go to the site in **Business Manager > Custom Objects > Custom Object Editor**. Create a new instance of Feed Job Configuration.
    - c. As ID provide the ID you previously used when creating a new instance of the `FeedJobConfiguration` custom object (the identifier used in pipeline `Custom_FeedJobs-StartDefault` → check the properties of the **Script Node**),

An example for TasksXML you can find in the file `documentation/TasksXMLCatalogOnly.xml`. That file lists all supported tasks and contains inline documentation. Project specific configurations can be derived from that file just by removing unneeded tasks and updating the information. It may be sensible to have the same task (e.g. `DownloadAndImportCatalog`) multiple times in a feed configuration if feeds from different sources or with different content are processed.

### 7. Testing

- a. The structure of the XML for the custom object `FeedJobConfiguration` can be found in the sample file `TasksXML.xml` in the folder “documentation”.
- b. There are sample import files in the folder “sampledata”.
- c. In most cases the feed jobs are triggered by a pre-defined schedule (e.g. hourly, daily). It is also possible to keep the schedule disabled and trigger it manually in Business Manager, or to provide code that triggers the job with the `RunJobNow` pipelet.

### IMPORTANT NOTE

It is not advisable to expose the `RunJobNow` pipelet in a public pipeline as this could be exploited to flood your job queue. If you must use this pipelet in the storefront, please implement a custom security mechanism so that only authorized requests can execute the pipelet.



### Run the Simple Feed Integration activity.

Walk through steps 1-3a with the students of the following exercise. Then have them do the rest on their own.



## 11.2. Exercise: Simple Feed Integration

1. Import two Simple Feed Cartridges as explained below. These two cartridges will help in importing data from WebDAV file server.
  - a. In Studio, import the two cartridges that appear in the `C:\projects\simplefeeds\cartridges` directory.
  - b. Add the following cartridges to the Project References on your sandbox connection project so they get uploaded to your sandbox.
    - i) `int_simplefeeds`
    - ii) `test_simplefeeds`
  - c. Add the `int_simplefeeds` cartridge to the **SiteGenesis** site (for downloading the files)

Manage Sites > SiteGenesis - Settings

General Settings Cache Security Robots

**SiteGenesis - Settings**

Click **Apply** to save the details. Click **Reset** to revert to the last saved state.

Instance Type: Sandbox / Development

Deprecated! The preferred way of configuring HTTP and HTTPS host name values set in this section will be used if no hostnames are defined by alias.

HTTP Hostname:

HTTPS Hostname:

Instance Type: All

Cartridges: `int_simplefeeds:cor`

Effective Cartridge Path: `int_simplefeeds:cont`

- d. Also add the `int_simplefeeds` cartridge to the **Business Manager** site (for debugging & email notifications) cartridge path.

Manage Sites > Sites-Site - Settings

Settings Cache

**Sites-Site - Settings**

Click **Apply** to save the details. Click **Reset** to revert to the last saved state.

Instance Type: Sandbox / Development

Deprecated! The preferred way of configuring HTTP and HTTPS host names is by using intended only to support older configuration style.

HTTP Hostname:

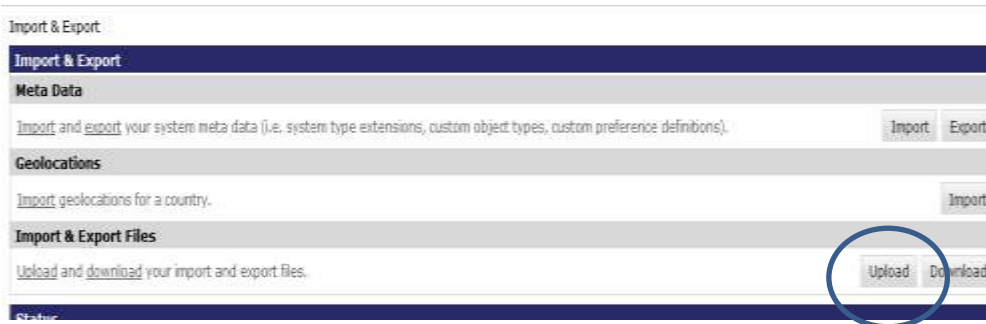
HTTPS Hostname:

Instance Type: All

Cartridges: `int_simplefeeds:bm_custom_plugin`

Effective Cartridge Path: `app_business_manager:bc_api:bc_in`

2. Import the `FeedJobConfiguration` Custom Object Definition from the xml file `FeedJobConfigurationCO.xml`. This type of object will be used later by a pipeline named `CustomFeedJobs` which is in the `int_simplefeeds` cartridge.
  - a. Navigate to **Administration** ⇒ **Site Development** ⇒ **Import & Export**.
  - b. Click on the **Upload** button on the right.



- c. Click on the **Browse** button to browse to `C:\projects\training\cartridges\simplefeeds\metadata\` and upload the `FeedJobConfigurationCO.xml` file.
- d. Click on **Back** button to go to the previous screen.
- e. Now click on the **Import** button on the top (Meta Data, not the one in the Geolocations section).



Result: this creates a `FeedJobConfiguration` custom object type for the instance.

- f. Check it by going to **Administration**→ **Site Development** → **Custom Object Definitions**. Do you see a Custom Object Type named `FeedJobConfiguration`?
3. Edit the `TasksXMLCatalogOnly.xml` file as explained below. The contents of this file will become an attribute value of our custom object which will be used by the pipeline to determine the logic for importing our data.

a. Open

C:\projects\training\cartridges\simplefeeds\documentation\  
TasksXMLCatalogOnly.xml file using your favorite editor.

b. Refer to your specific student## instance as follows.

```
<remote-folder-url>  
    https://mydisk.se/training/studentXX/  
</remote-folder-url>
```

Instead of studentXX, use the number given by the instructor.

c. Use your email address for job notifications for e.g.

```
<success-email>noreply@demandware.com</success-email>  
<error-email>noreply@demandware.com</error-email>
```

d. Note: This file is a simplified version of the TasksXML.xml file. It points to a specific, non- Demandware WebDAV server, and performs only a catalog download and import.

4. In **SiteGenesis ⇒ Custom Object ⇒ Custom Object Editor**, create a new FeedJobConfiguration object (not definition) with the following values for its attributes:

- ♦ id - catalogTest
- ♦ From Email - **your email**
- ♦ Tasks XML - copy the contents of  
C:\projects\training\cartridges\simplefeeds\documentati  
on\TasksXMLCatalogOnly.xml to this field.

5. Create a job to test the catalog import.

a. Navigate in Business Manager to **Administration > Operations > Job Schedules**.

- i. Create a **new schedule**,
- ii. Name it CatalogUpdate
- iii. Check the **Enabled** checkbox
- iv. Change the **Execution Scope** to Sites
- v. Set the pipeline to Custom\_FeedJobs,
- vi. **Start** node to StartDefault.
- vii. Click on the **Apply** button

Schedules > Details

General Sites Resources Notification Parameters

**New Schedule**

Fields with a red asterisk (\*) are mandatory.  
Click "Apply" to save the details. Click "Reset" to discard changes.  
Dates and times are in local standard format (date format: MM/dd/yyyy, time format: h:mm a)

**Name:**\* CatalogUpdate ☒ **Enabled**

**Description**

**Execution Scope:**\* Sites

**Pipeline:**\* Custom\_FeedJobs

**Startnode:**\* StartDefault

**All date and time inputs in instance time zone US/Eastern.**

**Run Once:** 11/03/2011 at 4:08 am  
MM/dd/yyyy h:mm a

- b. On the **Sites** tab select the **SiteGenesis** site and click on the **Apply** button

Schedules > CatalogUpdate - Sites

General Sites Resources Notification Parameters

**CatalogUpdate - Sites**

The list below contains all sites you may assign to your schedule.  
To configure a job schedule for a list of sites, select all of the sites.

Select All	Name
<input type="checkbox"/>	Sites-Site
<input checked="" type="checkbox"/>	SiteGenesis

- c. On the **Resources** tab
- Specify the **catalog** and **product** object types your feed job is going to modify.

Schedules > CatalogUpdate - Resources

General Sites Resources Notification Parameters

**CatalogUpdate - Resources**

To assign a new resource, either select a "System Resource" from the list below or click "Custom" to create a new resource.  
To remove a resource, select the check box next to the resource.

**New Resource**

System Resource:  Custom

Select All	Resource Path
<input type="checkbox"/>	product
<input type="checkbox"/>	catalog

- d. On the **Notification** tab
- Configure your email address to receive job execution status emails as shown in the screenshot below.
  - Also check the **Enabled** checkbox
  - Click on **Apply** button

Schedules > CatalogUpdate - Notification

General Sites Resources Notification Parameters

**CatalogUpdate - Notification**

Fields with a red asterisk (\*) are mandatory ONLY if notification is enabled.  
Click "Apply" to save the details. Click "Reset" to discard changes.  
The "To" and "CC" fields will accept a comma separated list of up to 5 email addresses. The "From" field will accept 1 email address.

Enabled ☒

Threshold: ☒ SUCCESS ☒ EXCEPTION ☒ ERROR ☒ HANG ☒ RETRY

From: donotreply@demandware.com

To: youremail@yourdomain.com

CC:

**Job Failure Rules**

Set the retry behavior, number of retries to attempt, and retry interval you would like to occur if this job has an error while executing. Send a notification if notification has been enabled.

Failure Rule: Continue as Scheduled

Retries: 0

Retry Interval: 0 Minutes

**Hang Detection**

Detect if this job has been running longer than the configured max execution time. Send a notification if notification has been enabled.

Enabled ☐

Max Execution Time: 60 Minutes

6. On the **Parameters** tab use the same **ID** as in the **Script** Node of the Custom\_FeedJobs-StartDefault pipeline (Check it, it is Jobid ?). As **Value** use the ID of the instance of your FeedJobConfiguration (catalogTest) custom object.

Schedules > CatalogUpdate - Parameters

General Sites Resources Notification Parameters

**CatalogUpdate - Parameters**

You can access your static job parameters via the job pipeline dictionary key `CurrentJobParameterMap`. Provide a Name and a Value, and click Add to create a new parameter entry. To remove a parameter entry click the red Delete icon to the right of the parameter. Click Reset to rollback changes.

Name	Value
Jobid	catalogTest

Add

Reset

7. Run the job:
  - a. Visit <https://mydisk.se/training/student##/> to see the contents of the WebDAV server.
  - b. Verify that the remote WebDAV server directory referenced in the TasksXML is already populated with test files from the `C:\projects\simplefeeds\sampledata` directory.
  - c. Test the job by running the feed job from Business Manager.

Schedules

**Schedules**

**Note, that the time based execution of custom schedules i**

The list below shows all job schedules defined for the Sites organization.

Select All	Name
<input checked="" type="checkbox"/>	<b>CatalogUpdate</b>

Enable Disable Run

8. Verify if the job worked (Your catalog is imported)

- a. Go to **Site** → **SiteGenesis** → **Products and Catalogs** → **Catalogs** → **Catalog** → **Cat1**
- b. Check if **Prod1** is assigned to it. Do you know where it came from? From the WebDAV server ...from a file named `Catalog_2008-03-11_20-49-12.xml` perhaps.



# Data Integration: Web Services



## Introduction

When you need real-time data integration with a 3<sup>rd</sup> party system or your own backend, you need to invoke a web service.

A web service is a remote service created to expose data in a database, or the result of a calculation. It is a remote procedure call that goes through HTTP. There are a number of supported protocols for web services but for the purpose of this course, we will focus on SOAP style web services.

If you are not familiar with web services or the SOAP protocol, you can learn more at the website: <http://www.w3schools.com/soap/default.asp>.

In Demandware, Axis 1.4 is used to compile WSDL files. As of this writing only SOAP 1.1 is supported. In Demandware, once the WSDL is uploaded all the supporting class files get automatically generated. To access these files you use the Demandware `dw.rpc.webreference` class. An instance of this class represents the package of all the generated WSDL classes:

```
var webref : WebReference = webreference.CurrencyConvertor;
```

The `webreference` class has one property, called `defaultService`. The `defaultService` property returns a stub:

```
var stub : Stub = webref.defaultService;
```

You can also invoke a specific service if you know its name and port:

```
webref.EducationServiceSOAP.getService('EducationService',  
'EducationServiceSOAP'),
```

The `Stub` instance allows you to access remote methods in your web service:

```
var response : ConversionRateResponse =  
stub.conversionRate(cr);
```

## Generating WSDL Javadocs

The graphical presentation of a WSDL in Studio does not always make it clear how to invoke methods on the Web Service. Optionally, you can generate Java classes and javadoc to help you write the Demandware Script code:

- ◆ Download the Axis 1.4 and Xerces jar files from Apache.

- ♦ Create an AXISCLASSPATH environment variable to point to the downloaded jars.
- ♦ Generate Java stubs using the command:

```
java -cp %AXISCLASSPATH% org.apache.axis.wsdl.WSDL2Java  
--noWrapped --all --package currencyconvertor  
CurrencyConverter.wsdl
```

- ♦ Generate the Javadoc for use in coding:

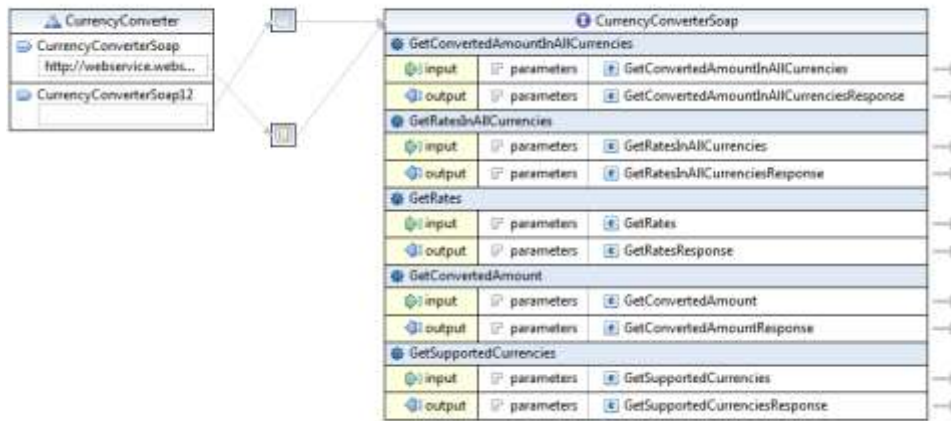
```
javadoc -d currencyconvertor\javadoc currencyconvertor
```

### WSDL File

A WSDL file is a document that describes a web service. It specifies the location of the service and the operations (or methods) the service exposes.

In Studio, you can view the WSDL file either graphically or programmatically:

- WSDL Graphical view



- WSDL source code view

```

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://microsoft.com/wsdl/name/t"
  <wsdl:types>
    <xs:schema elementFormDefault="qualified" targetNamespace="http://schemas.microsoft.com/finance/2006/01/01/GetConvertedAmountInAllCurrencies">
      <xs:element base="tns:GetConvertedAmountInAllCurrencies">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="1" name="Key" type="xs:string" />
            <xs:element minOccurs="0" maxOccurs="1" name="CurrencyFrom" type="xs:string" />
            <xs:element minOccurs="1" maxOccurs="1" name="Amount" type="xs:double" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
  <wsdl:message name="tns:GetConvertedAmountInAllCurrenciesResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="tns:GetConvertedAmountInAllCurrenciesResult" type="tns:Cur" />
      </xs:sequence>
    </xs:complexType>
  </wsdl:message>
  <xs:schema name="CurrencyXsd">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="1" name="CurrencyFrom" type="tns:CurrencyFromRow" />
        <xs:element minOccurs="0" maxOccurs="1" name="CurrencyTo" type="tns:CurrencyToRow" />
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
  <xs:schema name="CurrencyFromRow">
    <xs:complexType>
      <xs:sequence>

```

As mentioned previously, the WSDL file is all that is needed for the Demandware server to generate the java classes for accessing the methods in the remote service. You will need to write a script file for accessing those classes using the Demandware API.



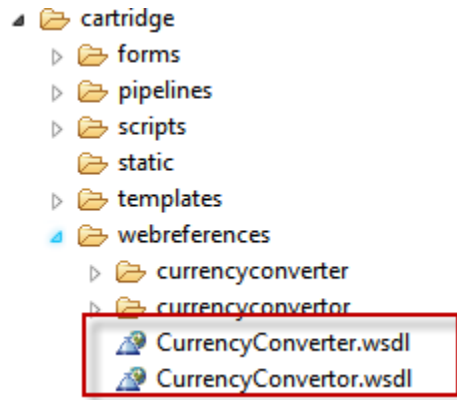
### Run the Use a Web Service activity.

Walk the students through the `CurrencyConvertor` web service by studying the WSDL, the javadoc and the script that invokes the service. Have the students walk through the code by using the debugger.



### To integrate a web service, you must perform the following steps:

1. Obtain the WSDL file for the web service you wish to invoke.
2. Place the WSDL file in the `webreferences` folder in your cartridge.



3. Using a script pipelet, add the following to your script:
  - Import the `dw.rpc` package
  - Create a webreference object (example below):
 

```
var webref : WebReference = webreferences.WSDLFileName
```
  - Get service stub object (example below):
 

```
var stub : Stub = webref.defaultService
```
4. Invoke the methods/properties from the web service you need. Generate Javadocs from the WSDL if you need help invoking the remote method(s).



### 11.3. Exercise: Web Services

Use a Web Service to obtain the conversion rate between two currencies.

1. Place the `solutions` cartridge in front of the `storefront` cartridge in the `SiteGenesis` cartridge path.
2. Locate the `CurrencyConvertor` pipeline.
3. Study the code in the `CurrencyConvertor.ds` script file.
4. Study the `webreferences/CurrencyConvertor.wsdl`
5. Review the Javadoc that was generated from the WSDL file:  
`webreferences/currencyconvertor` directory, specifically the single method in the `CurrencyConvertorSoap12Stub`:

```
public ConversionRateResponse conversionRate(ConversionRate parameters)
    throws java.rmi.RemoteException
```

6. Modify the `Assign` pipelet properties to **currencies of your choice**: make sure you use currencies supported in the WSDL.
7. Test the `CurrencyConvertor-Start` pipeline by debugging the script code and studying the objects instantiated.

Question	Answer
<p>1. When using web services, what does the instance of the <code>dw.rpc.WebReference</code> represent?</p> <ul style="list-style-type: none"><li>a) The Web Service</li><li>b) The package compiled from the WSDL</li><li>c) The WSDL</li><li>d) The remote procedure call</li></ul>	
<p>2. What protocols are supported for simple feeds?</p> <ul style="list-style-type: none"><li>a) FTP</li><li>b) WEBDAV</li><li>c) SFTP</li><li>d) WebDAV &amp; (S)FTP</li></ul>	



---

### Transition to Deconstructing SiteGenesis

---

## 12. Integration Framework



---

### Goal

The purpose and goal of this module is to understand another framework called Integration Framework.



---

### Time

2 Hrs



---

### Overview

We will discuss to import and export catalog and create your own workflow.



---

### Materials Needed

The following cartridges are needed to run this module:  
`bc_integrationframework`, `bm_integrationframework`,  
`bc_library`

---

# Integration Framework Overview



## 1. Introduction

The growing complexity of the batch processes required to operate a shop, have brought in some cases the existing implementation to its limits. On the one hand side because of the growing number of batch components (Jobs) and on the other hand due to lacking monitoring and notification capabilities.

This concept will define a general way how workflows and its components need to be built so that a common infrastructure can provide facilities for monitoring, logging, configuration, scheduling etc. One such framework which allows us to do so is Integration Framework. It is nothing but a combination of cartridges, preferences, custom objects and meta-data which together can help create a workflow and monitor it. The first step towards using Integration Framework is installing it.



## 12.1. Exercise: Installing Integration Framework Cartridges and basic setup

In this exercise you will do the basic setup needed for the Integration Framework.

1. If not already done, import three cartridges `bc_integrationframework`, `bm_integrationframework`, and `bc_library` into your UX Studio. Hint: **File → Import Projects → General → Existing Projects** and navigate to the cartridges that you want to import.
2. Make sure that these are marked for uploading to your Sandbox (Hint: **Demandware Server → Properties → Project References** check all cartridges that you want to upload – including the previously unselected `bc_library`)
3. **Import Site:** Find and import `site_template.zip` from the `integrationframework\sites` directory (it contains your meta data, custom objects and preferences). Hint: **Administration → Site Development → Site Import & Export**. Upload the zip file and then import it.
4. Assign the `bc_integrationframework` and the `bc_library` cartridge to the SiteGenesis site.
5. Assign the `bc_integrationframework`, `bm_integrationframework` and `bc_library` cartridges to Business Manager.
6. Add a site to the Schedule Instance using the following steps:
  - a. Navigate to **Administration → Global Preferences → Custom Preferences → Integration Framework**.
  - b. Under 'Available sites (IDs)' click '**Add another value**' for each site the jobs using the Integration Framework are running in and add two sites there (Ignore this step if they are already there)
    - i. Sites-Site
    - ii. SiteGenesis
  - c. Click **Apply**

**Global Custom Preferences**

This page allows you to define the global custom preferences of group Process Framework.  
For each Demandware instance type, you can enter a different preference value. Use the select box to switch between instance types. If no value is defined for a preference, the system uses the defined default value.  
To save your settings, press the 'Apply' button.

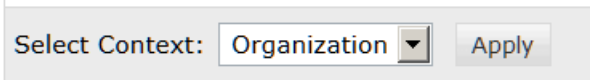
Instance Type: Sandbox / Development Apply

Preference Name	Sandbox / Development Value	Default Value
Available sites (IDs):	<div> <div>Sites-Site</div> <div>SiteGenesis</div> <div>Add another value</div> </div>	

All site IDs, which are configured for the instance.

Apply Reset

7. Assign Business Manager Modules **Workflow Schedules** and **Workflow Plan** to the admin role as explained below:
  - a. Navigate to **Administration → Organization → Roles & Permissions → Administrator → Business Manager Modules** (Tab)
  - b. Select the context as **Organization** and click on **Apply**



Select Context: Organization ▼ Apply

- c. Check the two modules **Workflow Schedules** and **Workflow Plan**. If you do not see them, then check step 1 through step 6 again. (If it still does not show up, request the instructor to restart your sandbox).
  - d. Click on update at the bottom of the screen.
8. You can now make use of Integration Framework. (You will see that in the next exercise)

# General concept

Every workflow that needs to be executed is modelled as a **WorkflowSchedule**, containing main information about frequency, run times, enablement, file log level, etc.

The screenshot shows a configuration form for a workflow schedule. The fields are as follows:

- ID:** CatalogExport
- Enabled:** ☐
- Name:** Catalog Export & Inventory Import
- Description:** Exports a Catalog of a specific site & imports an Inventory to the site
- Sites (their IDs):** ☐ MobileGenesis ☒ SiteGenesis
- File Log Level:** error
- Single:** ☐
  - Runtime:** [ ] at [ ]
- Recurring:** ☒
  - Active:** 07/17/2012 to: [ ]
  - Runtime:** 12:00 pm
  - Every:** 12 Hour
  - On these days:** ☒ Monday ☒ Tuesday ☒ Wednesday ☒ Thursday ☒ Friday ☒ Saturday ☒ Sunday
  - Enable Notifications:** ☒
    - On these statuses:** ☒ OK ☒ WARN ☒ ERROR
    - Email addresses:** address@email.com
    - [Add another value](#)

Every such **WorkflowSchedule** consists of simple **WorkflowComponentInstances** (created from the existing **WorkflowComponentDefinitions**), describing the steps within the workflow and their individual configurations.

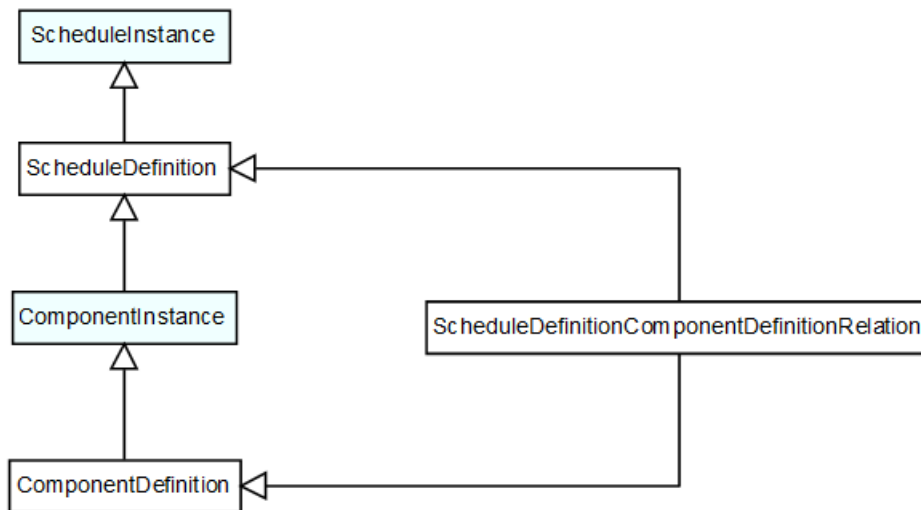
The screenshot shows a configuration form for a workflow component instance titled 'Component: Catalog export'. The fields are as follows:

- Type:** Catalog export
- Description:** Exports a Catalog
- Action:** GeneralExport-Catalog
- Disabled:** ☐
- Asynchron:** ☐
- File Log Level:** debug
- Parameters:**
  - Catalog ID:** apparel-catalog
  - File name prefix:** master\_catalog\_export
  - Export Categories:** ☒
  - Export Category Assignment:** ☒
  - Export Product Options:** ☒
  - Export Products:** ☒
  - Export Recommendations:** ☒
  - Export Variation Attributes:** ☒
  - Overwrite export file:** ☒

A **Delete** button is located at the bottom right.

For recurring schedules the schedule itself and its components are cloned for each schedule run.

In Demandware platform terminology, a `WorkflowSchedule` consists of a custom object defining the workflow and a number of job schedule definitions in the Demandware BM which check for workflows to execute on a regular basis. Every `WorkflowComponentInstance` relates to `WorkflowSchedule` objects as well.



## 2. Scheduled Job Configuration

Standard Demandware jobs are used to trigger the workflow engine (the job handler framework starts waiting workflow schedules defined in Operations for a specified scope). All other calculation is done by the framework itself. This allows for example the displaying of upcoming workflows even days in advance.

To setup a scheduled-Job:

1. Navigate to **Administration → Operations → Job Schedules**
2. Create a new or edit an old Schedule
3. Set the Execution Scope to either 'Sites' or 'Organization' depending on the Scope your pre-defined Workflow-Schedule run in. *If your Workflow-Schedule should run in both scopes, please create a Scheduled-Job for each Scope.*

- Make sure that pipeline 'Workflow' and start node 'Start' are already pre-configured :

**Workflow-Start-Site-1 - General**

Fields with a red asterisk (\*) are mandatory.  
Click "Apply" to save the details. Click "Reset" to discard changes.  
Dates and times are in local standard format (date format: MM/dd/yyyy, time format: h:mm a).

**Name:** Workflow-Start-Site-1 ☒ **Enabled**

**Description:** The job handler for the job framework. Starts waiting workflow schedules defined in Operations->Workflow Schedules.

**Execution Scope:** Sites

**Pipeline:** Workflow

**Startnode:** Start

**All date and time inputs in instance time zone Europe/Berlin.**

**Run Once:** 09/07/2012 at 10:47 am  
MM/dd/yyyy h:mm a

**Recurring Interval:**

**Active:** 01/01/2011 to   
MM/dd/yyyy MM/dd/yyyy

**Run Time:** 1:04 am  
h:mm a

**Every:** 5 Minutes

**On these days:** ☒ Monday ☒ Tuesday ☒ Wednesday ☒ Thursday ☒ Friday ☒ Saturday ☒ Sunday

**Run**

- The Provided parameter, JobId, associates to a workflow component with the job which triggers the frameworks processing engine. This JobId is used internally by the Workflow pipeline to be able to identify the job that invoked it.

**Workflow-Start-Site-1 - Parameters**

You can access your static job parameters via the job pipeline dictionary key `CurrentJobParameterMap`. Provide a Name and a Value and click **Add** to create a new parameter entry. To remove a parameter, click the **Remove** button right of the parameter. Click **Reset** to rollback changes.

Name	Value
JobId	Job-Sites-1
<input type="text"/>	<input type="text"/>

**Add**

- The Integration Framework is now completely configured and ready for use.

## Monitoring

A monitoring cockpit allows viewing the schedule of the current day as well as the schedule for a given date. For the selected day a list of Workflows that are scheduled is shown as well as their components are shown. All of them state their name, the current status and the status message.

Description

The Workflow overview shows the workflows of the selected day (default is the current date). In order to select a different day please use the "on ..." filter of the "Planned start" column (note that the from/to filters won't work).

In order to see the content of a workflow, simply click the "+" at the beginning of the row.

To sort by a certain column, simply click the header of that column. At the right of each header there is a menu with additional functionality.

Workflow Schedule Plan

Workflow Name	Planned start	Status	Start time	End time	Runtime	Site IDs	Log Files																													
Site IDs: SiteGenesis																																				
+ CleanupFiles	15.01.2013 04:25:00	FINISHED_OK	15.01.2013 04:25:00	15.01.2013 04:25:00	00:00:00.544	SiteGenesis	<a href="#">download</a>																													
+ CleanupFiles	15.01.2013 05:18:00	FINISHED_OK	15.01.2013 05:18:00	15.01.2013 05:18:00	00:00:00.000	SiteGenesis	<a href="#">download</a>																													
+ CleanupFiles	15.01.2013 06:31:00	FINISHED_OK	15.01.2013 06:31:00	15.01.2013 06:31:00	00:00:00.077	SiteGenesis	<a href="#">download</a>																													
+ Catalog Export & Inventory import	15.01.2013 06:45:00	FINISHED_ERROR	15.01.2013 06:45:00	15.01.2013 06:45:00	00:00:58.793	SiteGenesis	<a href="#">download</a>																													
<table> <tr> <th>Component Name</th> <th>Component Status</th> <th>Start time</th> <th>End Time</th> <th>Runtime</th> </tr> <tr> <td>+ Catalog export</td> <td>FINISHED_OK</td> <td>01/15/2013 06:45</td> <td>01/15/2013 06:45</td> <td>00:00:36.403</td> </tr> <tr> <td>+ Inventory import</td> <td>FINISHED_OK</td> <td>01/15/2013 06:45</td> <td>01/15/2013 06:45</td> <td>00:00:00.093</td> </tr> <tr> <td>+ Transfer From FTP</td> <td>FINISHED_ERROR</td> <td>01/15/2013 06:45</td> <td>01/15/2013 06:45</td> <td>00:00:21.195</td> </tr> </table> <table> <tr> <th>Time</th> <th>Log Message</th> <th>Log Files</th> </tr> <tr> <td>[2013-01-15 11:45:39.585 GMT]</td> <td>[INFO] Starting to download.</td> <td></td> </tr> <tr> <td>[2013-01-15 11:46:00.661 GMT]</td> <td>[ERROR] FTP-Download finished, the connection couldn't be established.</td> <td></td> </tr> </table>								Component Name	Component Status	Start time	End Time	Runtime	+ Catalog export	FINISHED_OK	01/15/2013 06:45	01/15/2013 06:45	00:00:36.403	+ Inventory import	FINISHED_OK	01/15/2013 06:45	01/15/2013 06:45	00:00:00.093	+ Transfer From FTP	FINISHED_ERROR	01/15/2013 06:45	01/15/2013 06:45	00:00:21.195	Time	Log Message	Log Files	[2013-01-15 11:45:39.585 GMT]	[INFO] Starting to download.		[2013-01-15 11:46:00.661 GMT]	[ERROR] FTP-Download finished, the connection couldn't be established.	
Component Name	Component Status	Start time	End Time	Runtime																																
+ Catalog export	FINISHED_OK	01/15/2013 06:45	01/15/2013 06:45	00:00:36.403																																
+ Inventory import	FINISHED_OK	01/15/2013 06:45	01/15/2013 06:45	00:00:00.093																																
+ Transfer From FTP	FINISHED_ERROR	01/15/2013 06:45	01/15/2013 06:45	00:00:21.195																																
Time	Log Message	Log Files																																		
[2013-01-15 11:45:39.585 GMT]	[INFO] Starting to download.																																			
[2013-01-15 11:46:00.661 GMT]	[ERROR] FTP-Download finished, the connection couldn't be established.																																			
Transfer To FTP																																				
WAITING																																				

Reload

## List of out of the box components


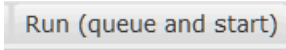
Components	
Component	Description
<b>Standard Components Import</b>	The component “Standard Components Import” allows importing any system or custom object from a designated local or remote directory (relatively to IMPEX/) via XML-Files which corresponds to a specific naming pattern.
<b>Standard Components-Download Files</b>	Allows you to import from a designated WebDAV or (S)FTP location into Demandware
<b>Catalog export</b>	The component “Catalog export” allows exporting of a specific Catalog into a designated local directory (relatively to IMPEX/).
<b>Order export</b>	The component “Order export” allows exporting orders (all or confirmed and paid) to a designated local directory (relatively to IMPEX/).
<b>Price-Book export</b>	The component “PriceBook export” allows exporting of a specific Price Book into a designated local directory (relatively to IMPEX/).
<b>Price import</b>	The component “Price import” allows importing Price-Books from a designated local directory (relatively to IMPEX/) via XML-Files which corresponds to a specific naming pattern.
<b>Import slots</b>	The component “Import slots” allows importing of content slot configurations from a designated local directory (relatively to IMPEX/) via XML-Files which corresponds to a specific naming pattern in a specified mode.
<b>Import content</b>	The component “Import content” allows importing of data from a designated local directory (relatively to IMPEX/) via XML-Files which corresponds to a specific naming pattern into a specified library and in a specified mode.

<b>Various Rebuild functionality</b>	To rebuild Search / Availability / ActiveData / Redirect / Suggestions / Synonyms Indexes
<b>Time condition</b>	The component “TimeCondition” allows ensuring that the workflow will not be continued in case a configured time is in the past or that the execution will be put on hold until a certain time is reached. This can be useful in case a replication or other workflow needs to happen before a certain point of time in order to not impact the production system performance.
<b>Date condition</b>	The component “DateCondition” allows ensuring that the workflow will not be continued in case a configured date is in the past or that the execution will be put on hold until a certain date is reached. This can be useful in case a replication or other workflow needs to happen before a certain date in order to not impact the production system performance.
<b>Date-Time condition</b>	The component “DateTimeCondition” allows ensuring that the workflow will not be continued in case a configured date and time is in the past or that the execution will be put on hold until certain date and time are reached. This can be useful in case a replication or other workflow needs to happen before a certain time in a certain day in order to not impact the production system performance.
<b>Workflow clean-up</b>	The component “Workflow-CleanUp” allows the removal of old custom objects which were created through the workflow framework.
<b>Clean-up files</b>	The component “CleanUpFiles” allows the removal and archive of old files in specific folders.
<b>Transfer to FTP</b>	The component “Transfer To FTP” allows the copy of files from a local directory (relatively to IMPEX/) to a FTP-Location.
<b>Transfer from FTP</b>	The component “Transfer From FTP” allows the copy of files to a local directory (relatively to IMPEX/) from a FTP-Location.
<b>Rebuild indexes</b>	The component “Rebuild indexes” allows the rebuilt of the configured indexes. This component can be useful after the import of a catalog for example.



For detailed information regarding the components and their Attributes please refer to <https://xchange.demandware.com> and search for **Integration Framework**.

### 12.2. Exercise: Exporting Catalog to an XML file

1. Navigate to **Administration → Operations → Workflow Schedules**
2. Click on the **New** button.
3. Fill in the form below with the following details :
  - a. **ID and Name** is up to you
  - b. Check **Enabled**
  - c. Mark **SiteGenesis**
  - d. File log level should be **info**
  - e. The date should be a **future date**
  - f. Enable **all** notifications
  - g. The email address should be **your own email address**
  - h. **Click on apply**
4. Add a new component named **Export-Catalog**
5. Choose **inherit** as file log level
6. Choose **inherit** as component log level
7. Catalog ID is **storefront-catalog-en**
8. File name prefix is **ExportedCatalog**
9. **Check all given options**
10. Click on **Apply**.
11. Click on Back to List 
12. Now check the box next to **the name you gave** and  click on at the bottom of the screen.
13. Navigate to **Administration → Operations → Workflow Plan**
  - a. Expand the following by clicking on the '+' sign



Keep expanding to see success message as below:

Workflow Name	Planned start	Status	Start time	End time
Site IDs: SiteGenesis				
ExportCatalog	03.06.2013 15:34:00	FINISHED_OK	03.06.2013 15:34:00	03.06.2013 15:34:00
Component Name		Component Status		
Export-Catalog		FINISHED_OK		
Time		Log Message		
[2013-06-03 19:34:10.528 GMT]		[INFO] Starting to export storefront-catalog-en.		
[2013-06-03 19:34:25.352 GMT]		[INFO] DW export successfully finished.		

14. Navigate to **Administration → Site Development → Development Setup**
15. Check your `/src/download/catalog` folder. (Hint: **Administration → Site Development → Development Setup → Import & Export** and Navigate from there) Do you see your catalog exported (download/catalog)? If not, consult your instructor.

### 12.3. Exercise: Importing catalog data from a remote WebDAV location

This demo imports Catalogs from a WebDAV server anywhere located. It uses exclusively out-of-the-box functionality.

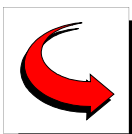
1. Before proceeding further, delete the previous catalogs that you imported in the simplefeeds exercise (**Site → Products and Catalogs → Catalogs**).
2. Navigate to **Administration → Operations → Workflow Schedules** and create a schedule as below:
  - a. ID: **CompleteWebDAVImport**
  - b. Enabled: **Checked**
  - c. Name: **CompleteWebDAVImport**
  - d. Description: **This is a complete import from WebDAV server to the Catalogs**
  - e. Sites: **SiteGenesis**
  - f. File Log Level: **info**
  - g. Component Log Level: **info**
  - h. Single: **Checked**
  - i. Runtime: **Give any future date and time**
  - j. Enable Notifications: **Check all**
  - k. Email Address: **give your own email address**
3. Add a **first** component to schedule as shown below:
  - a. Component: **StandardComponents-DownloadFiles**
  - b. Disabled: **Unchecked**
  - c. Asynchron: **Unchecked**
  - d. File Log Level: **inherit**
  - e. Component Log Level: **inherit**
  - f. Parameters:
    - i. Server type : **WEBDAV**
    - ii. URL of WebDAV server (use directory of your sandbox):  
<https://mydisk.se/training/student<your sandbox number>/>
    - iii. Username of WebDAV server: **training**
    - iv. Password: **train123**
    - v. Pattern of files to be imported: **^Catalog\_.\*\.xml\$**
    - vi. Target folder: **upload/catalog**
    - vii. Delete File: **KEEP\_ON\_SERVER**
    - viii. Extract Archives: **Unchecked**
4. Add a **second** component to schedule as shown below:
  - a. Component: **StandardComponents-Import**
  - b. Disabled: **Unchecked**
  - c. Asynchron: **Unchecked**

- d. File Log Level: **inherit**
  - e. Component Log Level: **inherit**
  - f. Parameters:
    - i. Object Type: **catalog**
    - ii. Working folder in IMPEX: **upload/catalog**
    - iii. File name pattern : **^Catalog\_.\*\.xml\$**
    - iv. Import Mode: **MERGE**
    - v. Treat 'no files found' as: **WARN**
    - vi. Treat Import Failed as: **ERROR**
    - vii. Catalog Import config (JSON): **<keep empty>**
    - viii. Handle file after process: **ARCHIVE\_ZIPPED\_WITH\_LOGS**
    - ix. Archive folder e.g.: **archive**
5. Run this **Workflow Schedule** you just created.
  6. Monitor it in **WorkflowPlan** (**Administration** → **Operations** → **Workflow Plan**)
  7. Check if `/src/archive` files has the archive with the five files in there.
  8. Check if your catalogs have been imported.

## Review



Question	Answer
1. Which of the following is true about Integration Framework? <ol style="list-style-type: none"> <li>a) Integration Framework lets you monitor the jobs.</li> <li>b) Integration Framework is the only way to create jobs.</li> </ol>	



## Transition to Deconstructing SiteGenesis

## 13. Deconstructing SiteGenesis



---

### Goal

The purpose and goal of this module is to familiarize you with the files used in SiteGenesis to render pages to a browser.



---

### Time

4 Hrs



---

### Overview

We will discuss the common files used to render pages in SiteGenesis and give an overview of the shopping cart checkout process.



---

### Materials Needed

None

---



### Introduction

As we discussed in the Demandware Architecture module, SiteGenesis is the reference storefront application for new client development.

The main pipelines used for generating storefront pages are:

- Search-Show
- Product-Show
- Page-Show
- Cart-Show
- Account-Show

In the past the first 3 pipelines only used to support SEO friendly URLs :

- Search-Show with CategoryID
- Product-Show with ProductID
- Page-Show for Content Assets

Since the 13.1 deployment Demandware supports customizable URL layout. You switch it on under **Site - Site Preferences - Storefront URLs**.

- Go to **SiteGenesis - Site URLs - URL Rules** to influence the settings for locales, Catalog URLs, Content URLs and Pipeline URLs

By now you should be able to open any pipeline in Studio and execute it from the storefront, using the debugger to step through each pipeline process.

### SiteGenesis Page Deconstruction Tools

You can use the Storefront Toolkit Page Information tool to help you break down the files used to generate a storefront page.

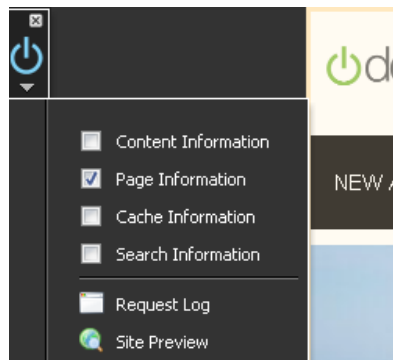


# Using the Storefront Toolkit: Page Information



## Introduction

The Storefront Toolkit is a development tool that aids a developer in troubleshooting a site and is only accessible in staging, development, and sandbox instances.



When you want to know what ISML templates are being used to generate a page, use the '**Page Information**' tool. When using the '**Page Information**' tool, you can quickly open up a template that is used to generate a page.

## Quilted Jacket

Item# 701642853695

\$159.00

**Product Rating**

★★★★☆

[Write a Review](#)

**SELECT COLOR**

**SELECT SIZE**

**QUANTITY**

1

\$159.00 | \$110.99 **ADD**

**Local Include**  
Open Template [/default/product/components/standardprice.isml \(storefront\)](/default/product/components/standardprice.isml (storefront))

**Local Include**  
Open Template [/default/product/components/pricing.isml \(storefront\)](/default/product/components/pricing.isml (storefront))

**Local Include**  
Open Template [/default/product/components/group.isml \(storefront\)](/default/product/components/group.isml (storefront))

**Local Include**  
Open Template [/default/product/productcontent.isml \(storefront\)](/default/product/productcontent.isml (storefront))

**## ???**

**## Local Include**  
Open Template [/default/product/producttopcontent.isml \(storefront\)](/default/product/producttopcontent.isml (storefront))

**### Decorator**  
Open Template [/default/product/pt\\_productdetails.isml \(storefront\)](/default/product/pt_productdetails.isml (storefront))

**#### Local Include**  
Open Template [/default/components/productbreadcrumbs.isml \(storefront\)](/default/components/productbreadcrumbs.isml (storefront))

**#### Local Include**  
Open Template [/default/components/browsertoolscheck.isml \(storefront\)](/default/components/browsertoolscheck.isml (storefront))

**##### Page**  
Open Template [/default/product/product.isml \(storefront\)](/default/product/product.isml (storefront))

### Run the Storefront Toolkit Page Information activity.

Walk the students through using the SFT to see and open page templates being used to generate a page.



To use the Storefront Toolkit Page Information tool, follow these steps:

1. Open a web browser.
2. Enter the URL address of the storefront you wish to troubleshoot in the URL address field of your browser.
3. The Storefront Toolkit control box will be located in the top-left corner of the page. If you do not see the toolkit, roll your mouse over the top-left corner of the browser window and the toolkit will appear.
4. To open the toolkit, click the down arrow of the control box.
5. Select the tool you wish to use from the checkboxes.
6. Select the **Page Information** box.
7. Hover your mouse over the storefront page to view the ISML template being used to generate a page:



11. Click the template link (be sure UX Studio is open) to open the template rendering the area of the page.



# Storefront Files



# Introduction

Some important files used in SiteGenesis for rendering pages to a browser are:

- `htmlhead.isml`, together with `htmlhead_UI.isml` and `footer_UI.isml` contain all of the CSS, jQuery libraries, and js files.

```

1<!--@comment-->THIS FILE ONLY CONTAINS DIFFERENCES BETWEEN SIMPLE AND RICH EYE LITE INCLUDED</!comment>
2<!--comment-->
3<!--@REQ/END: comments are control statements for the build cartridge which can be found in xChange https://xchange.demandware.com/docs/DOC-5728
4If you are not using the build cartridge the comments can be safely removed.
5</!comment>
6
7<!--comment-->TRIND PARTY STELSCHETS STYLING</!comment>
8<link href="/lib/StyleAssets/JS/JS/jquery/ui/jquery.ui.all.css" type="text/css" rel="stylesheet" />
9<link href="/lib/StyleAssets/JS/StyleAssets/CONTEXT_CATALOG_ajax/JS/jquery/ui/jquery.ui.all.css" type="text/css" rel="stylesheet" />
10
11<!-- JQUERY -->
12<script src="/lib/StyleAssets/JS/JS/jquery/jquery-1.7.1.min.js" type="text/javascript"></script>
13<script>var app=jQuery</script>
14<!--@REQ/END: comments are control statements for the build cartridge which can be found in xChange https://xchange.demandware.com/docs/DOC-5728
15If you are not using the build cartridge the comments can be safely removed.
16</!comment>
17</script>
18
19<!--@comment-->Demandware active data scripts</!comment>
20<script>

```

- `app.js` - Contains the Javascript for client-side storefront event handling

```

1 *
2 * All Java script logic for the application.
3 *
4 * The code relies on the jQuery JS library so
5 * will be also loaded.
6 */
7
8 var app = (function($){
9     // jQuery
10    if (!$) {
11        alert('Missing jQuery!');
12        return null;
13    }
14
15    // Global & private data goes here
16
17    // The scope public
18    return {
19        URLS : {}, // holds all specific URLs, where baseURL.html for some examples
20        resources : {}, // resource strings used in js
21        constants : {}, // platform constants, initialized in baseURL.html
22        containerId : "content",
23        ProductCache : null, // app.Product object gnt to the current/main product
24        clearDivHtml : "<div class='clear'></div>", // DOM Clearing -></div>,
25        currencyCodes : {}, // holds currency codes/symbol for the site
26
27        // default dialog box settings
28        dialogsSettings : {
29            iframe: true, // this is required mainly for IE8 where drop down bleed thru
30            autoOpen: false,
31            buttons: {},
32            modal: true,
33            overlay: {}
34        }
35    };
36 })(jQuery);

```

You should open each file to see the functionality being managed by each one.

### IMPORTANT NOTE

Certain ISML templates may include assets designed to support advanced UI functionality. In these cases, those specific includes should be removed and a single include to `<TEMPLATE_NAME>_UI.ISML` is inserted. A blank `<TEMPLATE_NAME>_UI.ISML` is created in Core as a blank file to ensure Core does not break. Then the `<TEMPLATE_NAME>_UI.ISML` file is created in Rich UI. In the Rich UI version, all of the specific element includes will be inserted.

An example of this clearly is in `HTMLHEAD.ISML`. This global include file initializes a host of variables and includes a wide range of support files. Some of these support files are JavaScript based and designed to support UI functionality – such as drop down menus. So based on the above direction, a second file `HTMLHEAD_UI.ISML` would be created. All of the UI includes, variable initializations, etc. are removed and a single line to include `HTMLHEAD_UI.ISML` is added. A blank `HTMLHEAD_UI.ISML` is created in Core and in the Rich UI cartridge a matching `HTMLHEAD_UI.ISML`. In the Rich UI version of the file, all of the removed elements from `HTMLHEAD.ISML` would be inserted.

## Product Detail Page



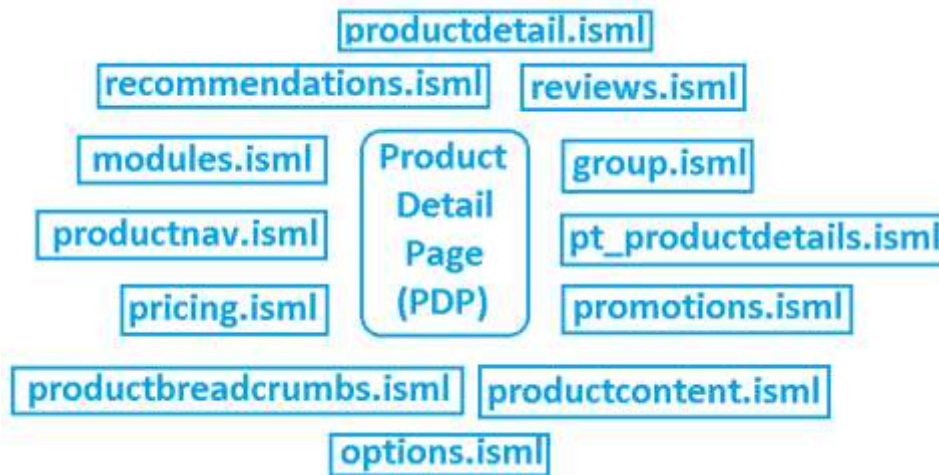
### Introduction

The product detail page (PDP) refers to the page that displays all the information for one specific product. It is the result of the Product-Show pipeline. It is rendered by `product.isml` and decorated by `pt_productdetails.isml`.



### Product Page Templates

There are many templates that are included in a product page:

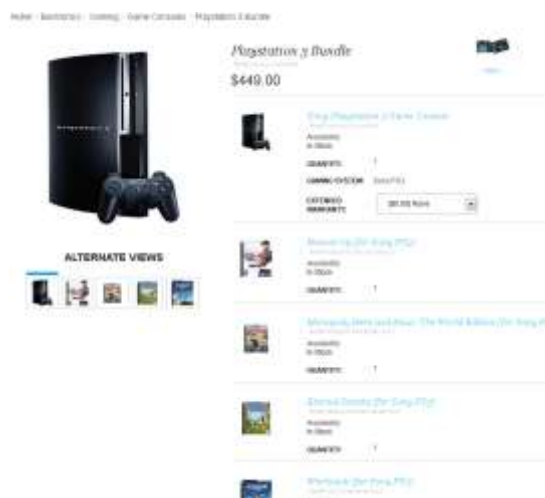


`product.isml` includes the decorator `pt_productdetails.isml` which renders all product pages. `productdetail.isml` decides if it's either a Product Set, Product Bundle or a regular product.

If the product has inventory, pricing, availability and option products, it will be displayed using the respective included files inside `productcontent.isml`, such as `recommendations.isml`, `productbreadcrumbs.isml` (for navigation), `modules.isml` (custom tag definitions), `reviews.isml` and `productnav.isml` (used in decorator) are supporting the appearance on the PDP.

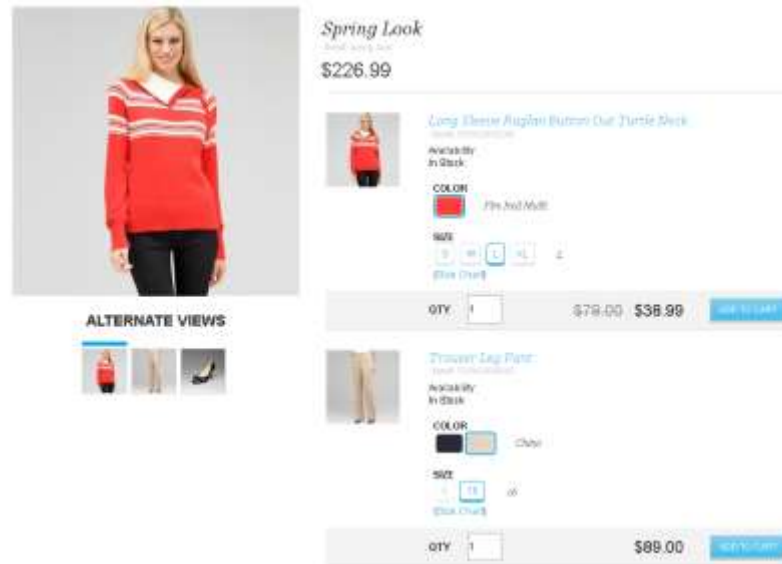
### Bundled Product Page

When a displayed product is a bundle, the page is rendered with `product.isml` which includes the template `producttopcontent.isml`.



### Product Sets

When a product is part of a product set, the page is rendered with `product.isml` and includes the template `producttopcontentPS.isml`.



### Data Manipulation in Product Detail Page

The data for `product.isml` comes from several places:

1. `pdict` provides:
  - a. The Product which identifies the selected product for pricing, recommendations, etc
  - b. `ProductSearchResult` and `ProductPagingModel` for rendering `productnav.isml` and `productbreadcrumbs.isml`:



Home > Womens > Clothing > Bottoms > Relaxed Fit Pant

- c. `CurrentHttpParameterMap.source` to determine if quickview, viewed from cart, or ajax search
- d. `CurrentVariationModel` to build the variation swatches

# Shopping Cart & Checkout Processing



## Introduction

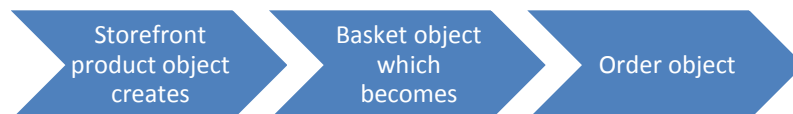
The process of placing an item into a shopping cart and then moving through the checkout process for an order in SiteGenesis is complex and involves **six pipelines** :

- `Cart`
- `COCustomer`
- `COShipping`
- `COBilling`
- `COPlaceOrder`
- `COSummary`

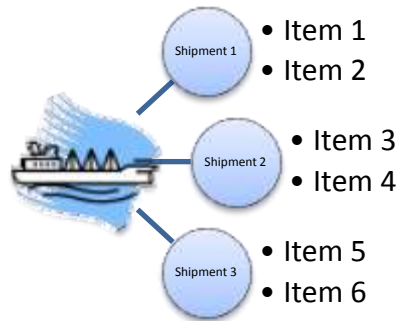
In this module, we will give you an overview of some high level concepts around checkout programming. After reviewing these concepts, we will walk you through the shopping cart and checkout process and explain what each pipeline/sub-pipeline does for each step of the way.

## Checkout Programming Concepts

When a user adds an item to their shopping cart we create a `Basket` object. Later, when the user wishes to finalize their order, we convert this `Basket` object into an `Order` object.



Both of these objects must contain all of our individual items that we need to charge to the user. Because we may want to send different items in the **same order to different addresses**, items are grouped into shipments.



All of these objects inherit from the class `LineItemCtr` (line item container). Let's have a look at what these objects can contain.

## LineItem Class

The base class used to represent any type of item to be included in the calculation of totals is the `LineItem`. There are many different types of line items, but they all serve basically the same purpose. A line item can be thought of exactly like a line item of an invoice which will later be used to sum up all of our invoice (or Basket > Order) totals and subtotals. So we create line items in our basket that represent a quantity of an item on our invoice, be that a shipping cost, coupon to be redeemed or a product to be purchased:

<u>Item Description</u>	<u>Amount Each</u>	<b>Ct.</b>	<b>tax</b>	<u>Totals</u>
Men's Black Leather Shoe 00956TW	\$125.00	<b>1</b>	<b>8.75</b>	<b>\$133.75</b>
Women's Leather Coat 45342VT	\$75.95	<b>1</b>	<b>5.31</b>	<b>\$81.26</b>
Gift Certificate	-\$50.00	<b>1</b>	<b>x</b>	<b>- \$50.00</b>
Shipping	\$7.50	<b>1</b>		<b>\$7.50</b>
<b>TOTAL</b>				<b>\$172.51</b>

Another type of line item is a `PriceAdjustment`. `PriceAdjustments` are used to adjust the price of a particular line item (like a discount or a surcharge.) They can be a positive or negative adjustment (amount.)

`OrderPaymentInstruments` are objects that live at a similar level to the line item, but they are not line items, because they are not items to be charged to the user (and calculated in the total,) but rather payment from the user against the total. A

`PaymentInstrument` represents a personal method of payment. So it does not only represent a Visa Card, it represents John Smith's Visa Card. This item points to a payment method (for example a type of credit card) but also contains personal information such as a card holder name, card number and card expiry date.

`OrderPaymentInstruments` also contain a `PaymentTransaction` object, which specifies the amount to be charged to the payment instrument. It is not dependent on any number or specific line item(s).

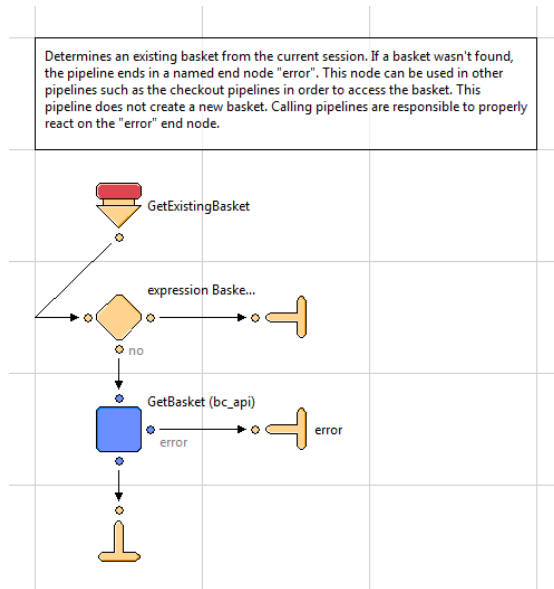
Now that we have had a look at how data is grouped and stored for an order, let's have a high level look at how we enable users to interact with this data.

### Steps for Rendering a Checkout Page

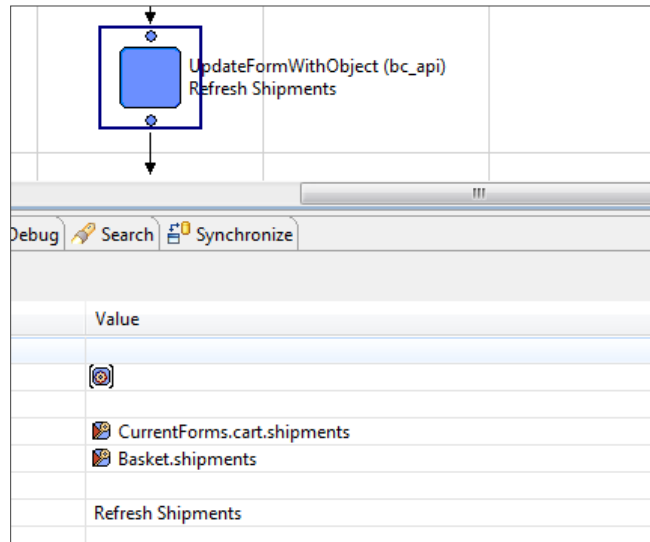
There are three basic steps to rendering a checkout page:

1. Get the most current Basket data associated with the session from the database. This is done by calling the `GetBasket` pipelet. In SiteGenesis a private pipeline for this is available:

Cart-GetExistingBasket sub-pipeline



2. Initialize the forms, pre-filling them with the data retrieved in step 1. This is achieved by using the pipelet `UpdateFormWithObject`. You may specify which dictionary object should be updated with which form group. The individual attributes are updated via mappings defined in the form metadata (bindings.) Later when a user submits data via the forms and having rendered the template, we will leverage the same mechanism with the pipelet `UpdateObjectWithForm` to store the submitted data in the database:



3. Prepare the view. Retrieve any other data from the system that you need for this page. Examples of this could be calculating applicable shipping methods or payment methods. Other examples may be retrieving basket or user specific product recommendation or promotional data. This step needs not be performed in a pipeline, it may also be performed in the isml template, or using ajax calls which is often preferred so that this data can be refreshed in real time as the user makes changes to relevant data on the page.

You now should be prepared to render your checkout page using an ICN (interaction continue node.) It is not required to use an ICN node in checkout but they come with many advantages, such as **automatic form validation** and the concept of form actions which allows you to **create very diverse page flow** without having a lot of public entry points for pages.

Now let's walk through SiteGenesis and see some of these concepts in action.

For this please put a breakpoint on the following Start nodes :

Cart - AddItem  
 Cart - Show  
 Cart - COStart  
 COCustomer - Start  
 COShipping - Start  
 COBilling - Start  
 COPlaceOrder - Start  
 COSummary - Start  
 COSummary - ShowConfirmation



### Shopping Cart


In SiteGenesis, there are two displays of a visitor's shopping cart: a mini-cart popup:



And a full-page cart display that opens when the **View Cart** button is pressed from the mini-cart or the cart link is clicked:

[Home](#) > [Cart](#)

#### Your Shopping Cart

PRODUCT	QTY	PRICE	TOTAL PRICE
<div></div> <div><p><i>Platinum Blue Stripes Easy Care Fitted Shirt</i></p><p>Item No: <b>008884304030</b></p><p>Color: <b>White</b></p><p>Size: <b>6</b></p><p><a href="#">Edit Details</a></p></div>	<div>1</div> <div><a href="#">Remove</a> <a href="#">Add to Wishlist</a> <a href="#">Add to Gift Registry</a> In Stock</div>	\$75.00	\$75.00

ENTER COUPON CODE

[Apply](#)

Update Cart

Subtotal

\$75.00

Shipping

N/A

Sales Tax

N/A

Estimated Total

\$75.00

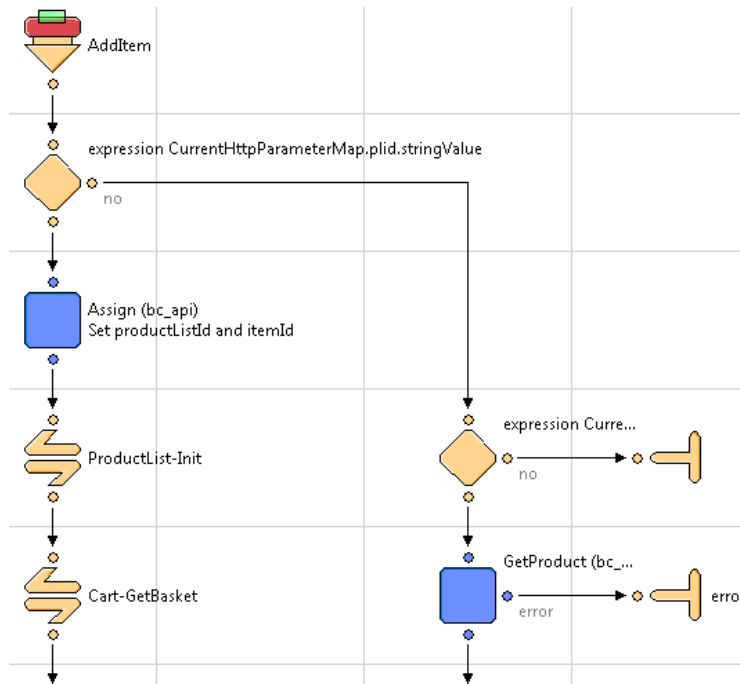
[« Continue Shopping](#)

[CHECKOUT](#)

When a customer puts a product into the basket, the process is typically done with an ajax call. The ajax call invokes the `Cart` sub-pipeline that is located in the `Cart` pipeline.

### AddItem Sub-Pipeline

The AddItem sub-pipeline adds a product to the mini cart. The pipelet AddProductToBasket creates a ProductLineItem (representing the added product sku and quantity) in the basket and also returns this object in the pipeline dictionary.

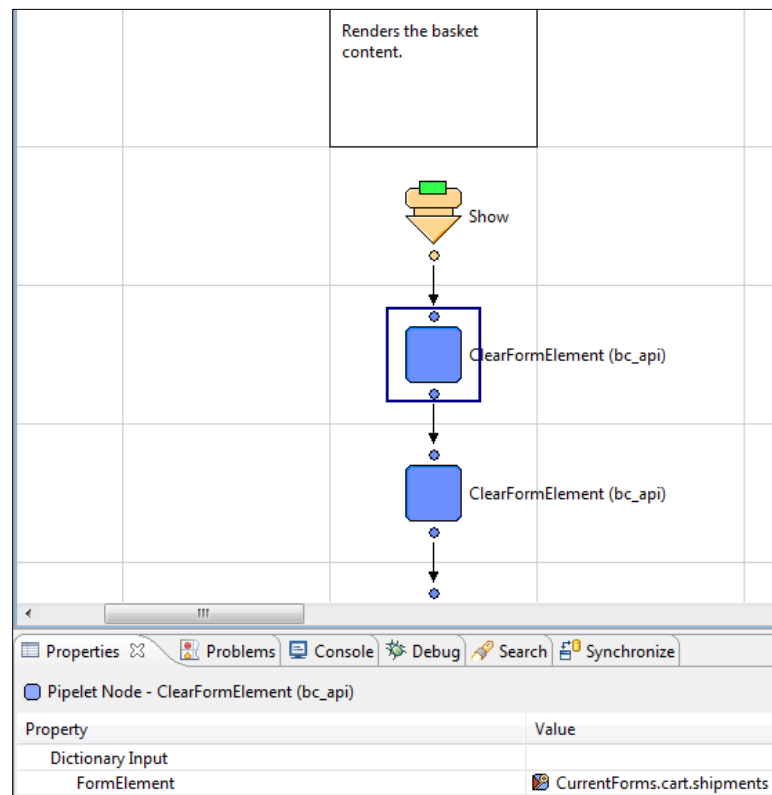


The overall functionality of the AddItem sub-pipeline is as follows:

- Checks whether product is coming from a product list (wish list or gift registry) - if not it makes sure there is a `productID`
- Product object is retrieved
- Checks for a basket object. If one is not present, it is instantiated.
- AddProductToBasket pipelet is called – adds a product to the basket
  - It is important to note that the pipelet creates and returns a `ProductLineItem`, or multiple line items, by creating these object(s) in the basket, representing the desired product sku(s) and quantity.
- Once anything is changed in the basket, a cart calculation is always performed. Updates order totals to represent what buyer has done so far.
- Once the '**Checkout**' button is pressed, the '`Cart-COStart`' pipeline is called.

### Cart-Show Sub-Pipeline

Once the user has added the product from the product detail and is ready to check out, they will click the **View Cart** button in the mini cart panel to get to the full view page. The pipeline that is called is `Cart-Show`. Since this page is the entry point to the checkout process, we must first ensure we have created all our form objects in the pipeline dictionary. We do this by using the `ClearFormElement` pipelet to create our form object defined in our form meta-data under the `CurrentForms` key. The pipelet below will create the empty cart form object in the dictionary.



Below you can now see the skeleton of the cart form object under the `CurrentForms` key. Later we will fill this form with the current data from our `basket` object.

▶ ◆ CurrentDomain	[InternalObject < com.demandware.beehive.core.interna...	
▶ ◆ CurrentForms	[Forms id=20630212]	
▶ ◆ cart	[Form formid=cart]	
▶ ◆ addCoupon	[FormAction formid=addCoupon]	
▶ ◆ calculateTotal	[FormAction formid=calculateTotal]	
▶ ◆ checkoutCart	[FormAction formid=checkoutCart]	
▶ ◆ continueShopping	[FormAction formid=continueShopping]	
▶ ◆ couponCode	[FormField formid=couponCode]	
▶ ◆ coupons	[FormList formid=coupons]	
▶ ◆ error	null	
▶ ◆ logout	[FormAction formid=logout]	
▶ ◆ object	null	
▶ ◆ register	[FormAction formid=register]	
▶ ◆ shipments	[FormList formid=shipments]	
▶ ◆ submittedAction	null	
▶ ◆ triggeredAction	null	
▶ ◆ unregistered	[FormAction formid=unregistered]	
▶ ◆ updateCart	[FormAction formid=updateCart]	
▶ ◆ valid	true	
▶ ◆ sendtofriend	[Form formid=sendtofriend]	
▶ ◆ CurrentHttpParameterMap	[HttpParameterMap id=23730048]	

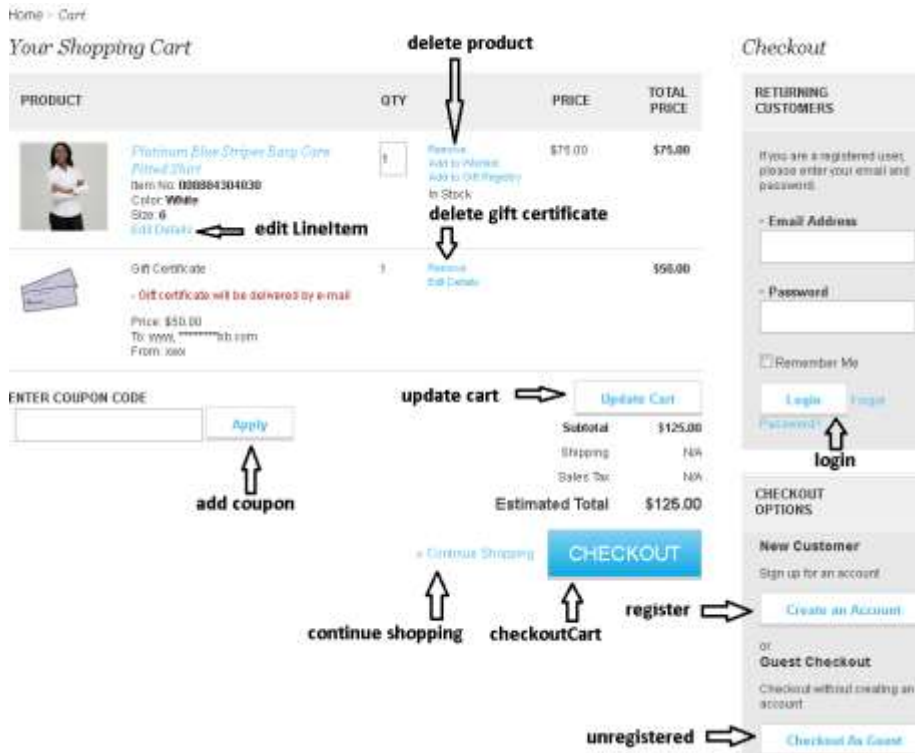
After we guarantee our form objects all exist, we get the current existing basket from the database. Once we have ensured the `basket` object is available in the dictionary, the pipeline `Cart-PrepareView` first updates the form objects with the basket data, and then it retrieves any page specific data we still require. In this case, if the user is a registered user and authenticated, we fetch any wish lists that are associated with the user or create one if none is found. Finally, the page is rendered using an ICN with many continue actions. Let's have a look at where these actions are called on the cart page.

There are many form actions that can be taken from the cart ICN. All of these actions either continue the checkout flow, or perform a business operation and return the user to the cart page. Here are some brief explanations of what these actions do:

- `deleteProduct`: Executes `RemoveProductLineItem` pipelet for the line item, removing it from the cart.
- `editLineItem`: Opens a dialog for the user to edit details (typically select a different variant) and replace the line item.
- `deleteGiftCertificate`: Executes `RemoveGiftCertificateLineItem` pipelet for the line item, removing it from the cart.
- `addCoupon`: Executes `AddCouponToBasket2` pipelet for the input entered into the text field.
- `updateCart`: Typically used in combination with editable quantity fields, this first checks for zero quantities (removing them) and then updates the basket with the `shipments` form object.
- `continueShopping`: Jumps to `Cart-ContinueShopping` which checks for the last visited in the click stream redirecting the user to this page.

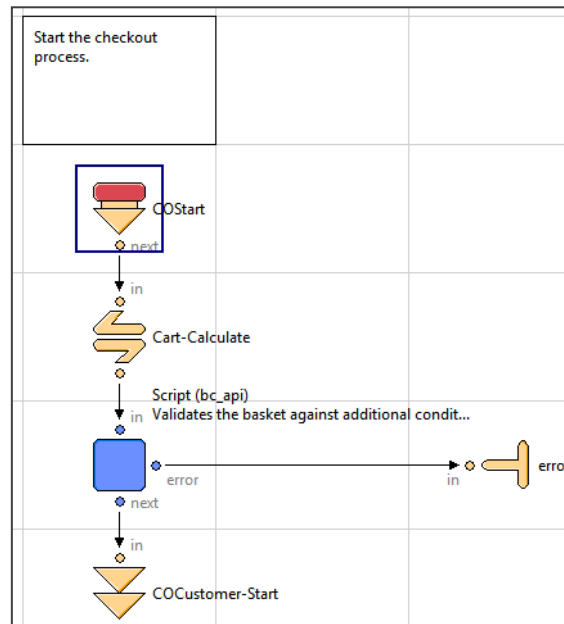
## Instructor Guide

- **checkoutCart**: Calls **Cart-COStart** which prompts the user to decide what type of checkout they would like to make (guest or registered).
- **unregistered**: Jumps directly to the shipping page, the user has indicated they would like to check out as a guest.
- **register**: Takes the user through the registration process, logging them in and returning them to the cart page.
- **login**: The user is logged in and taken to the shipping page



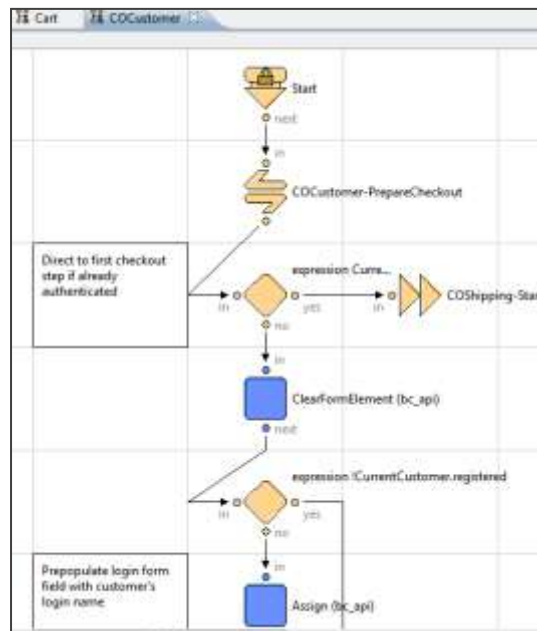
### Cart-COStart Sub-Pipeline

The **COStart** sub-pipeline within the **Cart** pipeline starts the **checkout process**. It validates that the basket total could be calculated (not N/A,) that the products represented by the line items are still available on the site and that any coupons added to the basket are valid. If successful, the pipeline jumps to the **COCustomer** pipeline's **Start** node.



## COCustomer Pipeline

The first step of the checkout process is to provide an option to the visitor to choose checkout type (returning, guest or create account).

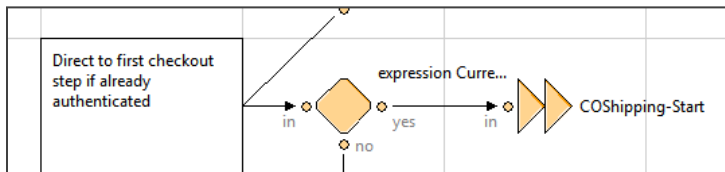


For the purposes of this lesson, we will assume the visitor selects to proceed with an 'unregistered user' checkout process.

### Guest Checkout

Checkout without creating an account

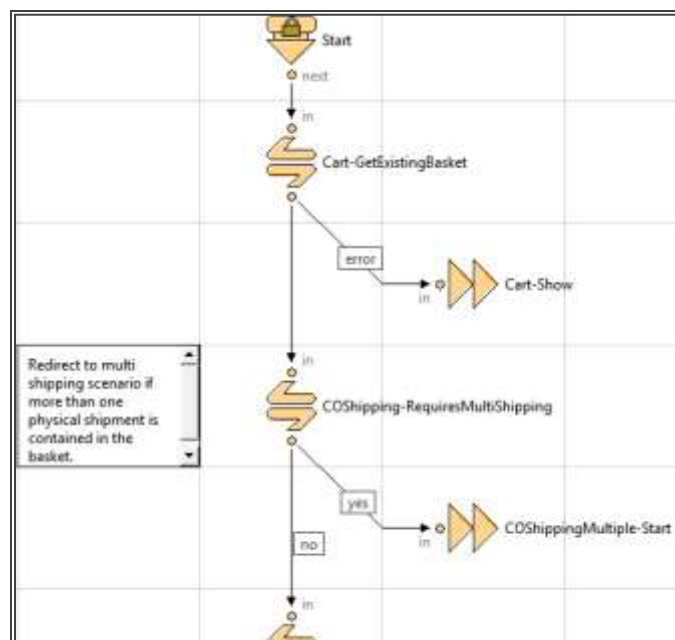
[Checkout As Guest](#)



Next, the 'COShipping-Start' pipeline/Start node is called.

### COShipping Pipeline

The COShipping-Start pipeline **Start** node is the starting point for single shipping scenario.



After performing the steps outlined above, this pipeline displays an ICN that asks for visitor input of where they want their items to be delivered.

The screenshot displays a checkout interface with three steps: STEP 1: Shipping, STEP 2: Billing, and STEP 3: Place Order. The current step is STEP 1: Shipping. The page is titled 'Checkout' and includes a 'Help? 888-555-1234' link.

**SELECT OR ENTER A SHIPPING ADDRESS** (Required)

Fields for shipping address:

- First Name
- Last Name
- Address 1
- Address 2
- Country (Dropdown)
- State (Dropdown)
- City
- Zip Code
- Phone

Example: 212 212 1234

☐ Use this address for billing

Is this a gift? ☐ No ☐ Yes

**SELECT SHIPPING METHOD** (Required)

Shipping methods:

- ☒ Ground \$5.00 (Order received within 3-7 business days)
- ☐ 3 Day Express \$8.00 (Order received in 3 business days)

**ORDER SUMMARY**

Items:

- Example 1: Sneaker Shoes (Quantity: 1, Price: \$50.00)
- Example 2: Sneaker Shoes (Quantity: 1, Price: \$50.00)

Subtotal: \$100.00

Order Discount: -\$0.00

Shipping Total: \$5.00

Taxes: \$0.00

**Estimated \$105.00 Total**

Shipping Method: **SHIPPING DELIVER VIA EMAIL**

It is for one single shipping address only.

Once a shipping address is entered successfully, the UI makes an ajax call to the public pipeline `COShipping-UpdateShippingMethodList`, which calculates the applicable shipping methods for this basket.

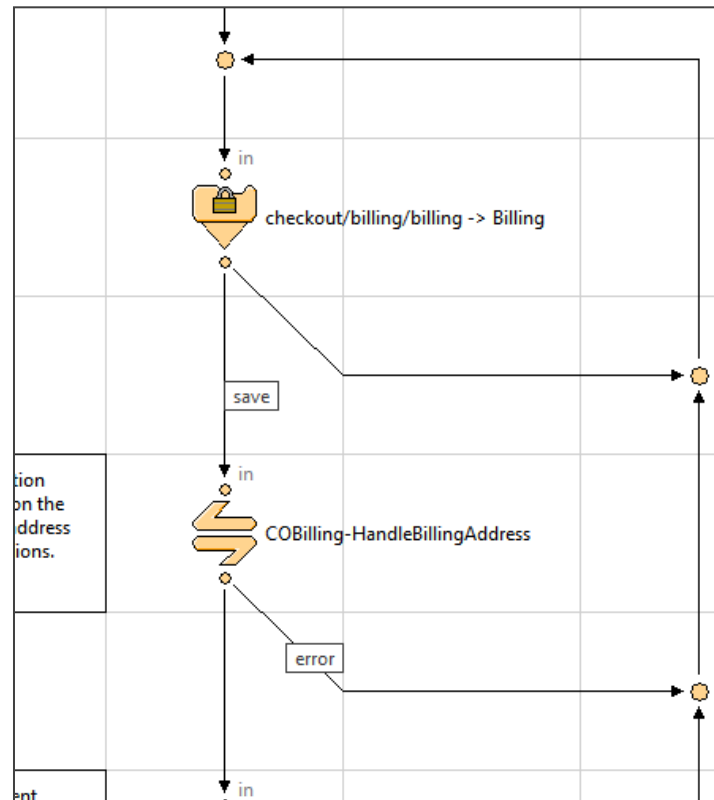
**In Business Manager**, in the Shipping Methods module, **rules and costs can be defined for each shipping method** which will be considered while calculating this list of methods and costs. The interface (view) is then updated with the applicable shipping methods and costs. Once the user has selected an applicable shipping method and chosen to continue checkout, the pipeline reviews the basket and makes sure that there are no empty shipments (if some basket manipulation has been done for example). Next, if there is not already a shipping address, one is created on the shipment. Lastly, the shipping method for the shipment is updated with the selected method. All values are set based on the forms value. The pipeline then exits with a jump node to the '`COBilling-Start`' pipeline/Start node.

### COBilling Pipeline

The `COBilling-Start` pipeline is the starting point for billing. This pipeline implements the billing logic. It is responsible for providing the payment method selection as well as entering a billing address. Again, the same steps outlined above are taken before rendering the template.

The first ICN that is rendered is the Billing form:





If the visitor had selected in the shipping form to use the same address for billing, then the address will be pre-populated from those fields.

Checkout **STEP 1: Shipping** - **STEP 2: Billing** - STEP 3: Place Order [Help? 888-553-9216](#)

**SELECT OR ENTER A BILLING ADDRESS** [+ expand](#)

First Name:

Last Name:

Address 1:

Address 2:

Country:

State:

City:

Zip Code:

Phone:


Example: 333-333-3333

Email:

☐ Please add me to the Demandware email list; Demandware does not share or sell personal info.

**ORDER SUMMARY** [edit](#)

 **Women's Chelsea Boot**  
 Brand: Chelsea  
 Color: Brown  
 Material: Leather  
 Size: 8.5  
 Qty: 1 \$85.00

 **Gift Certificate**  
 Qty: 1 \$100.00

**Subtotal: \$135.00**  
**Order Discount: -\$2.00**  
**Shipping: \$5.00**  
**Taxes: \$4.00**  
**Order Total: \$127.00**

**Shipping #1** [edit](#)

**SHIPPING ADDRESS**

1234 Main Street  
 Jena, AR 87743  
 United States

## Payment Methods Module

Payment methods can be defined in the Payment Methods module of Business Manager. This module gives a merchant control over the payment methods the storefront will accept and under which circumstances.



As part of step 3 (prepare the view) for the billing page, we will retrieve a list of applicable payment methods and cards.

SELECT PAYMENT METHOD + required

☐ Credit Card: ☒ Pay Pal: ☐ Bill Me Later:

Name on Card:   
 Type:   
 Number:   
 Expiration Date:    
 Security Code:

**CONTINUE >**

When the user submits payment information, that payment information is validated against any rules defined in business manager for that payment method/card. If the payment information is valid, a payment instrument is created based on the data submitted in the form. There can be multiple payment instruments since an order can be paid partially with a credit card and partially with a gift card, certificate or even multiple credit cards.

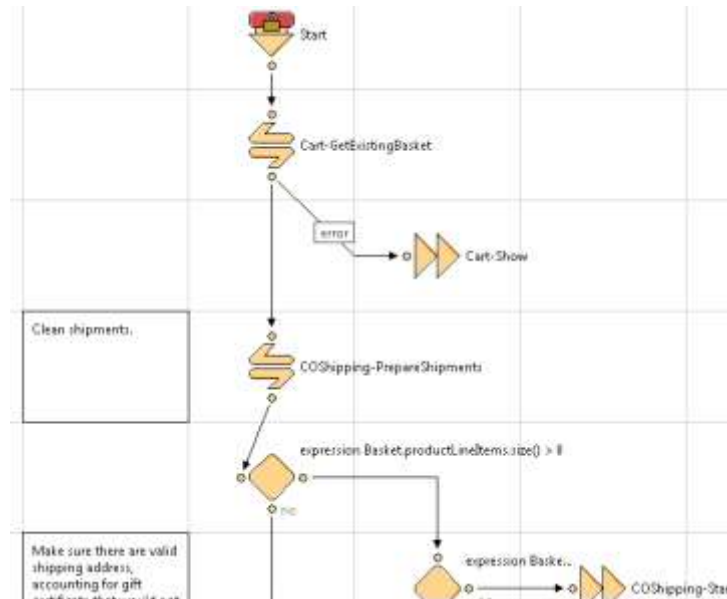
Once the payment method is successfully created, the pipeline exits with a jump node to `COSummary`

### COSummary Pipeline

The `summary.isml` template in the `COSummary` pipeline displays the totals, shipping, billing and payment panels at the right side of the checkout page. Since the user may not interact with data on this page, we only perform steps 1 and 3. Once the **'Submit' button is clicked** by the visitor, the `'PlaceOrder'` action for the button calls the `COPlaceOrder` pipeline.

### COPlaceOrder Pipeline

This pipeline is responsible to create an order from the current basket.



The business process for the pipeline is as follows:

- Gets the existing basket
- Makes sure the shipments are in order
- Calculates the cart
- Validates the payment
- Checks inventory of product line item
- Checks coupons to make sure they are still valid

After these final checks are complete, inventory for the order is reserved using the `CreateOrder2` pipelet and a database transaction is opened.

This pipelet creates an `Order` based on the specified `Basket`. If the order could be created successfully, it will be in status `CREATED`. The `Basket` will be removed from the session and marked for removal.

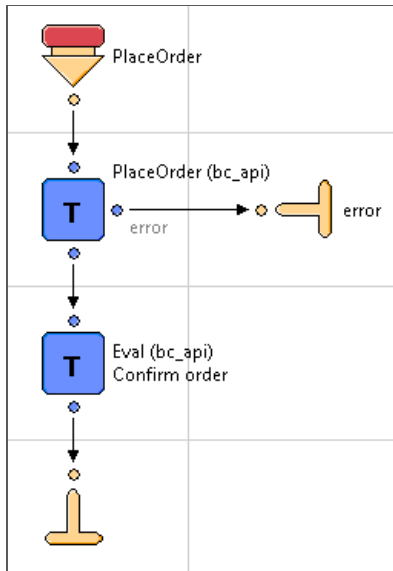
Next is the process of creating an order number and processing the payment.

The processing of the payment method ends with either:

- Authorize
- End
- Decline

- Error

When the payment is authorized, the `PlaceOrder` sub-pipeline is called. The `PlaceOrder` sub-pipeline is responsible for creating the order, and setting the order confirmation status. The pipelet `PlaceOrder` decrements inventory for all products contained in the order. (Means the pipelet `ReserveInventoryForOrder` must not be subsequently called.) The pipelet redeems all coupons contained in the order.



Once the order is placed and the database transaction has successfully been committed, a confirmation email is sent to the buyer. The pipeline will then exit with a jump node to `COSummary-ShowConfirmation` which displays a confirmation ICN node to the buyer that gives the buyer the option of creating an account. If the order creation has failed, the pipeline exits with a jump node back to the order summary page.

## Review & Lab

SiteGenesis Deconstruction Question	True	False
The base class used to represent any type of item to be included in the calculation of totals is <code>LineItem</code>		
The file <code>app.js</code> is used for client-side storefront event handling		
The <code>COPlaceOrder</code> Pipeline is responsible for creating an order from the current basket		



**Transition to Site Maintenance**

## 14. Site Maintenance



---

### Goal

The purpose and goal of this module is to familiarize you with the tools available to you for improving site performance and replicating code and data.



---

### Time

2 Hrs



---

### Overview

We will demonstrate how to use the Pipeline Profiler, discuss how to implement page caching, and discuss and demonstrate how to replicate code and data in the P.I.G.



---

### Materials Needed

None

---

# Site & Page Caching



## Introduction

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.

In Demandware, caching is controlled on a per page basis, via the ISML template for the page. Caching is set on a page using the `<iscache>` tag:

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

These are the 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

## Caching Parameters

The following parameters can be set with the `<iscache>` tag:

- **status = "off|on"**
  - ♦ `off` disables page caching
  - ♦ `on` enables page caching (the default)

This setting allows for caching to be turned on and off programmatically. `status="off"` is considered the shortest duration, so be careful when using it on an included template (see rules above).

- **type = "relative | daily"**

Relative allows you to specify a certain period of time, in minutes and hours, after which the page will be deleted from the cache. Daily allows you to specify a specific 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"**

Lets you 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:

- The defined caching time is exceeded, or
- A replication has been performed (with the exception of coupons and geolocation data), or
- 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:

- Has to 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

Demandware provides caching metrics under the **Site ⇒ Analytics ⇒ Technical Reports** menu. The **Pipeline Performance** report will capture information as follows:

Pipeline	Call Count	Includes	Total	%	Processing Time (ms)							Total	Avg	Caching
					Avg	< 100	1000	3000	5000	10000	100000			
Page-Show	9,819	0.0	218,890	39.10%	22	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	218,890	22	
Default-Start	1,273	0.0	69,717	16.74%	32	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	69,717	32	
Analytics	4,274	0.0	48,538	13.64%	11	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	48,538	11	
Tracking	229	1.0	25,705	8.91%	117	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	25,705	117	
Search-														
ShowContent	318	0.0	4,317	1.13%	8	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4,317	8	
Page-Include	38	2.8	2,791	0.75%	100	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2,791	100	
Search-Show	32	0.0	2,749	0.74%	55	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	2,749	55	
Home-Show	32	0.0	685	0.19%	21	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	685	21	
Search-														
HomePage	14	0.0	546	0.15%	39	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	546	39	
SiteMap-Search	1	0.0	89	0.02%	89	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	89	89	
Link-Page	3	0.0	52	0.01%	21	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	52	21	
Cart-Product@Product	3	0.0	52	0.01%	21	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	52	21	

These types of analytics are not collected on sandboxes, only on Production instances. While this chart is hard to read, 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  $\leq$  to this value to be in a good performance range.

For Product-Show the average response is 320ms-400ms. Customers should be  $\leq$  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 below 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
```



### 14.1. Exercise: Page-Level Caching

1. Create a new pipeline and call it `CachedPage`. Add a **Start** node and an **Interaction** node.
2. Create an ISML template that has caching enabled for 30 minutes:  

```
<iscache type="relative" minute="30" />
```
3. Add a `Date` object to the page that prints the current time:  

```
<isprint value="{new Date()}" style="DATE_TIME" />
```
4. Assign the template to the **Interaction** node in your `CachedPage` pipeline.
5. Test the template in your SiteGenesis storefront. Refresh your page. Does the time change on refresh?
6. 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

Most of the time, a single page should not be cached completely. Some parts of the page should be cached, while not others. 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.



### 14.2. Exercise: Partial Page Caching

1. In the template for the `CachedPage` pipeline, add a remote include call to the `Product-IncludeLastVisited` subpipeline:

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

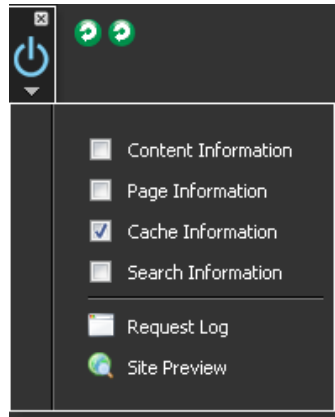
2. Invalidate the cache in Business Manager.
3. Go to another browser window and visit a few products (3 at most).
4. Refresh the `CachedPage-Start` pipeline.
5. Visit more products on the other browser.

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

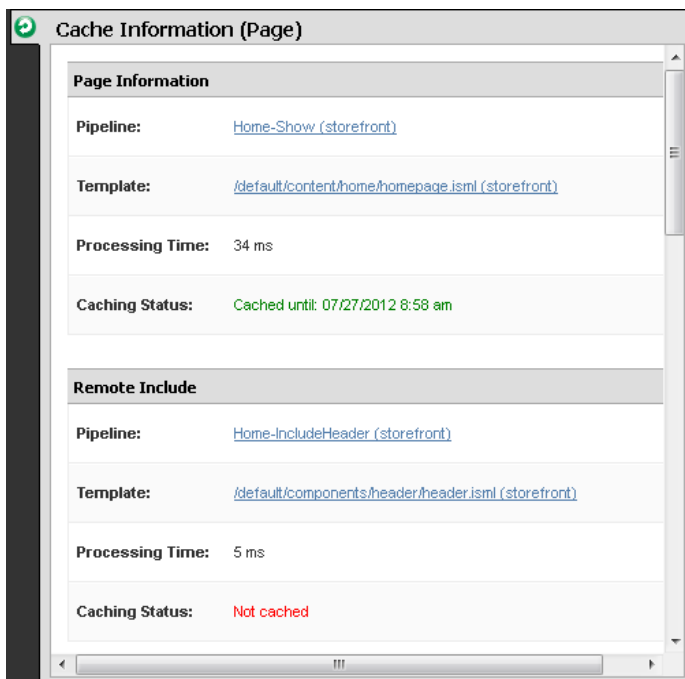


### Using the Storefront Toolkit to Determine Cache Settings

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:





### 14.3. Exercise: Using 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?

# Site Performance



## Introduction

The Pipeline Profiler is a tool in Business Manager that gives you insight into pipeline and script performance.

This tool helps you track pipeline execution metrics, which is a critical component of overall page and site load and performance. Using this tool enables you to proactively identify bottlenecks in performance while developing applications.

It is located in **Administration > Operations > Pipeline Profiler**.



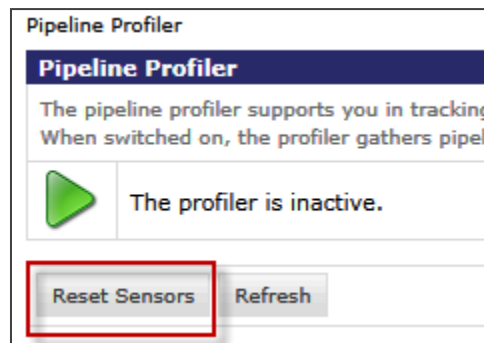
## Run the Pipeline Profiler activity.

Demonstrate how to use the Pipeline Profiler.



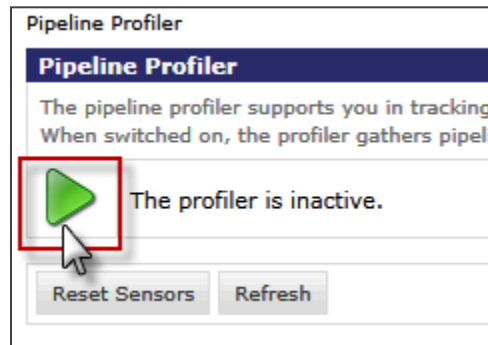
To track the performance of a pipeline using the Pipeline Profiler, follow these steps:

1. Log into Business Manager.
2. Click **Administration > Operations > Pipeline Profiler**
3. Reset previously collected statistics

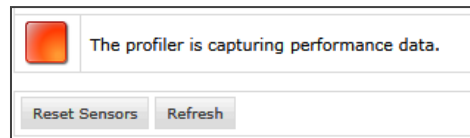


4. Turn on Pipeline Profiler





Profiler Off



Profiler On

5. Browse specific pipeline in storefront
6. Return to profiler and analyze the collected data
  - a. Click the site link you are capturing data for:



- 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.

Pipeline Name	Pipeline Start Node	Hits	Total Time	Average Time	Minimum Time	Maximum Time
SiteNavigationBar	CreateSiteNavigationBar	1	4	4	4	4
OutContent	Do	1	0	0	0	0
SiteNavigationBar	IncludeFrameBottom	1	1	1	1	1
SiteNavigationBar	IncludeFrameTop	1	63	63	63	63
Prefix	Start	5	11	2	1	3
IncludeGlobalNavigationBar	Start	1	6	6	6	6

- 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.

## Instructor Guide

**Profile - Pipeline Performance**

The Pipeline Performance Detail page displays the results of the performance profiling for a selected pipeline. All values are displayed in milliseconds.

**SiteNavigationBar**

**Overall Pipeline Performance (including pipeline and template run times)**

Pipeline Name	Start Node Name	HRs	Total Time	Average Time	Minimum Time	Maximum Time
SiteNavigationBar	IncludeFrameTop	3	123	61	60	63
SiteNavigationBar	CreateSiteNavigationBar	2	12	6	4	8
SiteNavigationBar	IncludeFrameBottom	2	3	1	1	2

**Subpipelines called directly from pipeline SiteNavigationBar**

Pipeline Name	Start Node Name	HRs	Total Time	Average Time	Minimum Time	Maximum Time
SiteNavigationBar	Prefo	2	4	2	1	3

**Pipeline Performance**

Pipeline Name	Pipeline Node ID	HRs	Total Time	Average Time	Minimum Time	Maximum Time
SiteNavigationBar	IncludeFrameTop.1 (CPipelineNodeCustomizerPreference.1	2	8	8	8	8
SiteNavigationBar	IncludeFrameTop.1 (CPipelineNodeGetVersionInformation.1	2	8	8	8	8
SiteNavigationBar	IncludeFrameTop.1 (CPipelineNodeGetQuotaReport.1	2	8	8	8	8
SiteNavigationBar	IncludeFrameTop.2 (CPipelineNodeGetDatabaseUsage.1	2	165	52	51	54
SiteNavigationBar	IncludeFrameTop.2 (CPipelineNodeCheckPipelinePermissions.1	2	7	3	3	4
SiteNavigationBar	IncludeFrameTop.2 (CPipelineNodeSetDictionaryValue.1	2	8	8	8	8

7. Test the pipeline or a different one again
8. While the pipeline profiler runs you have also access to captured script data.

### Browse Captured Script Data

#### Script Data

9. Turn off the profiler and analyze the results
10. If you make modifications to the pipeline, retest to verify if performance has improved

# Code Replication

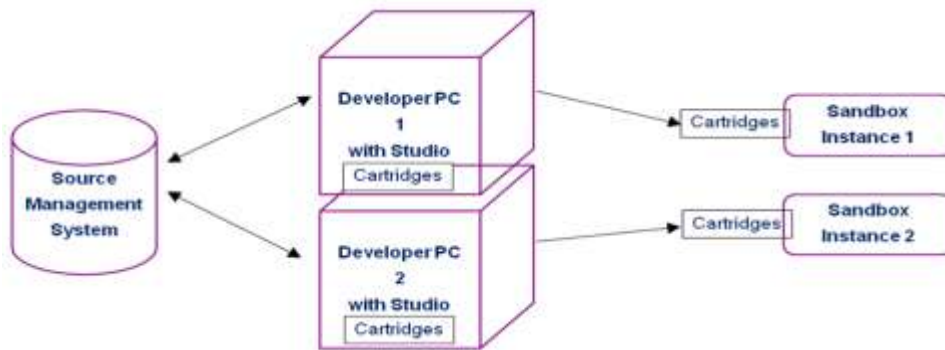


## Introduction

Code replication is set and managed in Business Manager. Once you have uploaded a new code version to the P.I.G. 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.



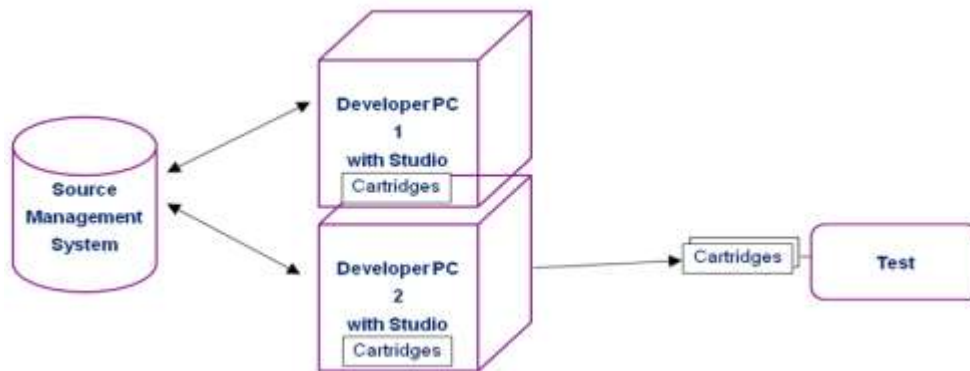
UX Studio integrates with SVN for source management. To learn more about using SVN in UX Studio, view our online webinar in XChange:

<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 the: **Administration > Site Development > Code Deployment** page.

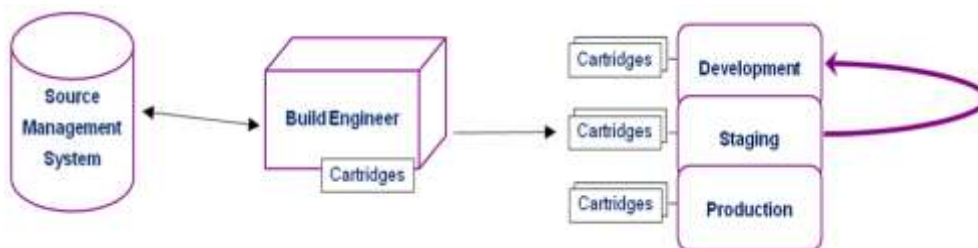
Next, the developer uploads custom cartridges with Studio or WebDAV client using 2-factor authentication and tests the storefront in STAGING. A rollback to a previous version is available.

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



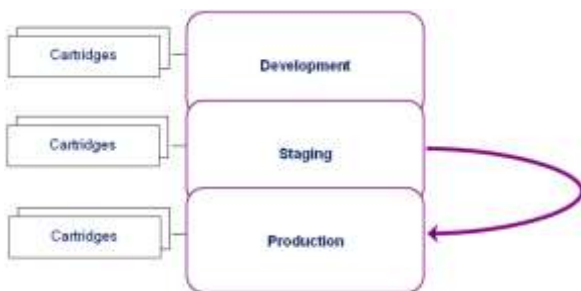
For testing in a sandbox, you will need to export site data to the global directory from staging and import it into your sandbox using the Site Import/Export module in Business Manager.

When code meta data (site preferences, new attributes, etc.) needs to be tested, a replication from STAGING to DEVELOPMENT should be performed by a build engineer:



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.



This process is also performed in Business Manager.

Data replication will be covered in the next lesson.



## Run the Code Replication activity.

Using the Training STAGING instance, demonstrate how a developer would replicate code between STAGING and DEVELOPMENT.



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. Click **Administration**→**Replication**→**Code Replication**.



3. Click the **New** button to create a new replication process.
4. From the **Target** drop-down menu, specify whether the replication process is to DEVELOPMENT or PRODUCTION.

<b>Process ID:</b>	1307236313540
<b>Target:</b>	Development (http://207.211.90.23:80) ▼
<b>Description:</b>	<div>Development (http://207.211.90.23:80)</div> <div>Production (http://207.211.90.22:80)</div>

5. Select whether you want to process to run manually or automatically.

<b>Activation:</b>	Manual ▼
<b>Notification:</b>	<div>Manual</div> <div>Automatic</div>

6. Click **Next**.
7. At the next screen, specify what type of replication you want:

## Instructor Guide

Code Replication Processes > Replication Process Step 1 - General > Replication Process Step 2 - Replication Type

**1348754456822 - Replication Type**

Step 2 of 3. Select the replication type for the new replication process and click **Apply**.  
Select the code version to transfer from the list below. If no selection is made the active version will be transferred by default.  
Click **Next** to review your settings and initiate the process. Click **Cancel** to go back to the list of replication processes.

Replication Type: Code Transfer & Activation

Select	Active	
<input checked="" type="radio"/>	✓	Active version
<input type="radio"/>		SG2
<input type="radio"/>	✓	version1

<< Previous

- a. Code Transfer & Activation: will immediately activate the new code version.
  - b. Code Transfer: Will only transfer the code.
8. Click **Next**.
  9. Click **Start** to start the replication process. Click **Create** to add the replication process to the list.
  10. If you selected the process to run manually, you will need to start the job from the list by clicking **Start**:

Select All	Process ID	Target	Type	Start Time	End Time	Status	
<input type="checkbox"/>	1307236311540	Development	Transfer			Waiting	<b>Start</b>

# Data Replication

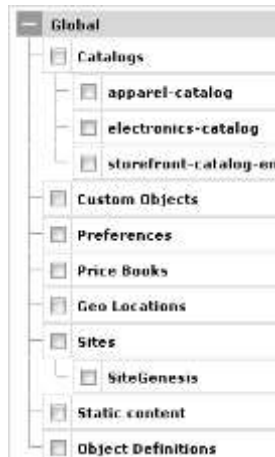


## Introduction

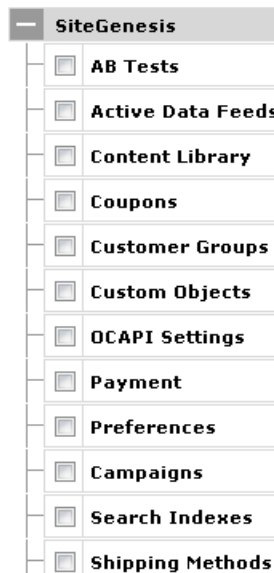
Data replication is a process to promote merchant edits, product and system objects from Staging to Production (or Development). The best practice is to replicate to development first, verify that data and storefront work and then replicate from staging to production.

Data can be replicated granularly:

- Organization objects



- Per Site objects



A Data Replication process consists of two phases:

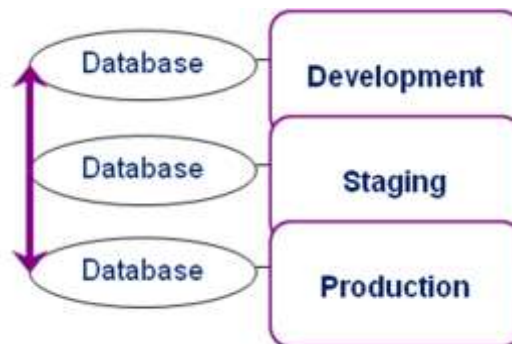
- Transfer – long running processes where data is copied from STAGING into “shadow” tables and folders on PRODUCTION. No changes are shown in storefront.
- Publishing – Very fast process. Changes in “shadow” tables and folders become active, page cache is purged, and the new version is shown in storefront.

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>.



### Run the Data Replication activity.

Using the training Staging instance, demonstrate how developers would replicate data between Staging and Development.





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. Click **Administration**→**Replication**→**Data Replication**.
3. Click **New** to create a new data replication process.
4. Specify the target for the data replication process.

<b>Target:</b>	Development (http://207.211.90.23:80) ▼
<b>Description:</b>	Development (http://207.211.90.23:80) Production (http://207.211.90.22:80)

5. Select whether you want to process to run manually or automatically.

<b>Activation:</b>	Manual ▼
<b>Notification:</b>	Manual Automatic

- a. If automatically, specify the date and time the process should run.

<b>Activation:</b>	Automatic ▼	Start on	06/04/2011	...	9:11 pm
--------------------	-------------	----------	------------	-----	---------

6. Specify when you want an email notification and who should receive it.

<b>Notification:</b>	When Process ends ▼	to	aashline@demandware.com
----------------------	---------------------	----	-------------------------

7. Click **Next**.
8. At the next screen, specify what type of replication you want:

<b>Replication Type:</b>	Data Transfer & Publishing ▼
<b>Replication Tasks</b>	Data Transfer & Publishing Data Transfer
Global	

5. Next, expand the sites to select the site data you wish to replicate.

SiteGenesis	
<input type="checkbox"/>	<b>AB Tests</b> All AB tests and contained test experiences.
<input type="checkbox"/>	<b>Active Data Feeds</b> All active data feed definitions.
<input checked="" type="checkbox"/>	<b>Content Library</b> All library content including content assets, folders and library static content.
<input type="checkbox"/>	<b>Coupons</b> Coupon configurations and single coupon codes.
<input type="checkbox"/>	<b>Customer Groups</b> Definition of customer groups.
<input checked="" type="checkbox"/>	<b>Custom Objects</b> Site specific custom objects.
<input type="checkbox"/>	<b>OCAPI Settings</b> The OCAPI settings for this site.
<input checked="" type="checkbox"/>	<b>Payment</b> Payment processors and their preferences, payment methods and payment cards a
<input type="checkbox"/>	<b>Preferences</b> Site specific system and custom preferences including assignments to catalogs, pri
<input checked="" type="checkbox"/>	<b>Campaigns</b> Campaigns and promotions.
<input type="checkbox"/>	<b>Search Indexes</b> Search indexes for products, spelling, content, synonym, redirect and suggest (ava
<input checked="" type="checkbox"/>	<b>Shipping Methods</b> All shipping methods.

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

## Review & Lab

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