
*Student Guide*

# Developing for Commerce Cloud B2C Commerce

CCD-101

# Table of Contents


# Developing for Commerce Cloud B2C Commerce (CCD-101)

# Developing for Commerce Cloud B2C Commerce (CCD-101)

## Audience

- Application Developers, Solution Architects, System Integrators

## Prerequisites

- Development experience with JavaScript, HTML, XML, CSS, jQuery, and Java
- Experience working with e-commerce applications recommended
- Completion of these courses:
  - GEN001: B2C Commerce Overview
  - DEV001: B2C Commerce Architecture Overview
  - CCM-101: Managing the Storefront (recommended)
- Eclipse installation completed
- Familiarity with Eclipse IDE a plus

# Copyright

# Forward Looking Statements



## Statement under the Private Securities Litigation Reform Act of 1995:

This document and other items we publish, including through social media outlets, may contain forward-looking statements, the achievement or success of which involves risks, uncertainties, and assumptions. If any such risks or uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make.

The risks and uncertainties referred to above include – but are not limited to – risks associated with possible fluctuations in our financial and operating results; our rate of growth and anticipated revenue run rate, including our ability to convert deferred revenue and unbilled deferred revenue into revenue and, as appropriate, cash flow, and our ability to grow deferred revenue and unbilled deferred revenue; errors, interruptions or delays in our service or Web hosting; breaches of our security measures; the financial impact of any previous and future acquisitions; the nature of our business model; our ability to continue to release, and gain customer acceptance of, new and improved versions of our service; successful customer deployment and utilization of our existing and future services; changes in our sales cycle; competition; various financial aspects of our subscription model; unexpected increases in attrition or decreases in new business; our ability to realize benefits from strategic partnerships; reliance on third-party computer hardware and software; the emerging markets in which we operate; unique aspects of entering or expanding in international markets; our ability to hire, retain and motivate employees and manage our growth; changes in our customer base; technological developments; regulatory developments; litigation related to intellectual property and other matters, and any related claims, negotiations and settlements; unanticipated changes in our effective tax rate; factors affecting our outstanding convertible notes and credit facility; fluctuations in the number of shares we have outstanding and the price of such shares; foreign currency exchange rates; collection of receivables; interest rates; factors affecting our deferred tax assets and ability to value and utilize them, including the timing of achieving profitability on a pre-tax basis; the potential negative impact of indirect tax exposure; the risks and expenses associated with our real estate and office facilities space; and general developments in the economy, financial markets, and credit markets.

Further information on these and other factors that could affect the financial results of salesforce.com, inc. is included in the reports on Forms 10-K, 10-Q and 8-K and in other filings we make with the Securities and Exchange Commission from time to time, including our most recent Form 10-K. These documents are available on the SEC Filings section of the Investor Information section of our website at www.salesforce.com/investor.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make their purchase decisions based upon features that are currently available.

# Course Agenda



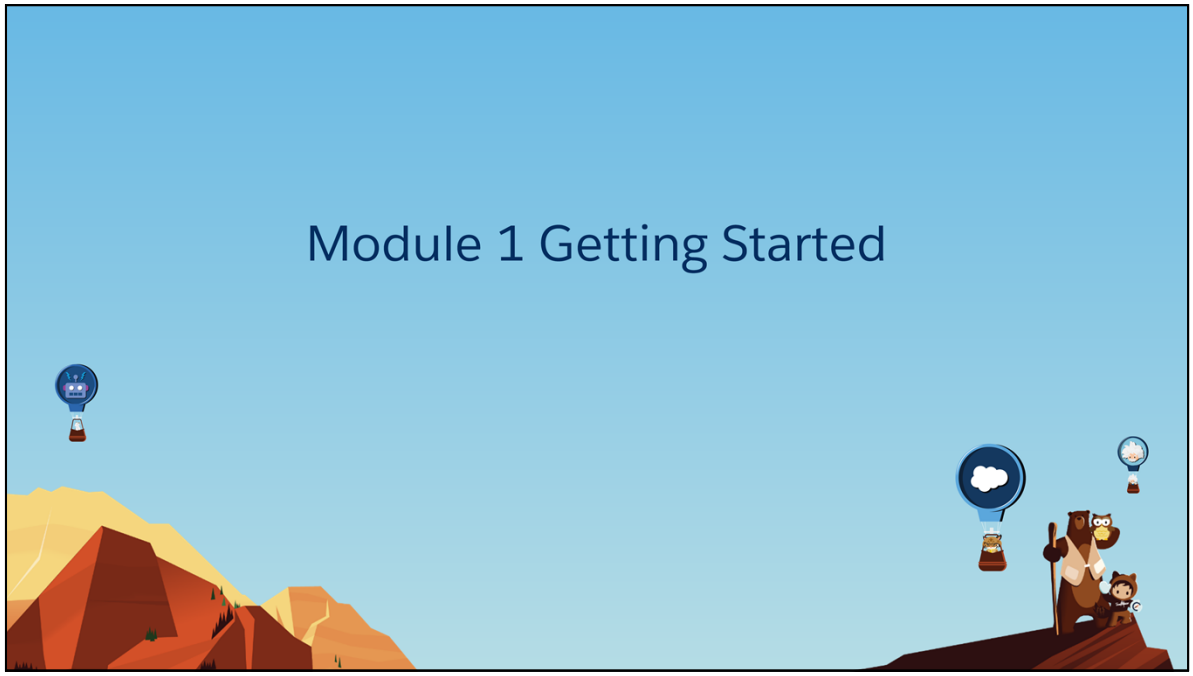
**DAY 1**
- Module 1: Getting Started
- Module 2: UX Studio
- Module 3: Cartridges
- Module 4: JavaScript Controllers

**DAY 2**
- Module 4: JavaScript Controllers (cont.)
- Module 5: ISML
- Module 6: Content Slots

**DAY 3**
- Module 7: Commerce Cloud B2C Commerce Script
- Module 8: Forms Framework
- Module 9: Custom Objects

**DAY 4**
- Module 10: Data Binding and Explicit Transactions
- Module 11: Site Maintenance
- Appendix A: Pipelines
- Exam information and tips

# Module 1 Getting Started



## Module 1: Getting Started

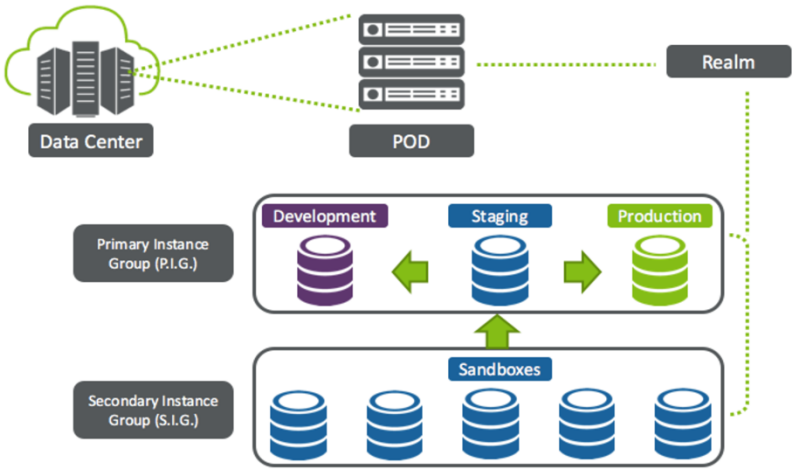


**Objectives:**

- Describe the key components of the Commerce Cloud platform.
- Create a new empty site.
- Import a copy of SiteGenesis into a site and configure its settings.

**Lessons:**

1.1 B2C Commerce Overview
1.2 SiteGenesis Overview
1.3 Site Configuration




B2C Commerce Architecture

A realm contains segmentation for development, staging, and production for one or more storefronts.

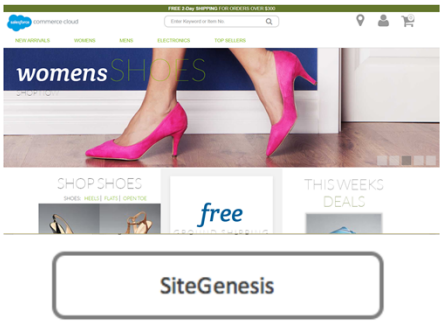
Every Realm includes a Primary Instance Group (PIG) which includes three Commerce Cloud B2C Commerce instances:

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

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

## Business Manager






SiteGenesis

Control Center

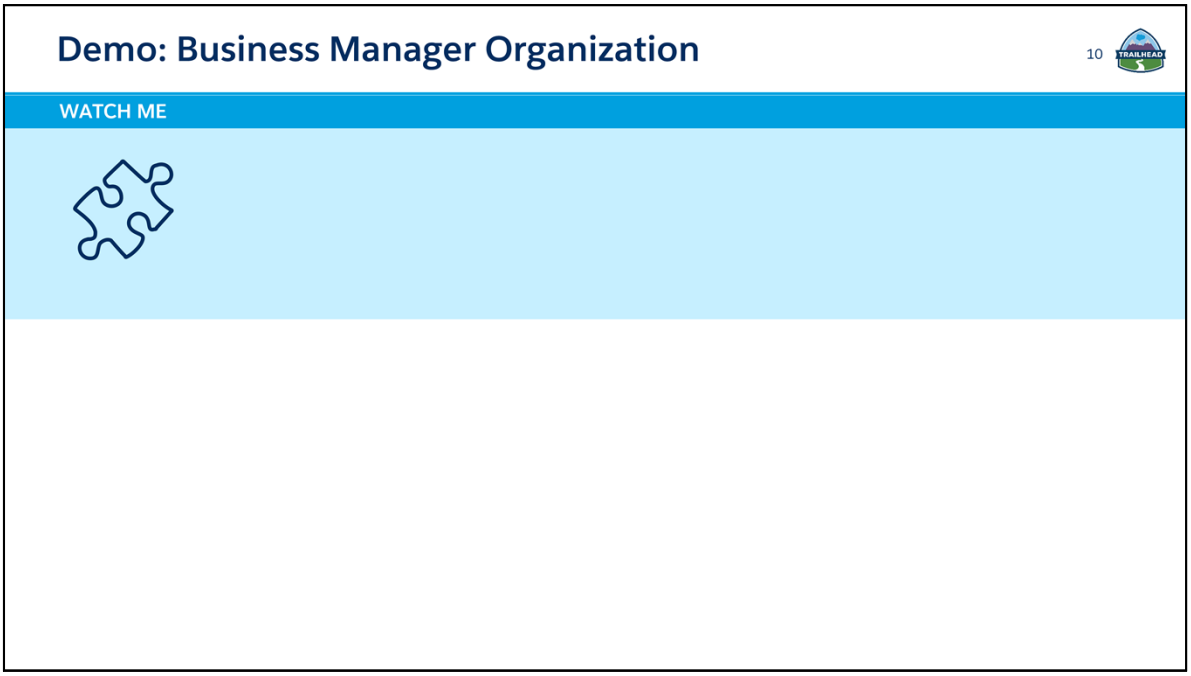
Business Manager

UX Studio

Both merchants and developers use Business Manager to manage administrative tasks. Every B2C Commerce instance has a Business Manager portal. For example, a merchandiser would log into Business Manager in the Staging instance.

Merchants use Business Manager to manage:
- Products & Catalogs
- Content
- Marketing campaigns
- Search settings
- Customers
- Site Analytics
- Site URLs
- Site Preferences

Developers use Business Manager to manage:
- Code & Data Replication
- Code Versioning
- Site Development
- Data Import/Export
- Global Preferences for all sites/organization



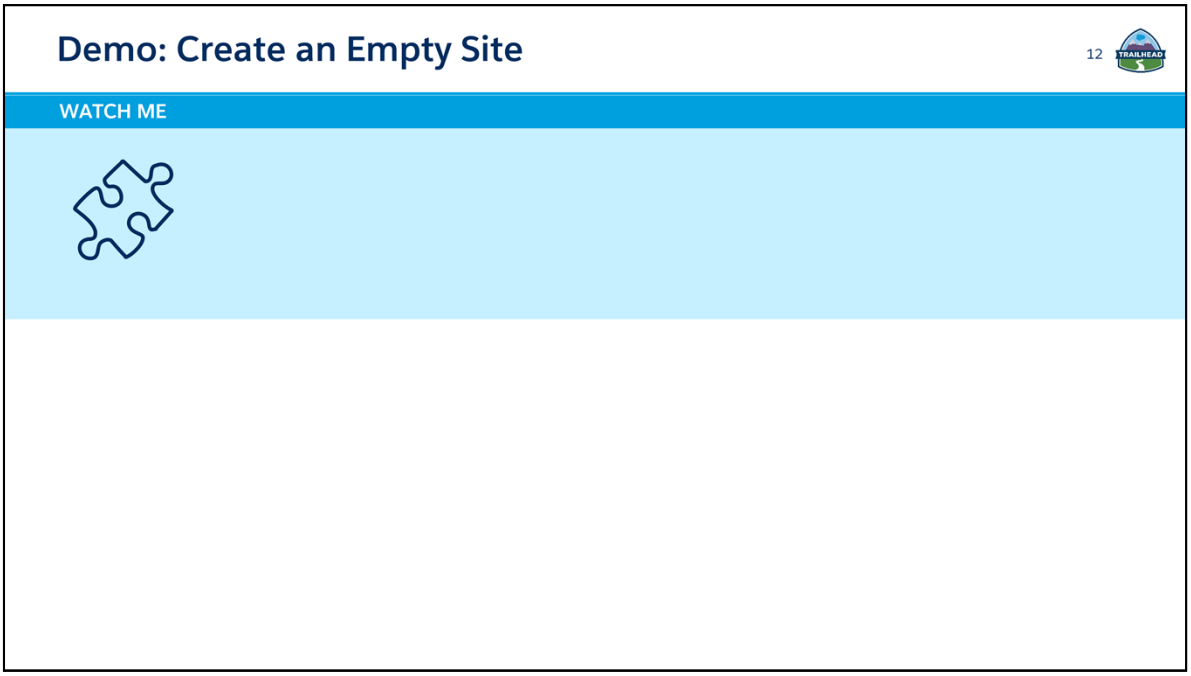

1. In Business Manager, click each of the Merchant menu items in SiteGenesis to determine the main tasks in Business Manager. These are located on the left side under **Site – SiteGenesis**.
2. Click the **Administration** menu items to determine the main tasks in Business Manager.

## Deploy a Storefront

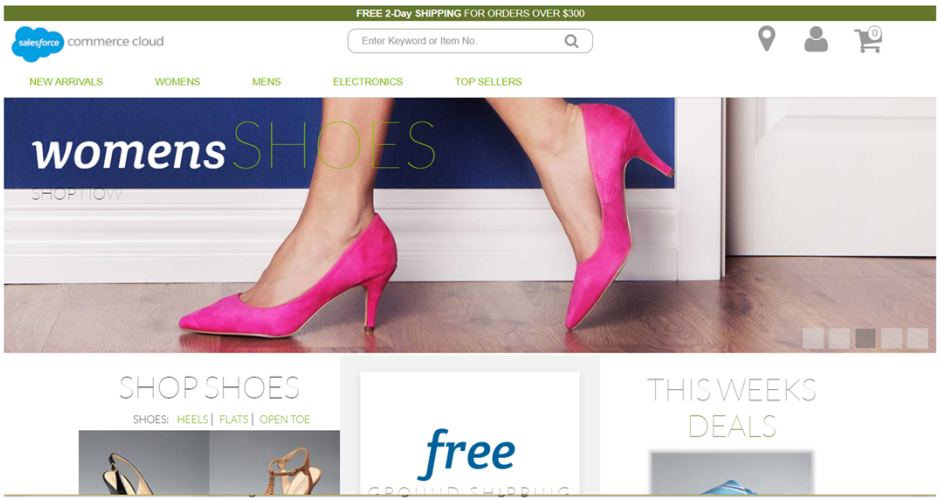


When you first log into Business Manager for a given instance, by default no storefront has been deployed. You must either:

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




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




Commerce Cloud includes the SiteGenesis sample reference site, which you can use as the basis of your custom Commerce Cloud B2C Commerce sites. It is a full featured demonstration eCommerce site, which you can use to explore the B2C Commerce platform and its capabilities. SiteGenesis is a resource for both developers and merchants:

- For developers, it provides sample code–scripts, and ISML templates.
- For merchants, it provides sample configurations for catalogs, categories, products, and so on.

**Notes on Importing SiteGenesis**

Import the current version of the SiteGenesis package (read-only) as a sample site into every Sandbox instance.

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

- It is recommended that you import SiteGenesis into an **empty** sandbox **before** importing your custom sites. The prevents you from overwriting existing attributes and data for the custom site. After importing SiteGenesis, you can validate its behavior by comparing it to the site running on the demo instance.


**Import the latest version of SiteGenesis**

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

**Import a site from a local copy**

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

**Import from a remote server**

1. Enter all required data for accessing the remote server account, including the Hostname, Login, and Password. Click **Connect**.
2. You can view the importable files from the remote server. Select the radio button next to the name of the import file you want to use.
3. Click **Import**.
4. A confirmation message displays. Click **OK**.
5. You can view the status of the import in the **Status** section of the page.

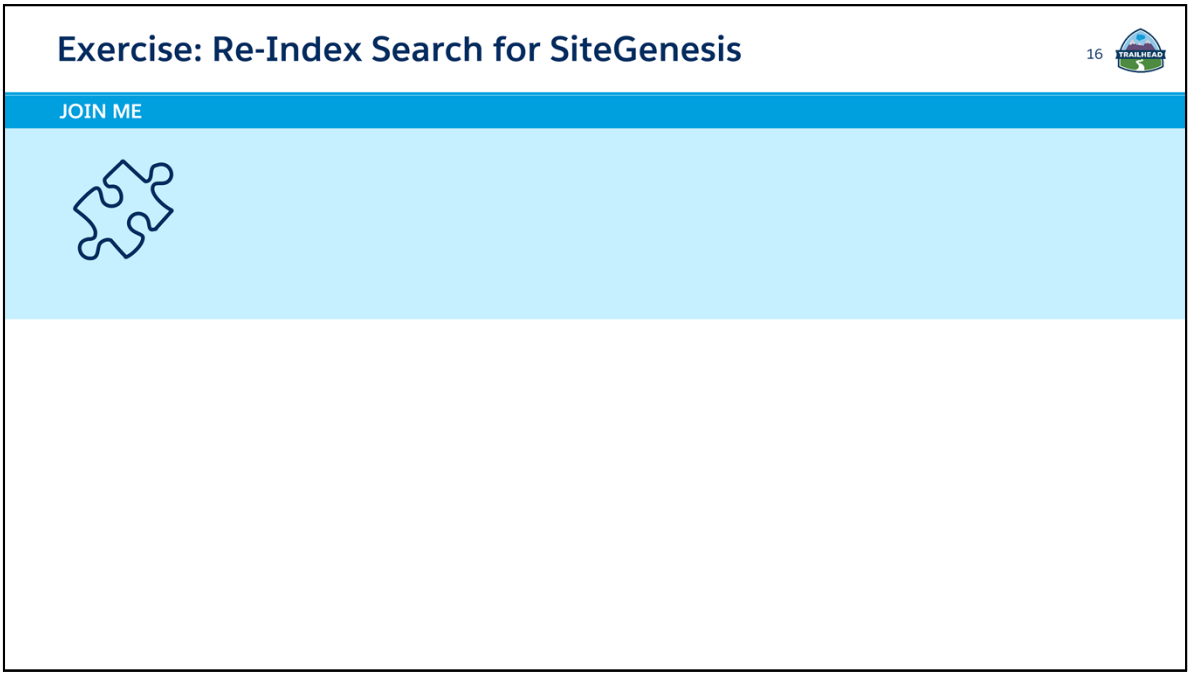
When your import has completed, Business Manager lists the new site. You will also receive an email that the job is complete.

## Lesson 1.3: Site Configuration




After creating an empty site, disable site caching in your sandbox to see your code changes immediately in the site. This prevents the page cache from taking effect, so that pages reflect the most recent code changes. In production instances the cache is on by default.

You need to index the site to be able to search for products from the storefront.




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

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

## Knowledge Check 1.1



**True or false:**

1. A Realm is a B2C Commerce instance used only by developers.
2. Merchants use Business Manager to manage products and catalogs.
3. You can import SiteGenesis through site import at any time without risk.
4. Catalogs are not shareable between sites within an organization.
5. A site must have one and only one site catalog.

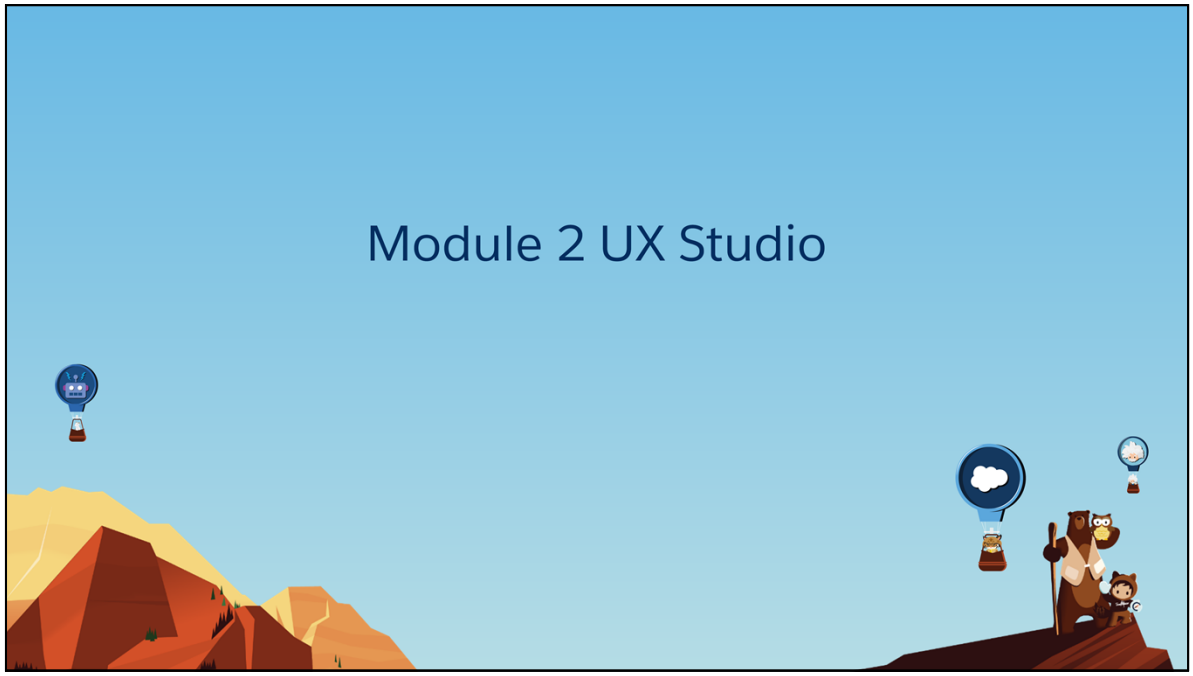
# Knowledge Check 1.2



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 |

# Module 2 UX Studio



## Module 2: UX Studio

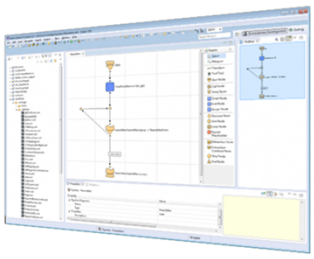


**Objectives:**

- Use UX Studio to create a new workspace.
- Set up a server connection
- Navigate the user interface.

**Lessons:**

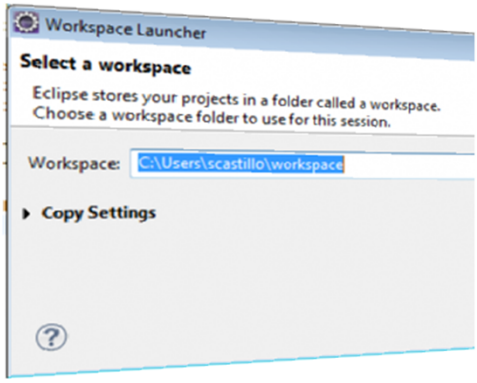
2.1 Creating a Workspace
2.2 Creating a Server Connection
2.3 Commerce Cloud B2C Commerce Views



UX Studio is an Integrated Development Environment (IDE) used for programming in the Commerce Cloud platform. It is a plugin built on the Eclipse open-source development platform (www.eclipse.org), which many Java developers use to build Java applications. It is not necessary to know Java to use UX Studio.

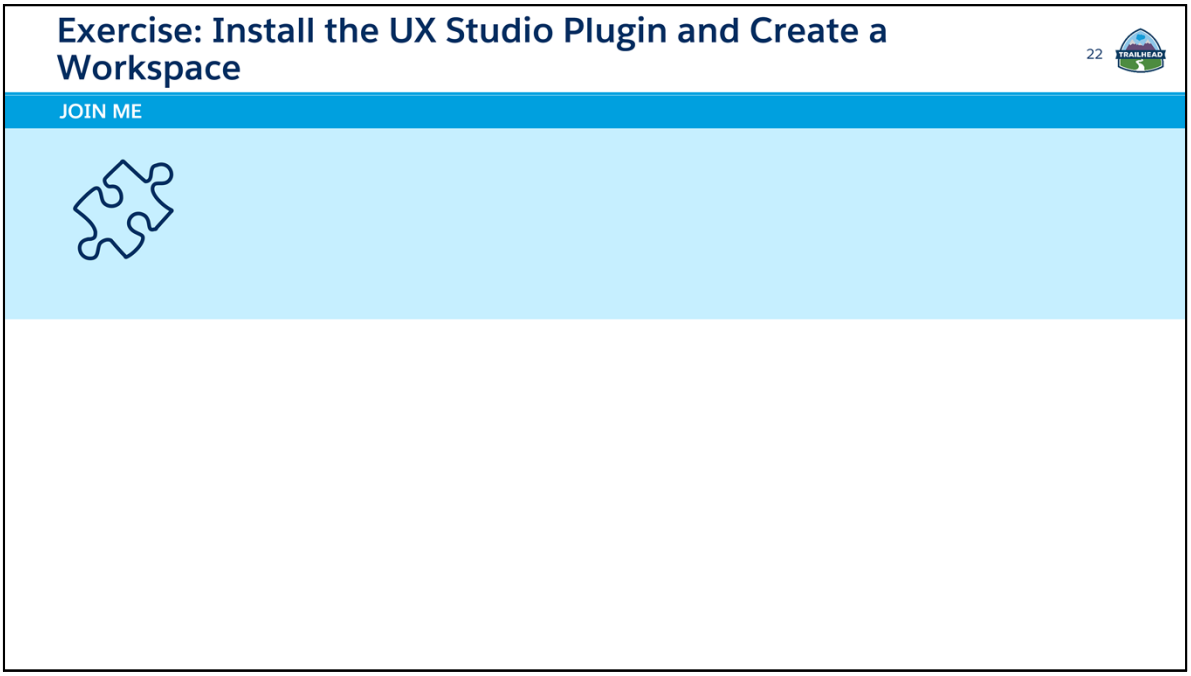
## Lesson 2.1: Creating a Workspace






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

Each workspace should have only one B2C Commerce server connection. For example, if you are working on numerous client projects, you should create a separate workspace for each client. Each client workspace then has only one specific server connection.




Create a new workspace (when using UX Studio for the first time):
Note: This assumes that you have installed the UX Studio plugin into Eclipse.

1. The first time you use the application, you will be prompted to create a new workspace name. You can use a workspace that references your client.
2. Eclipse displays a Welcome message in the main working area.
3. Select **Help > Install New Software**.
4. Click **Add**. The **Add Repository** dialog displays.
5. In the **Name** field, enter UX Studio.
6. In the **Location** field, enter one of the following URLs based on your version of Eclipse:
   a. Mars: http://developer.salesforce.com/media/commercecloud/uxstudio/4.5
   b. Neon: http://developer.salesforce.com/media/commercecloud/uxstudio/4.6
7. Provide a name for the URL.
8. Select **Salesforce Commerce Cloud** from the list. Click **Next**. At this point, Eclipse compares UX Studio's requirements with what is available to ensure compatibility. The **Install Details** dialog displays.
9. Click **Next**. The **Review Licenses** dialog displays.
10. Select the license agreement radio button. Click **Finish**. The **Installing Software** dialog displays, indicating the installation's progress.
11. In the **Security Warning** dialog, click **OK**.
12. Select **Yes** when prompted to restart Eclipse.

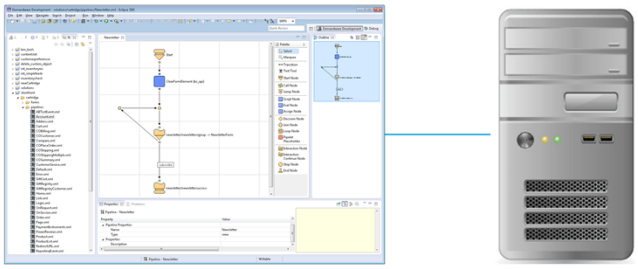
UX Studio is now installed. You can now use the **Digital Development** perspective in the upper right corner.

If you already have a workspace and need to create a new one:

1. From the main menu in UX Studio, select **File > Switch Workspace > Other**.
2. The **Workspace Launcher** dialog displays.
3. In the Workspace field, enter a new workspace name and location. Click **OK.**
4. UX Studio closes and reopens.

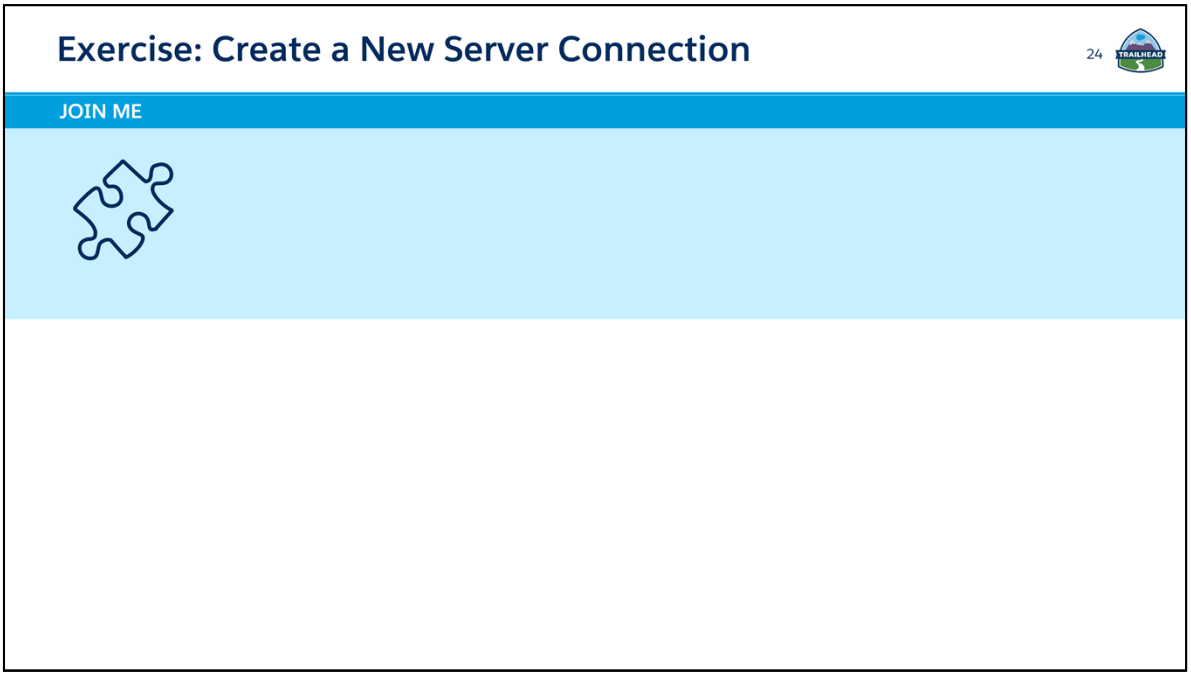
## Lesson 2.2: Creating a Server Connection





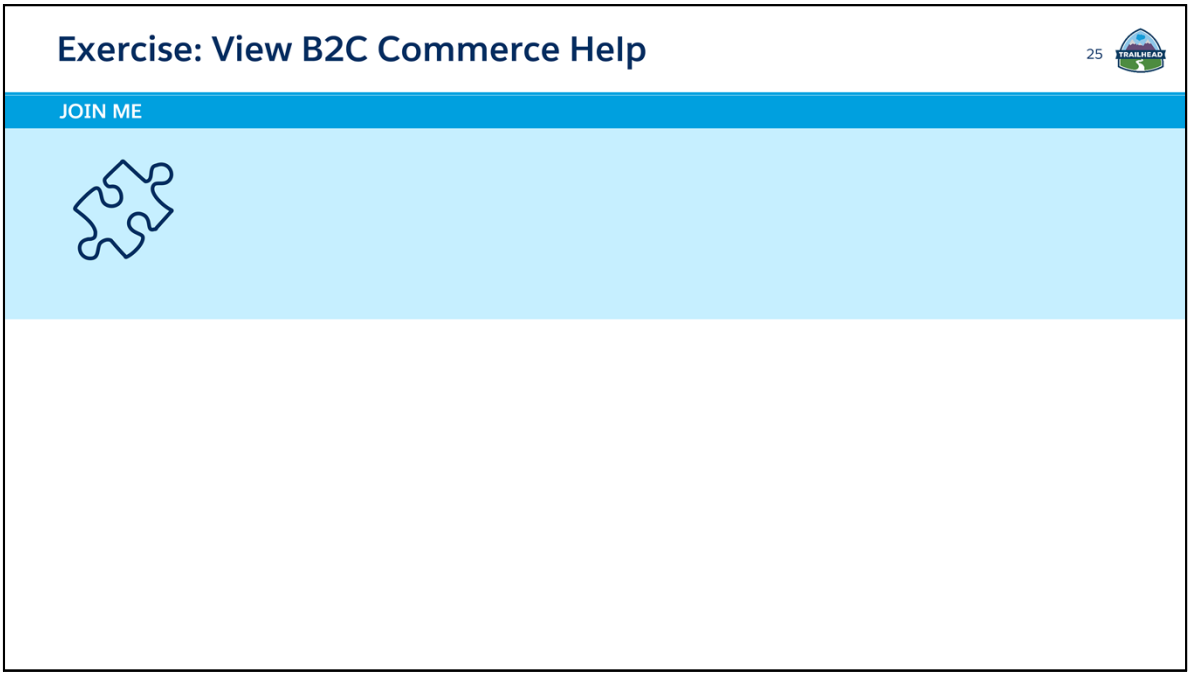
You must create a server connection in UX Studio to be able to upload your code to the B2C Commerce server instance. Note: It is a one-way push connection; you cannot pull code onto a local computer from the B2C Commerce server.

In order to upload your code to a Commerce Cloud server, you must create a server connection in UX Studio. This enables you to push your code to the server instance. However, you will not be able to pull the code onto your local computer from the Commerce Cloud server–as it is a one-way push connection



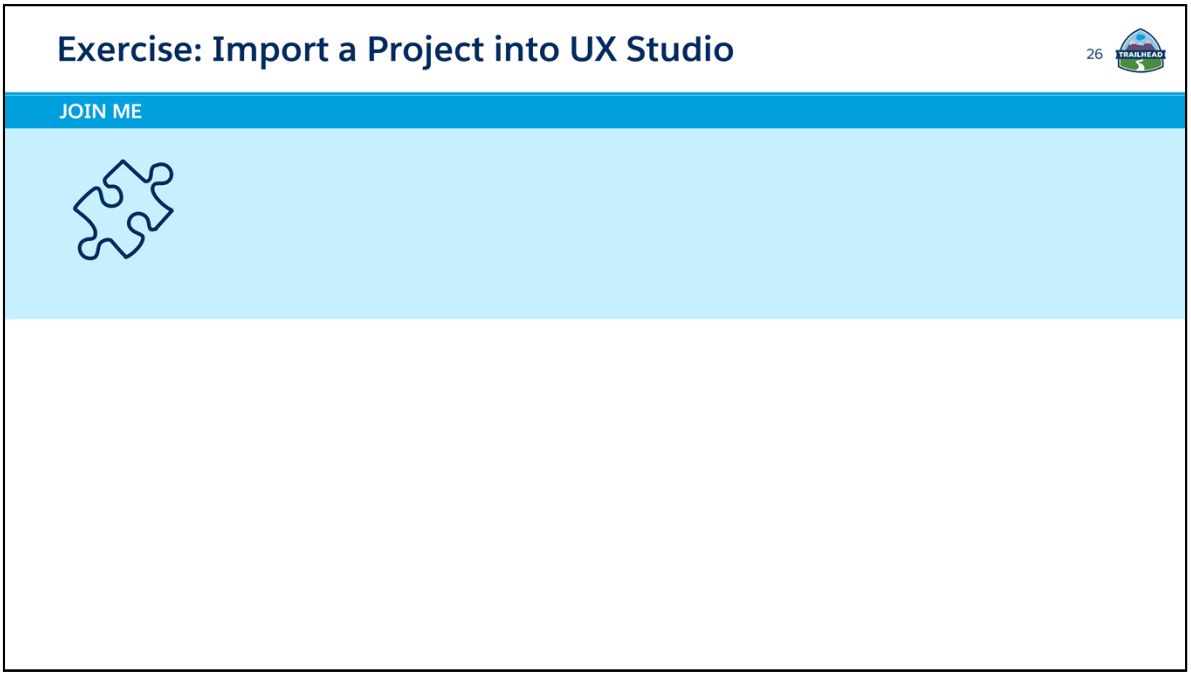

1. From UX Studio, select **File > New > Digital Server Connection**. The **New Digital Server Connection** dialog displays.
2. In the **Project name** and **Host name** fields, enter the host name provided by your instructor or client:
   - student##-training-na-dw.demandware.net (where # is unique for each student).
   - partner##.cloud01.pod3.demandware.net (where partner## varies by partner company)
3. Enter your Business Manager password.
4. Click **Next**.
5. A security warning regarding an invalid certificate for your sandbox displays. Click **Yes** to continue.
6. In the Target version directory field, select **version1** as the target version for your uploaded files.
7. Click **Finish**.

Your project is now connected to your sandbox. You will use that connection to upload any cartridge projects to that sandbox.




To open the B2C Commerce API Javadoc:

1. From UX Studio, click **Help > Help Contents**.
2. Expand the **B2C Commerce API** link.
3. Select any of the available help items. Note: The first two items offer Javadoc-style help.




1. In UX Studio, select **File > Import.** An Import dialog displays.
2. Expand the **General** menu and select **Existing Projects into Workspace**.
3. Click **Next**.
4. In the **Select Root Directory** field, click **Browse**.
5. Find the folder on your hard drive where cartridges are stored. Your instructor will provide a zip file with all solution cartridges for you to install locally.
6. Click **OK**.
7. Any cartridges in the folder structure (including subfolders) will display in the **Projects** box. Click **Select All.**
8. Click **Finish**.
9. 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.
   **Note:** If you want to upload the imported cartridges to the active server, click **Yes**. Otherwise the cartridges will reside in your workspace, but will *not* upload automatically when you make changes.
10. Click **OK** to upload the cartridges and finish the import process.
    **Note:** You might receive a dialog that asks if you want to delete projects on the server not matching the ones in your workspace. If you are the only one working on that instance (e.g. it's your personal sandbox), you can make that decision. However, if you are working on a collaborative instance consult first with your colleagues.

**Note**: If you import cartridges before you have an active server connection or have failed to link a cartridge to the server, perform the following steps to ensure that cartridges upload correctly:

- Right-click the server connection and select **Properties.**
- In the **Properties** dialog, select **Project References.** Then select every cartridge that you want uploaded to the server connection. Click **OK.**

## Lesson 2.3: B2C Commerce Views

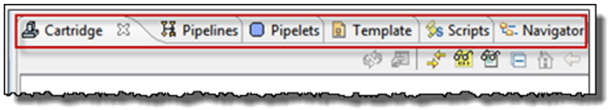


The UX Studio plugin provides access to specific B2C Commerce programming files.

These files are sorted and given their own custom views in the application.

The primary files include: cartridges, pipelines, pipelets, templates, and scripts.

UX Studio contains tabs for each.

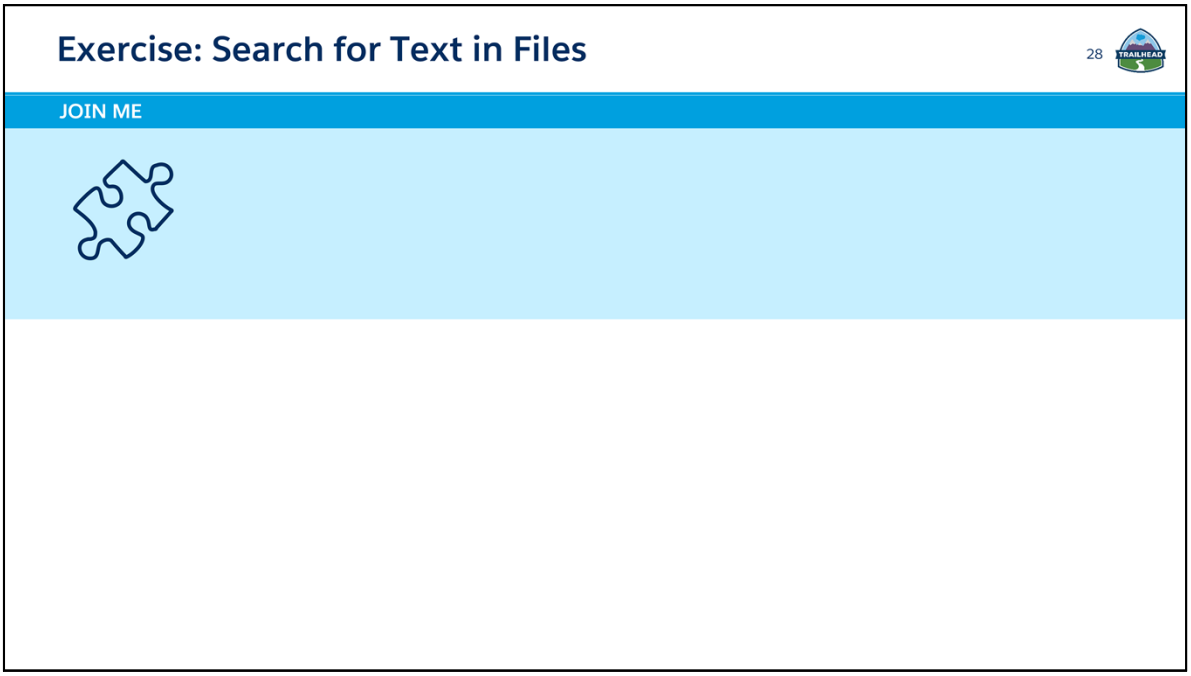



The UX Studio plugin provides access to specific B2C Commerce programming files. These files are sorted and given their own custom views in the application. The primary files include: cartridges, pipelines, pipelets, templates, and scripts.

UX Studio contains tabs for each.

**Tips:**

- You can 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 the **Show/Hide Resource Part** icon.
- The **Navigation** tab enables you to view all files in your workspace in a tree view structure. It also facilitates common tasks, such as: copy/paste, file comparisons, etc.




## Searching for Text in Files

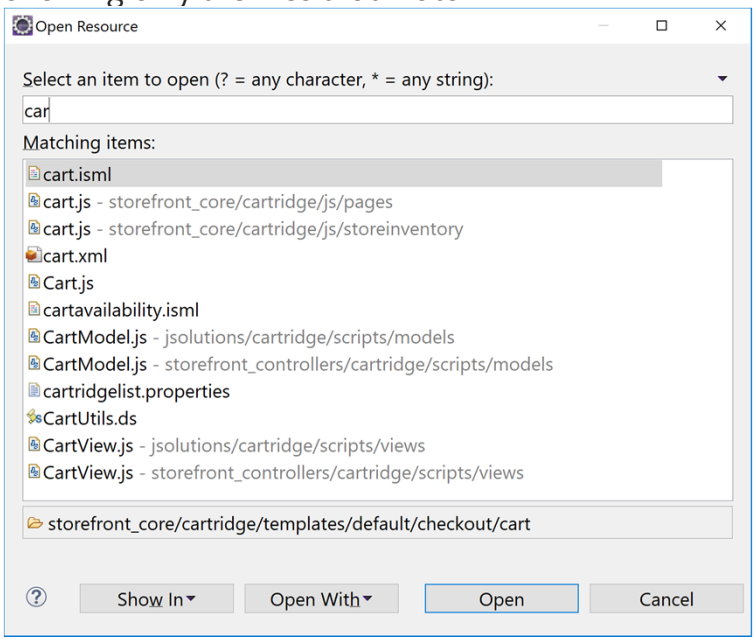
There are often numerous files used to display a single web page in a browser. You can use the UX Studio search tool to quickly find specific code within that set of files.

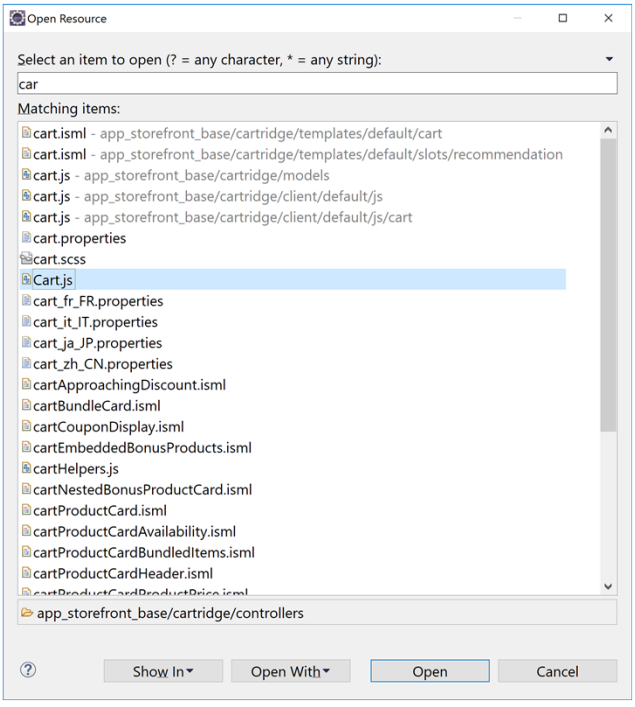
To search for text in files:

1. In the main menu bar, select **Search > File**. Select the **File Search** tab. The search window displays.
2. In the **Containing text** field, enter your text search criteria.
3. In the **File name patterns** field, enter any patterns to use as a filter.
4. Click **Search**. Your results display in the **Search** box.
5. Double-click a file to view it. The file will open in the workspace area with the search term highlighted.

## Use Open Resource to Search for Files

If you know the name of a file and want to find it, you can use Open Resource. In the main menu, click **Navigate > Open Resource**, or use the shortcut keys Ctrl-Shift-R. This opens a dialog in which you can enter the file name. The example below shows the characters “car” in the Open Resource dialog. The search result is all files that begin with those letters. As you type more letters, the list continually updates showing only the files that match.



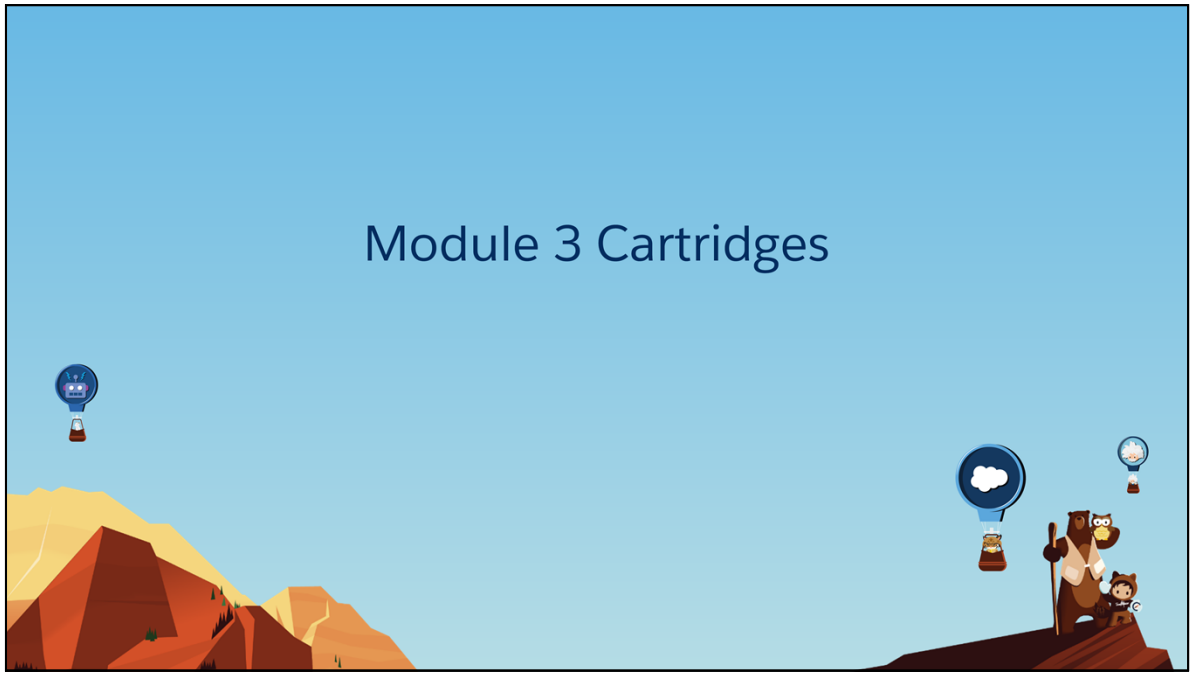



## Knowledge Check 2



1. To upload your code to a B2C Commerce server, you should:
   A. Copy files to the root folder on the web server.
   B. Connect to a production server in a PIG.
   C. Create a server connection in Studio.
   D. Contact support to open a server connection for you.
2. To find text in any workspace file:
   A. Select File > Find Text.
   B. Select Search > File.
   C. Select Edit > Find.
   D. Use the Windows search option.

# Module 3 Cartridges



## Module 3: Cartridges

31


**Objectives:**

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

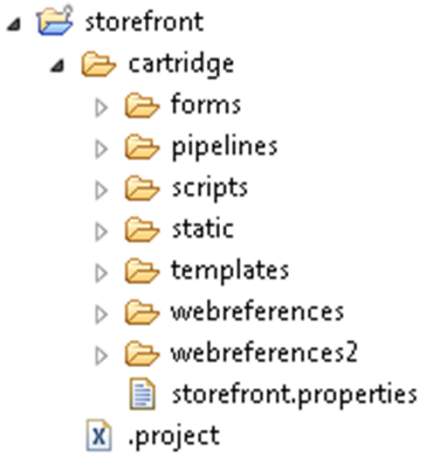
**Lessons:**

3.1 Cartridge Path
3.2 Cartridge Types
3.3 Creating a SiteGenesis Storefront Cartridge

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

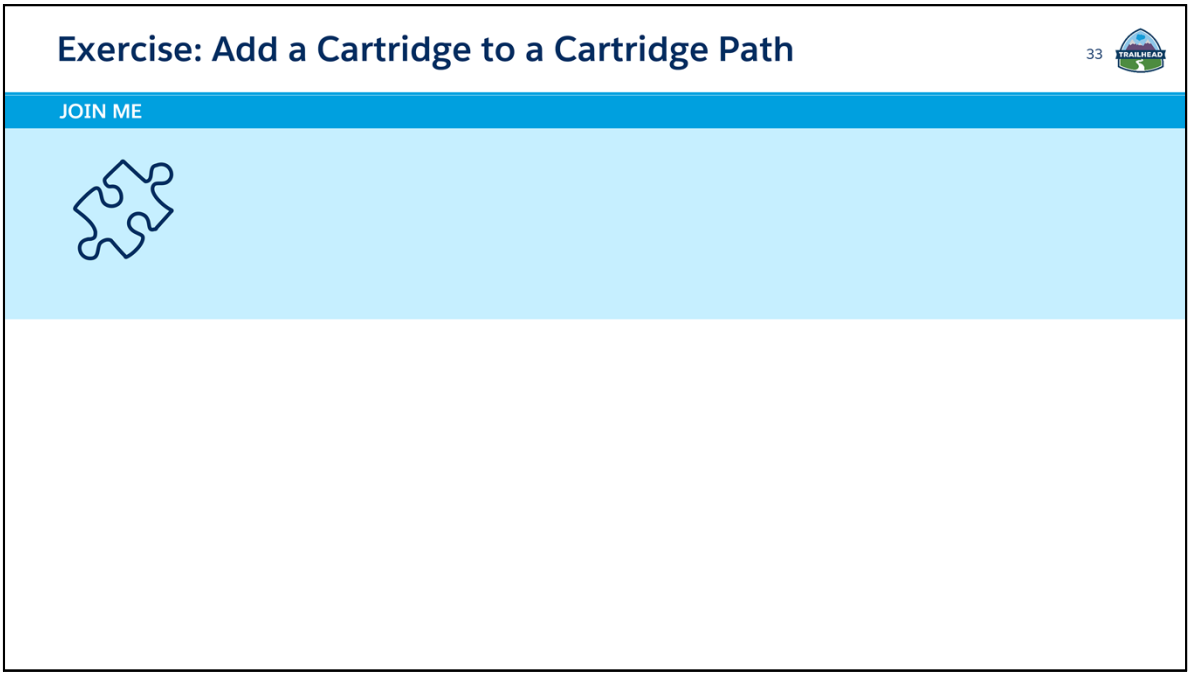
A cartridge is fully contained in one directory. Every cartridge has specific sub-directories where certain file-types **must** be stored. For instance, all Commerce Cloud script files must be stored in a **scripts** folder.

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




For a site to use a cartridge, you must add the cartridge to the cartridge path in Business Manager. Select **Sites > Manage Sites > Site Genesis – Settings**.

When a call is made to a file, the Commerce Cloud 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 the `product.isml` file and that file is located in two cartridges that are both in the cartridge path, the Commerce Cloud server will use the first one it finds.




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

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

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

5. Click **Apply**.

# Lesson 3.2: Cartridge Types



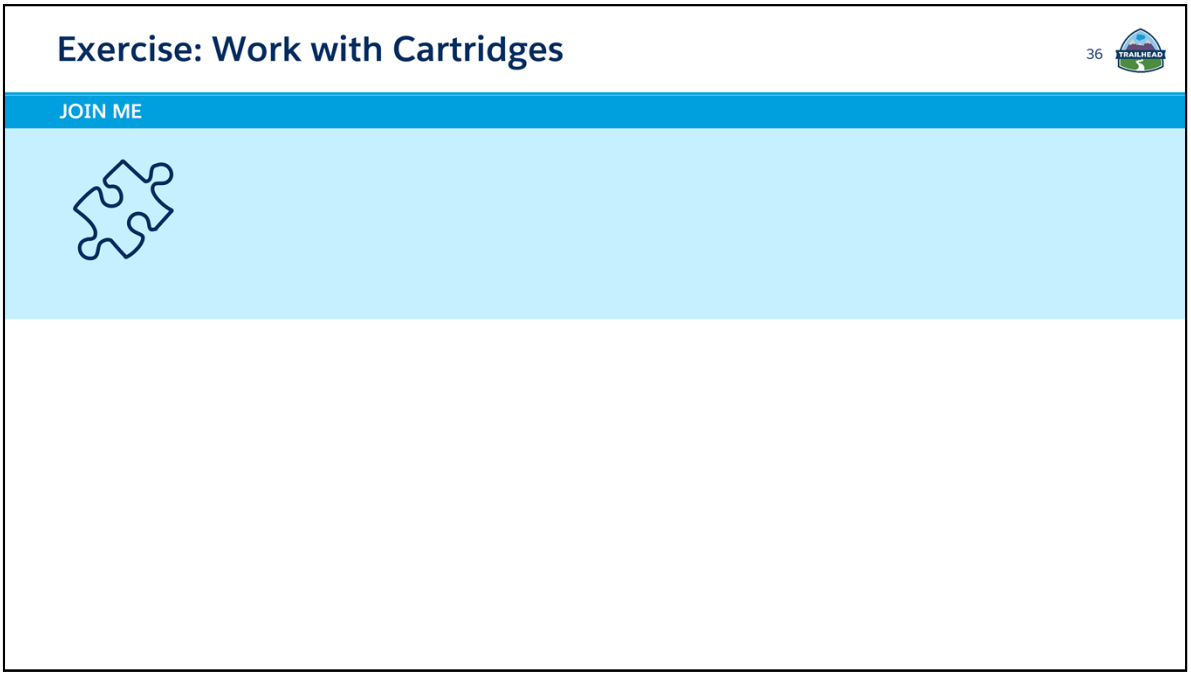
Your business needs determine the type. However, every new site will have at least one cartridge.

| Cartridge Type | Description |
|---|---|
| SiteGenesis 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. |
| Cartridge | Use to build site-specific, re-usable functionality when there are multiple sites in a production instance.<br>You may want to add a new cartridge when functionality is:<br>▪ Generic: reusable code used in multiple sites.<br>▪ An integration to an external system.<br>▪ Specific to a localized site: CSS, images and resource files for a language-specific site. |
| Business Manager Extension Cartridge | See your Commerce Cloud B2C Commerce documentation for more information. |

## Best Practices



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




**View the WebDAV Cartridge Directory in Business Manager**

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

**Create a New Version on the Server**

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

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

**Create an Empty Cartridge**

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

1. In UX Studio, log into your workspace.
2. From the main menu, select **File > New > Cartridge**. The **New Cartridge** dialog displays.
3. Enter the cartridge properties exactly as listed:
   - Name: training
   - Location: C:\projects\DigitalServer\sources\cartridges
   - Link to Server: Checked
4. Click **Finish**.
5. Review the **training** cartridge. Notice that the directory structure was created but there are no files.

The reference cartridge: code
CSS and advanced UI elements: look and feel

Changes made to the new storefront cartridge are uploaded to the server.

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

When you create a new storefront cartridge in UX Studio, a copy of the **SiteGenesis** cartridge downloads to your workspace and is renamed with the name you specify. The reference cartridge has all of the code needed for a SiteGenesis storefront to work in Commerce Cloud B2C Commerce. It contains:
- The business layer, all of the server-side components (such as B2C Commerce scripts).
- The simple presentation layer, including ISML templates, common CSS files, forms, and resource files.

The specific CSS and advanced UI elements required create the look and feel of the SiteGenesis storefront.

Changes made to the new storefront cartridge are uploaded to the server. To view them immediately:
1. Set the cartridge to be uploaded to your workspace server.
2. Put the new cartridge in the cartridge path.
3. Disable site caching for the site.

## Exercise: Create a New Storefront Cartridge



JOIN ME

1. In UX Studio, log into your workspace.
2. From the main menu, select **File > New > SiteGenesis Storefront Cartridge**. The **New Storefront Cartridge** dialog displays.
3. Complete the fields in the New Standard Storefront cartridge dialog.
   - Name: storefront
   - Location: C:\projects\DigitalServer\sources\cartridges
   - Attach to Commerce Cloud B2C Commerce Servers (studentxxx.training.dw.demandware.net): Checked
4. Click **Finish**.

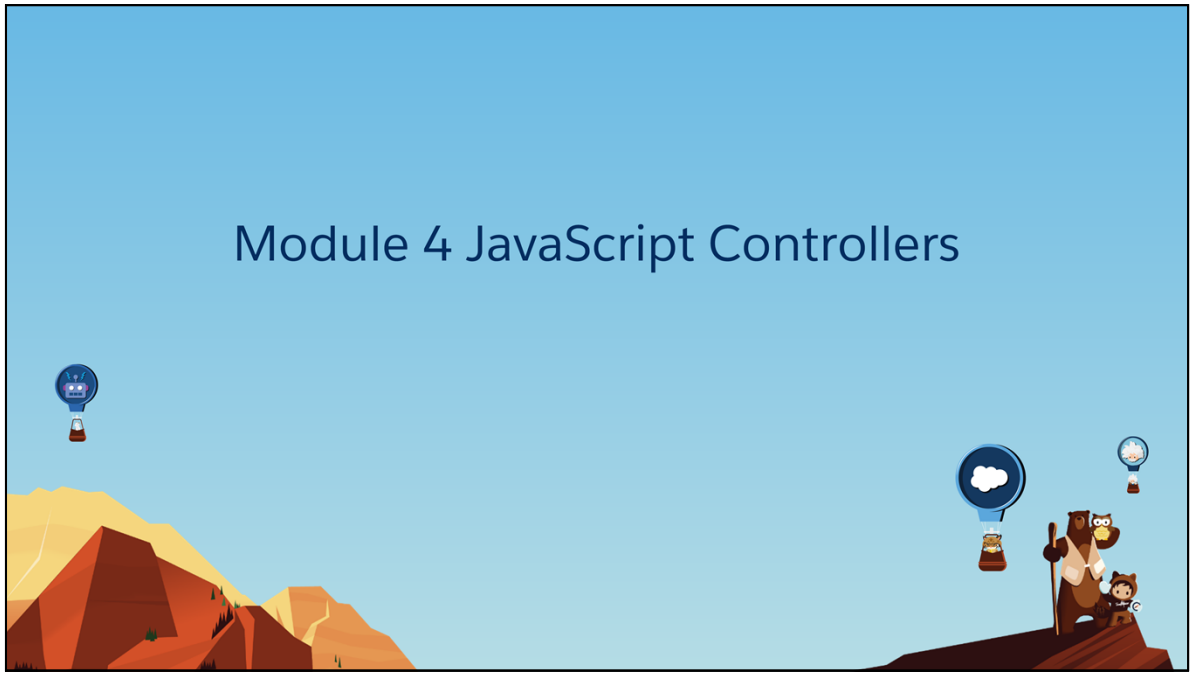
# Knowledge Check 3



**True or false:**

1. A SiteGenesis Storefront cartridge are an empty cartridges with empty sub-folders.
2. You should create a new cartridge when you need to build generic functionality that can be reused in many sites.
3. You can view a list of all files in a cartridge located on a B2C Commerce server from Business Manager.

# Module 4 JavaScript Controllers



## Module 4: JavaScript Controllers

41

**Objectives:**

- Describe JavaScript controller usage.
- Create, execute, and troubleshoot JavaScript controllers.

**Lessons:**

4.1 Introduction to JavaScript Controllers

4.2 Displaying a JavaScript Controller

NOTE: Since Pipelines are the legacy controller in DW, the “new” module would logically be controllers. There are two types of controllers: Pipeline and JavaScript.

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

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

### Cartridge Folder Structure

```
<cartridge>
     +-- modules
     +-- package.json
     +-- cartridge
     +-- controllers
     +-- forms
     +-- pipelines
     +-- scripts
     +-- static
```

JavaScript controllers enable you to use a common open technology to build the business logic of the site. Note: Salesforce recommends that all new sites use JavaScript controllers rather than pipelines. However, if you need to maintain an existing site that use pipelines, see Appendix A.

## Controller Code Example



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

function start() {

        ISML.renderTemplate(
                'helloworld1.isml',
                      {
                        myParameteronISML:"Hello from Controllers"
                      }
                      );
};
exports.Start = guard.ensure(['get'], start);
```

The `require` keyword imports a class from the API package to be used in the code.

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

## Controller Code Example



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

Never write to the response object in production.

The ISML object gives the control to ISML, which can display the results. You can use response object directly to display the results as shown below but it is not recommended.

## Controller Code Example



```
ISML.renderTemplate(
                'demo.isml',
                {
                      myParameter:”Message from Controllers”
                }
                 );
```

```
${pdict.myParameter}
```

Ideally the output should be rendered through ISML which is the view (Just like HTML or JSP). Also to pass the results to the ISML. Here is an example code to output the results to an isml named `demo.isml` and passing a string message to the parameter `myParameter` to the ISML.

The parameter `myParameter` gets loaded on a hashmap named `pdict` and can can be retrieved from the ISML (`demo.isml`) using the syntax shown.

More details about the `pdict` are coming later in this course.

## Import Packages



Use the `require` method to import B2C Commerce script packages, JavaScript, or B2C Commerce script modules.

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

You can use it anywhere.

CommonJS modules are JavaScript files that are loaded using the require(String) function. This function returns a module object, which wraps the script code from the file. Within a module implementation, the module object can be accessed via the module variable.

Use the `require` method to import B2C Commerce script packages, JavaScript, or B2C Commerce script modules. You can use it anywhere in your script so that you can only load the functionality as needed to improve performance.

Note: Earlier versions of SiteGenesis used `importPackage` to import Commerce B2C packages into scripts. `Require` is now the recommended approach as it has it has greater flexibility and improves performance. However, you may still see `importPackage` if you are maintaining a site from earlier version.

## Export Packages



```
require('storefront_controllers/cartridge/scr
ipts/guard');
exports.MyPublicName =
guard.ensure(['filter1', 'filter2'],
myFunctionName);
```

Supported Filters:
- https
- http
- get
- post
- loggedIn

guard.js

Every CommonJS module object has an exports property which can be used by the module implementation to expose its public functions or properties. Only functions and properties that are explicitly exported are accessible from other modules, all others are private and not visible.

For controllers, SiteGenesis uses the concept of guards to wrap controller functions when they are exported. The functions in the guard module act as a request filter and let you specify multiple levels of access to controller functionality, such as:
- require HTTPS
- require or forbid a certain HTTP method, like GET, POST,...
- require that the current customer is logged in
- require that the function may only be executed as remote include, but *not* as top-level request

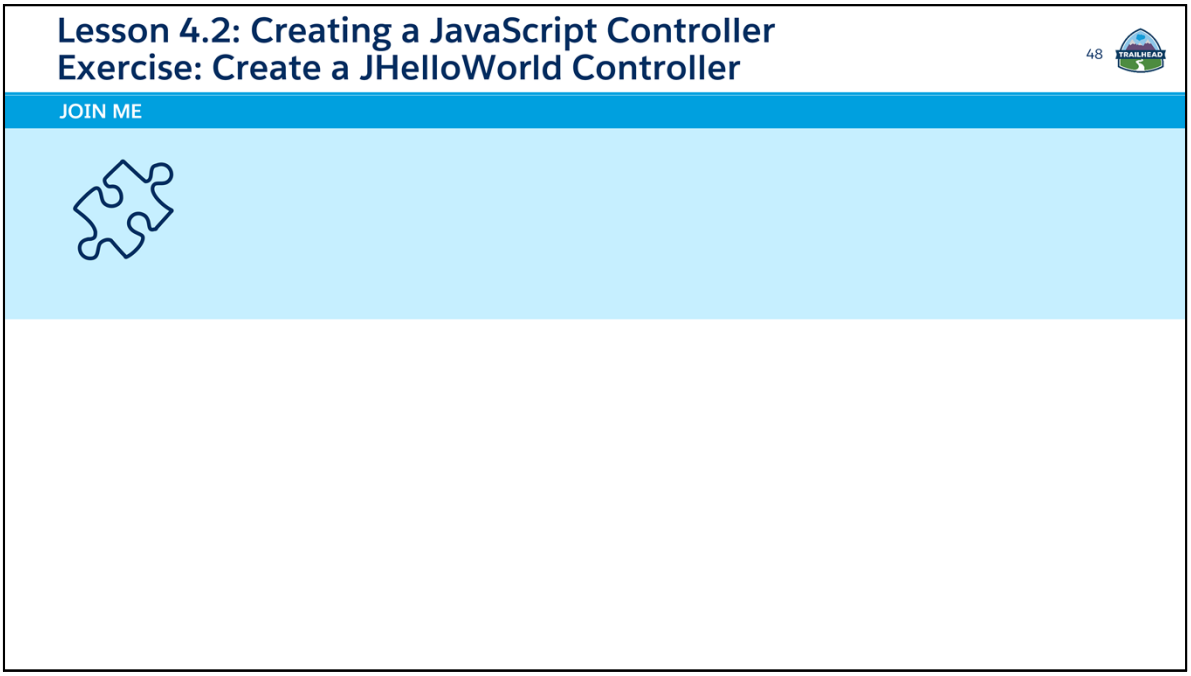
A guard is a wrapper function that encapsulates a delegate function and only invokes it if its guard condition is satisfied. The guard conditions represent single preconditions that can also be combined with each other. The conditions use the Digital API to determine the properties of the current request and logic to determine whether to continue request processing or throw an error.
Example: require GET

```
    exports.StartCheckout = guard.ensure(['get'], startCheckout);
```

Example: require HTTPS and that the customer is logged in.

```
    exports.EditProfile = guard.ensure(['get', 'https', 'loggedIn'],
editProfile);
```

**Note:** You can also create custom guards functions, such as "stagingOnly" or "loggedInAsEmployee".




1. In Business Manager, select **Administration > Sites > Manage Sites > SiteGenesis > Settings**.
2. If it is not already there, add the `storefront_controllers` cartridge to the cartridge path. It should now be similar to: `training:storefront_controllers:storefront_core`
3. Upload your cartridge to the Sandbox:
4. Select Eclipse. Right-click and select **DigitalServer > Properties > Project References**. Check the `storefront_controllers` cartridge.
5. Create a new controller named `JHelloWorld.js` in the training cartridge (right-click controllers. Select **New** file). Note: This is the only cartridge that you will use in this course.
6. Copy and paste the following structure to create a start function. Use ISML and guard.

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

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

function start() {

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

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

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

8. In the templates/default folder, create an ISML file named `helloworld1.isml` with the following code in it. Note: pdict will be discussed later.

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

9. Execute the controller.
10. Navigate to the storefront.
11. At the end of the URL add `/default/JHelloWorld-Start`.

12. Press <enter> to execute the controller.

## Pipeline Setting Troubleshooting



Use this checklist to review your settings from the Navigator view.

If your pipeline does not work, use this checklist to review your settings from the Navigator view.

- Right-click **DigitalServer**. A pop-up menu displays. Hover your mouse pointer over the **Digital Server** menu item**.** Ensure that **Active Server** and **Auto-Upload** are checked.
- Select **DigitalServer > Properties > Project References**. Ensure that your cartridges are checked for being uploaded to the server.
- Select **DigitalServer > Digital Server> Change Upload Staging Directory**. Ensure that the **Target** version directory and **Active** version directory match.
- Select **DigitalServer > Digital Server> Update Server Configuration**. Ensure that the configuration strings are correct.
- In Business Manager, select **Administration > Sites > Manage Sites > SiteGenesis.** Select the **Settings** tab. Check your cartridge path. It should have the exact name as the cartridges in Eclipse. Names are case sensitive and should have no spaces in the path. There should be no semicolons in place of colons.
- Select **Administration > Sites > Manage Sites > SiteGenesis.** Select the **Cache Tab.** Check if **Time to** live (TTL) is 0 and **Enable Page Caching** is disabled**.**
- If you have not done so, index your site. Select **Site > SiteGenesis > Search > Search Indexes.** Check all checkboxes and click **Reindex**.
- Save your project before executing the pipeline and type the URL correctly.

In Eclipse, in the **Project** menu, ensure that your project is configured to **build automatically**.

## Pipeline Dictionary of Global Variables



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

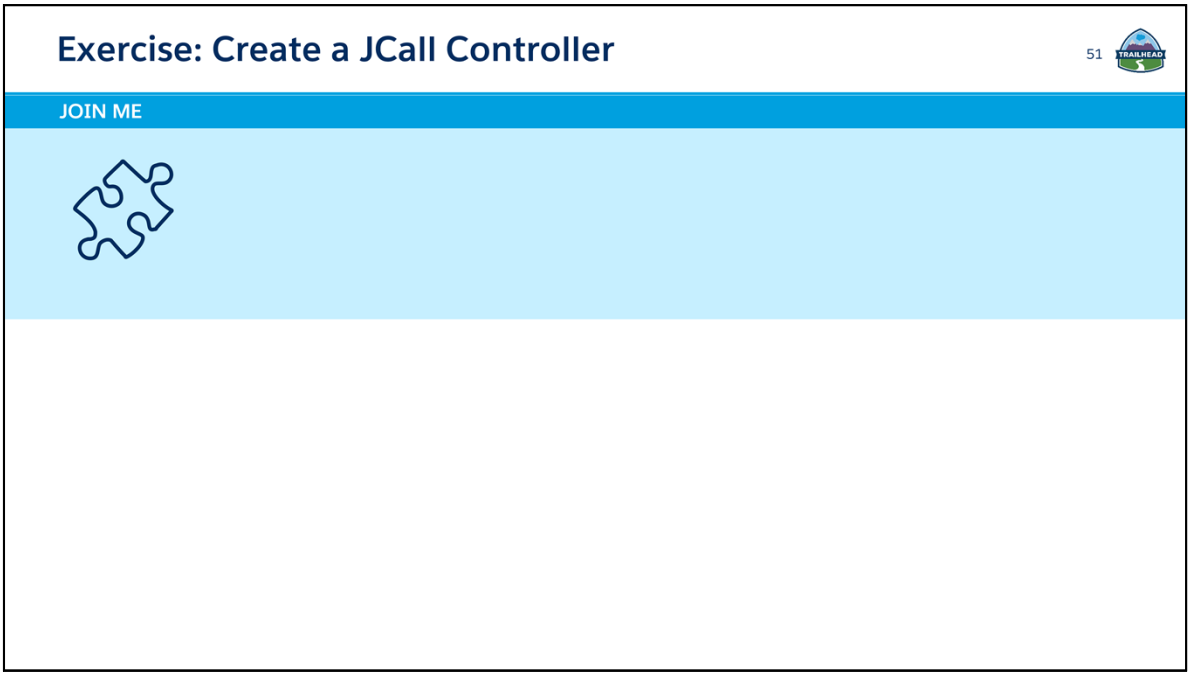
- The default pdict keys will still be available to the templates for backward compatibility.
- Any additional values must be passed to the template by the controller.
- The syntax for passing values will be covered next.

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.

pdict is a hashmap on which key, object pairs can be loaded. However, there are some built in pdict keys (variables) that provide access to the most commonly used objects, such as session and request.

JavaScript controllers can use alternatives to pdict keys. Here are some of them. The strikethroughs indicate implicit packages or classes in JavaScript controllers.

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




The goal of this activity is to display query parameters using JavaScript controllers.

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

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

if (myParam.stringValue != null)
        {
/* Use the quickcard section "Giving control to ISML" to give control to
call/jnotEmpty.isml and loading myParam on a variable
paramOnPdict*/
        }
else{
         ISML.renderTemplate(
                    'call/jempty.isml',
                   {
                          paramOnPdict:'param not found'
                  }
         );
};
```

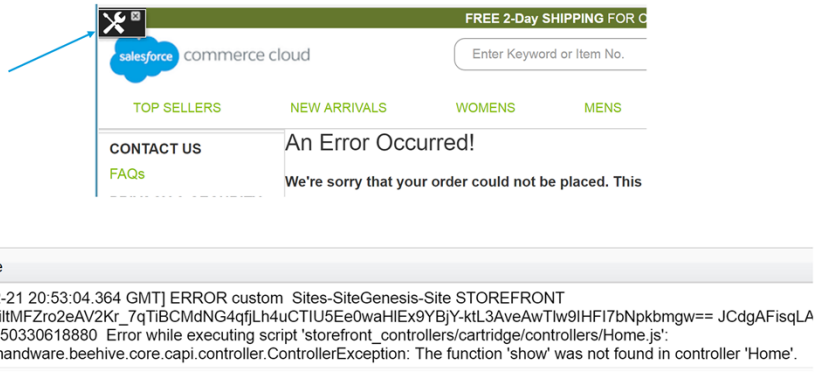
5. Follow the instructions in the template comments to complete the code.
6. Create two templates `jnotEmpty.isml` and `jempty.isml` (under `templates/default/call`) that display different messages.

- The successful path should have an ISML code in `jnotEmpty.isml` as shown:
  `Got the parameter ${pdict.paramOnPdict.stringValue}`
- The path which does not have parameter () should have ISML code in `jempty.isml` as shown:
  `Could not find the parameter!`

7. Execute the code by Navigating to storefront and adding `/default/JCall?param=1234` to the URL at the end.
8. Execute the code without the query parameter.

# Troubleshooting with the Request Log Tool



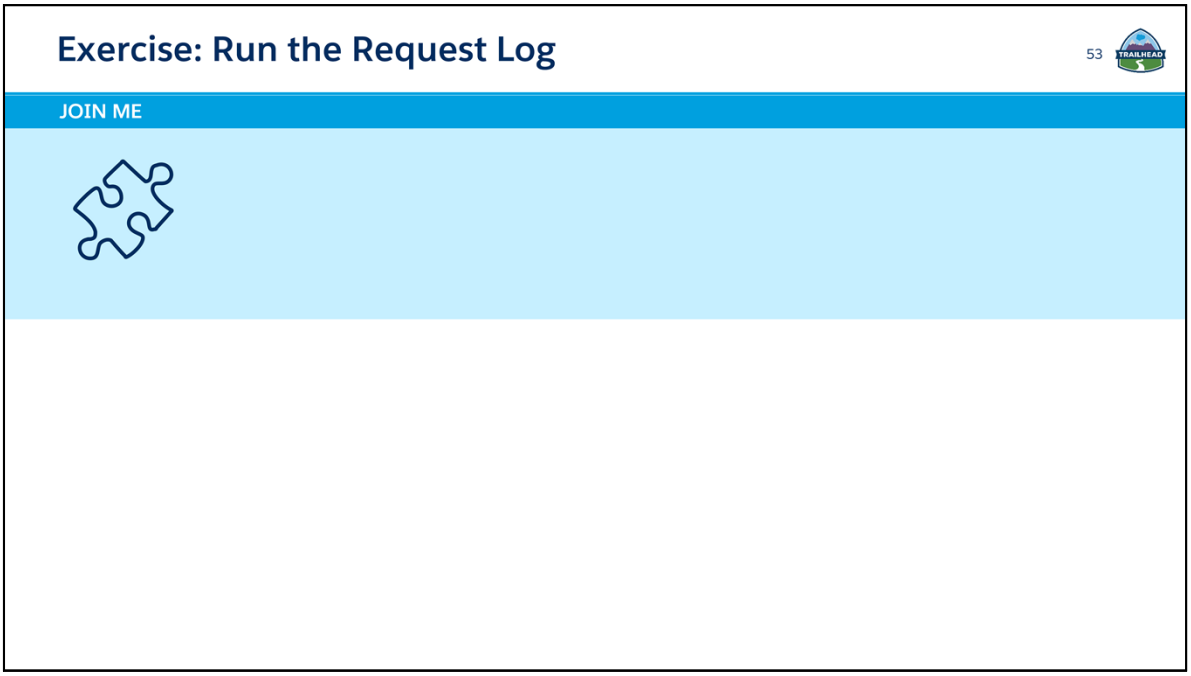



The Request Log tool enables you to troubleshoot error messages on your storefront.

It displays the log for the last request and any request prior to that request during the current session.

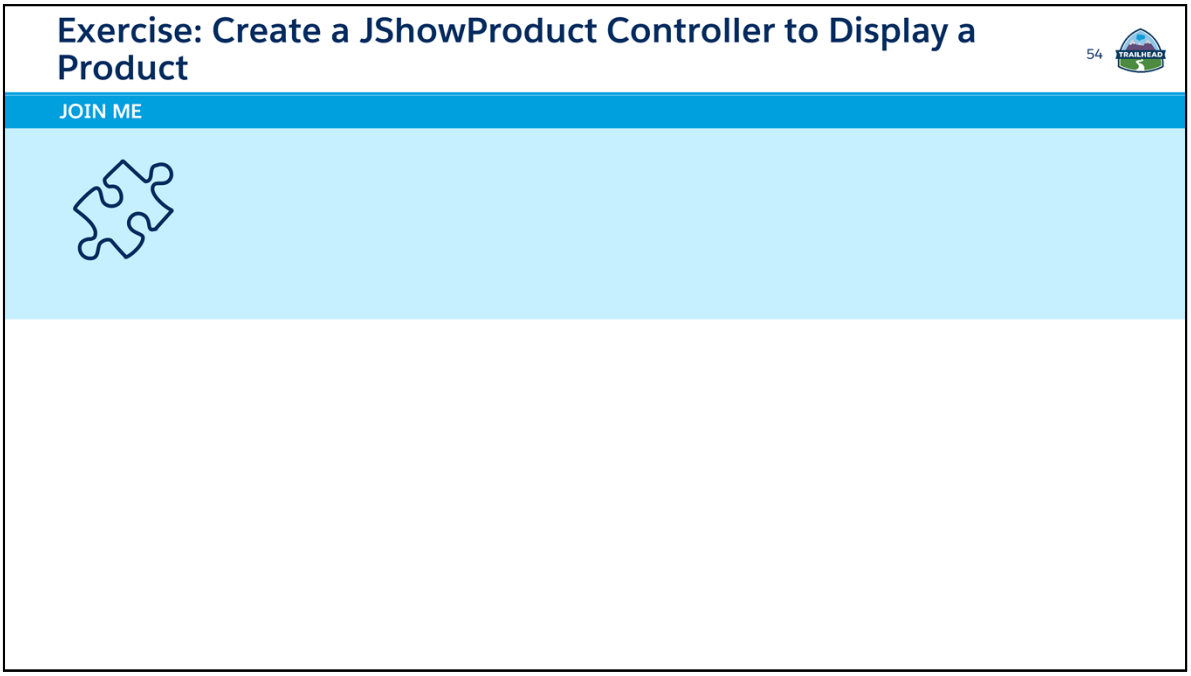
You can view both debug messages and error messages.

The tool is part of the Storefront Toolkit, which is available in all instances (except Production).




1. From Business Manager, click the storefront link to open your sandbox storefront.
2. Click the Storefront Toolkit drop-down located in the upper-left hand corner of the site.
3. Check the **Request Log**. The Request Log window displays.

Note: If a login screen displays instead of the request log, enter your Business Manager login credentials; close the window; and repeat steps 2-3.




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

```
'use strict';
/** @module controllers/JShowProduct */

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

function start() {

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

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

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

5. Get the product from ProductMgr as shown.

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

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

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

7. If not already created, create `templates/default/productnotfound.isml` with the following code in it: `${pdict.message}`
8. If not already created, create `templates/default/productfound.isml` with the following code in it: `${pdict.myProduct.name} has been found`
9. Run the JavaScript Controller (add `/default/JShowProduct?pid=P0048` at the end of storefront

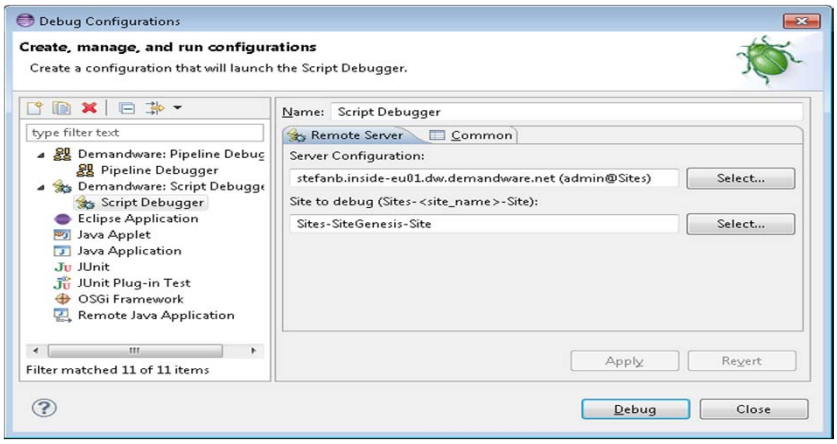
URL).

## Exercise: Create a Script/Controller Debug Configuration



JOIN ME

1. In UX Studio, find the menu to create debug configurations.
2. Double-click **UX Studio: 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 `JShowProduct` controller you previously created.
6. Put a breakpoint in the first executable line inside the controller's `start()` function: double-click the gray border to the left of the highlighted line. The breakpoint display has a blue dot.

```
 9 function start() {
10     var parameterMap = request.httpParameterMap;
11     var parameterId =parameterMap.pid.stringValue
12     var product = ProductMgr.getProduct(parameterId);
13     if (product==null) {
```

7. Refresh the controller's invocation on the browser to hit the breakpoint (**F5**).
8. The debugger stops at the breakpoint.
9. Debug the script.
10. Check the **Variables** window to determine the `args` that are coming into the `execute()` function.
11. Use **F5** to execute the line.
12. `Study the output variable which has the product: it should not be null.`
13. Execute through the end of the controller (**F8**). The product name should display in the browser.
14. Fix any errors that you may have found, or just continue.
15. Debug the controller again, but this time use an invalid product ID in the URL.
16. Change the `product` URL parameter on the browser to a non-existing product.
17. After the breakpoint, verify the output variable holding the product: it should be `null` in this case.

18. Execute through the end of the controller.

# Knowledge Check 4



**True or false:**

Are these statements about Commerce Cloud JavaScript controller true?

1. If the execution of a controller is not providing accurate results, you should use the request log tool.
2. You should preferably use `response.getWriter(..)` to print results on the browser.

# Module 5 ISML

## Module 5: ISML



**Objectives:**

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

**Lessons:**

5.1 ISML Tags and Expressions
5.2 Creating and Accessing Variables
5.3 Reusing Code in Templates
5.4 Conditional Statements and Loops

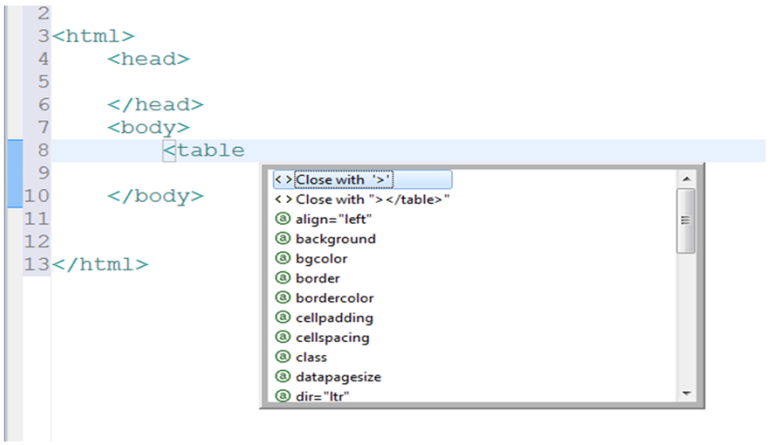
Internet Store Markup Language (ISML) templates are files with a .isml extension. They define how data, tags, and page markup are transformed into HTML that is sent to the browser, using Cascading Style Sheets (CSS) for page layout and styling.

Commerce Cloud B2C Commerce uses templates to generate dynamic HTML-based web pages for responses sent back to the client. Templates are created using ISML tags and expressions.

When describing a B2C Commerce application using the Model-View-Controller (MVC) pattern, the B2C Commerce Script API represents the model, templates represent the view, and the JavaScript controllers are the controller.

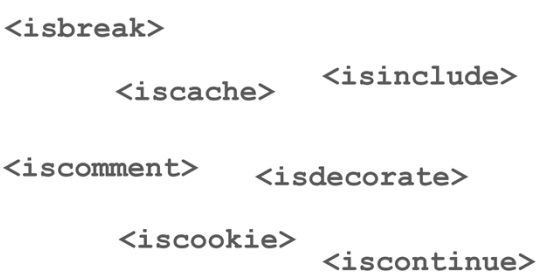
## Create an ISML Template






1. In UX Studio, select a cartridge in Navigator View. Select **File > New > ISML Template.** The **Create Template** dialog displays.
2. In the parent folder field, enter the name of the folder where you want to store your template. If the folder does not exist, it will be created.
3. In the Template name box, enter a name for your template. You do not need to enter the .isml extension.
4. Click **Finish**.

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




ISML tags are Commerce Cloud proprietary extensions to HTML that developers use inside ISML templates. ISML tags and expressions can only be written in in 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.

# ISML Tag Categories (1)



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

| Group | Tags | Purpose |
|---|---|---|
| HTTP-related | <iscookie> | Sets cookies in the browser |
| | <iscontent> | Sets the MIME type |
| | <isredirect> | Redirects browsers to specific URLs |
| | <isstatus> | Define status codes |
| Flow Control | <isif> | Evaluates a condition |
| | <iselse> <iselseif> | Specifying alternative logic when an <isif> condition does not evaluate to true |
| | <isloop> | Creates a loop statement |
| | <isnext> | Jumps to the next iteration in a loop statement |
| | <isbreak> | Terminates loops |
| Variable-related | <isset> | Creates a variable |
| | <isremove> | Removes a variable |

# ISML Tag Categories (2)



| Group | Tags | Purpose |
|---|---|---|
| Include | <isinclude> | Includes the contents of one template on the current template |
| | <ismodule> | Declares a custom tag |
| | <iscomponent> | Includes the output of a controller or pipeline on the current page |
| Scripting | <isscript> | Allows Commerce Cloud Digital 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



Based on B2C Commerce Script (implements ECMAScript standard).

To access a property of the Product object in the pipeline dictionary:

```
${pdict.object.property}
```

Example:

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

ISML Expressions are based on the B2C Commerce Script language. Since B2C Commerce Script implements the ECMAScript standard, access to variables, methods, and objects is the same as using JavaScript.

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

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

ISML expressions can also access B2C Commerce Script classes and methods. Two packages are available implicitly in ISML, so classes do not need to be fully qualified:

- TopLevel package: session.getCustomer()
- dw.web package: URLUtils.url(), URLUtils.webRoot()
  `TopLevel` package has a class named `global` which is implied so doesn't need to be in the prefix.

Other access to classes and methods must be fully qualified:
`${dw.system.Site.getCurrent().getName()}`

## Examples of ISML Expressions



TopLevel package and global class are implicit.

Fully Qualified ISML Expression:

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

Equivalent ISML Expression:

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

You can replace the get method with properties. Equivalent ISML expressions:

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

## ISML Expressions



ISML expressions allow complex arithmetical, boolean, and string operations:
`${pdict.myProduct.getLongDescription() != null}`

Frequently used tags:
`<isset>`, `<isinclude>`, `<isdecorate>`, `<isloop>` and the conditional tags `<isif>`, `<iselseif>`, and `<iselse>`

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

**<isredirect> tag**
This tag can redirect the control to another pipeline and redirect can be permanent or temporary.

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

**<iscomment> tag**
This tag is used to write comments in the ISML. For example.

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

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

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

`MONEY_LONG` prints money with currency symbol e.g. $3,333.00
`MONEY_SHORT` prints money without the symbol e.g. 3,333.00
`DECIMAL` prints the value with two decimal places e.g. 3,455.35
`INTEGER` rounds of and prints only the integer portion e.g. 3,455
`DATE_LONG` prints date in the long format like Jul 24, 2016
`DATE_SHORT` prints date in the long format like 07/24/2016
`encoding="off"` prints strings containing HTML, for example:

`<h1> Welcome to Developing for Digital I Class</h1>` prints as:

Welcome to Developing for Digital I Class

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

You can create and access your own custom variables in an ISML template by using the <isset> tag. When using the <isset> tag, name and value are required attributes that must be assigned. The default scope is session, so you must be careful to qualify your variables accordingly.

Example:

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

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

session scope

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

Retrieving from session

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

request Scope

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

pdict Scope

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

Retrieving from pdict

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

Page Scope

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

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

Retrieving form page

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

## Variable Attributes



### Value Attribute

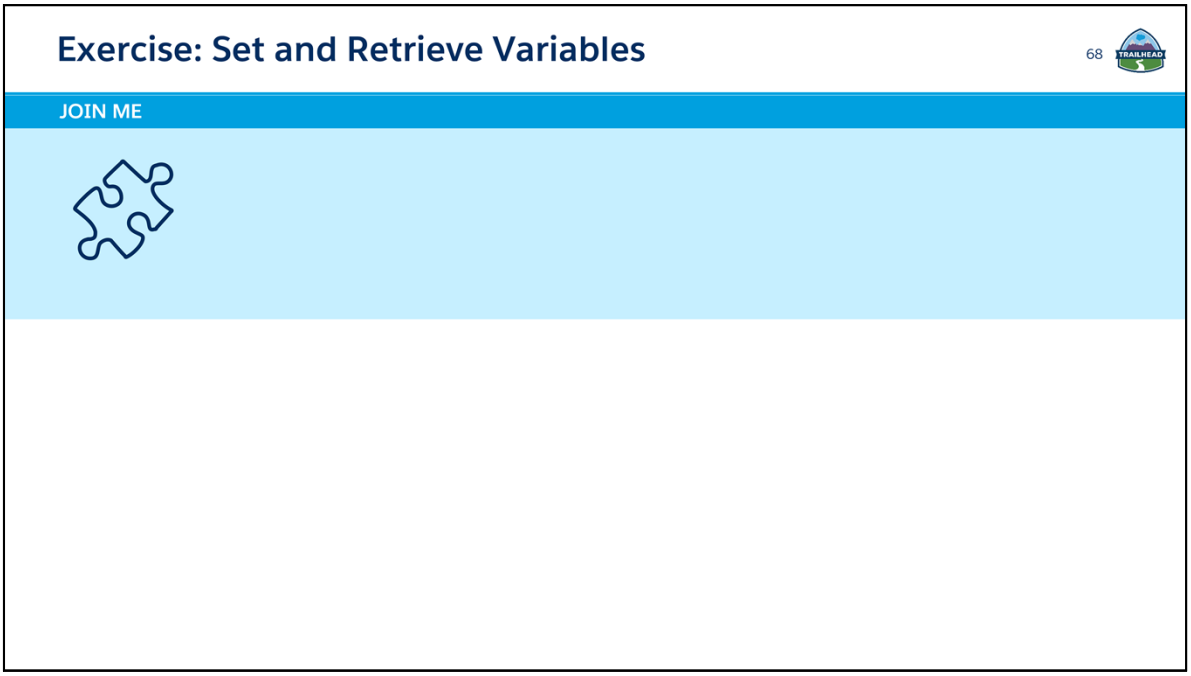
The value attribute can be a hardcoded string or number, or an ISML expression accessing another variable or object.

| Value Type | Example |
|---|---|
| String | value="hardcoded text" |
| Expression | value="${pdict.myProduct.name}" |

### Scope Attribute

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

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




1. Create a controller (using the quickcard as a guide) called `VarTest`. Note: You can use JVarTest.js from `jsolutions` cartridge.
2. Create a new ISML template called `vartest`.
3. In the template, create a new variable called `sessionVar` with hardcoded text as the value and print the value to the page:
   `<isset name="sessionVar" value="${1}" scope = "session"/>`
4. Display the contents of the `sessionVar` variable. Its value is:
   `${session.custom.sessionVar}<br/>`
5. Open a web browser and test the controller.
6. Add similar examples of request and page variables to the `vartest` template and display them.
7. Modify the examples using boolean and string values (i.e., “`${false}`” and “`Hello`”).
8. Test the JavaScript controller again to see the new variable values.
9. Increment the variables using the following syntax:
   `"${request.custom.requestVar + 1}"`

## Lesson 5.3: Reusing Code in Templates



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

To reuse code in ISML templates, use the following tags:

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

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

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

## Local Includes



Syntax:

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

Example:

Template 1:

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

Template 2:
(calendar.isml)

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

Browser Output:

**My Template**
**Included template**

**Local Include Syntax**
`<isinclude template="[directory/]templatename"/>`

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

**Locally include one template into another:**

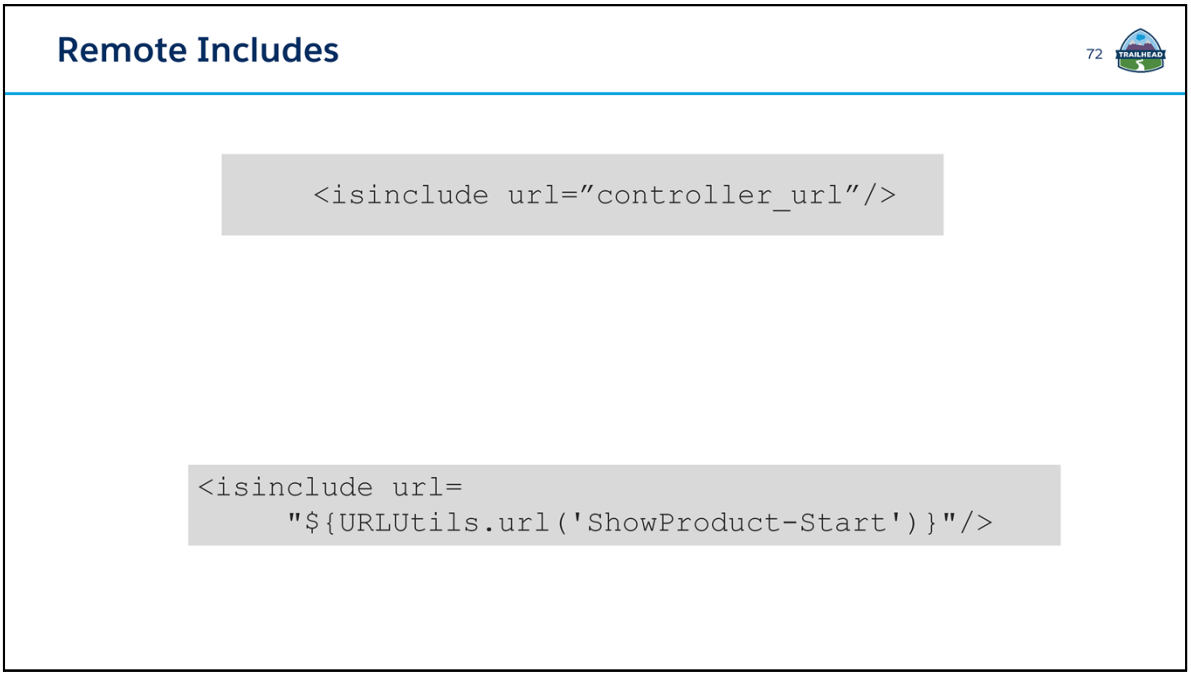
1. Open any ISML template.
2. In the ISML code, determine where you want to embed the locally included template.
3. Add the <isinclude> tag to the template.
4. Save the template.
5. To test, use your template in a JavaScript controller.

## Exercise: Use Local Include in ShowProduct



JOIN ME

1. Study the template to include.
2. Locate and study the `producttile.isml` template.
3. Notice that the first `<isset>` tag expects `pdict.product` in the pipeline dictionary.
4. Include `producttile.isml` in your current template:
5. Open the `JShowProduct` controller that you previously created.
   Note that you will output `myProduct` object on `pdict`, however the `producttile` template expects `pdict.product`. These are not the same variables.
6. Open the `productfound.isml` template.
7. 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"/>
8. Use a local include to display the product tile:
   <isinclude template="product/producttile"/>
9. Test the JavaScript controller with an existing product:
   `JShowProduct-Start?pid=P0048`.




**Remote Includes**
Using a remote include in a template will invoke another controller or pipeline which returns HTML at runtime. This example calls a pipeline without passing URL parameters:
Syntax
```
<isinclude url=”pipeline_url”/>
```

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

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

Here is an example of passing URL parameters:
```
<isinclude url="${URLUtils.https('Product-
GetLowATSThreshold','productid','ETOTE','typeOfTV','Wide-screen')}"/>
```

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

You can also implement a remote include, via the <iscomponent> tag. It supports passing multiple attributes.
```
  <iscomponent
         pipeline = <string> | <expression>
         [locale = <string> | <expression> ]
         [any number of additional arbitrarily named parameters]
  />
```

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

## Use a Remote Include



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

```
<isinclude url="${URLUtils.url('Controller-Function',
['param', 'value', …])}"/>
```

4. Save the template.
5. Test your controller.

## Exercise: Use a Remote Include

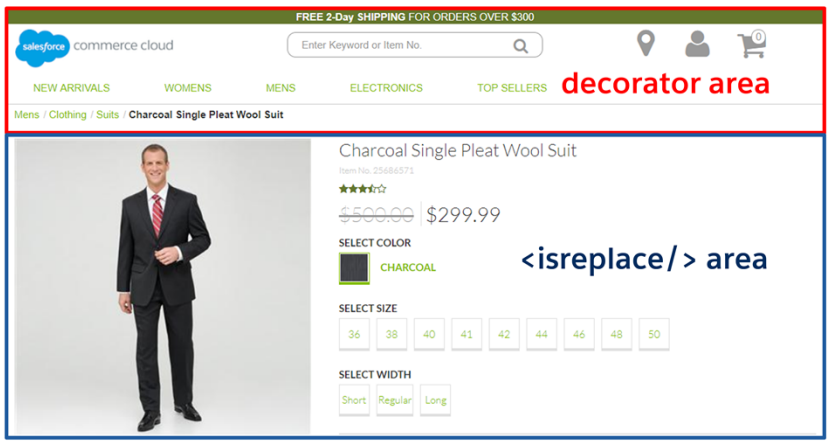


JOIN ME

1. Study the B2C Commerce Script API help for the `dw.web.URLUtils` class, `url()` method.
2. Locate and study the `Product-IncludeLastVisited` (look for `Product.js`) controller in the storefront cartridge.
3. Study the `lastvisited.isml` template, specifically:
   - The use of the <isloop> tag.
   - The use of the `pdict.LastVisitedProducts`.
4. Open the `JShowProduct` controller and the `productfound.isml` template.
5. Add a remote include at the bottom of the template to show the last visited products. Verify your syntax to ensure it is exactly as shown:
   `<isinclude url="${URLUtils.url('Product-IncludeLastVisited') }"/>`
6. Test the controller with an existing product: `JShowProduct-Start?pid=P0048`.
7. On a different browser tab, visit at least three other products in the storefront.
8. Retest the controller: all the visited products should display.

## Use Decorator Templates

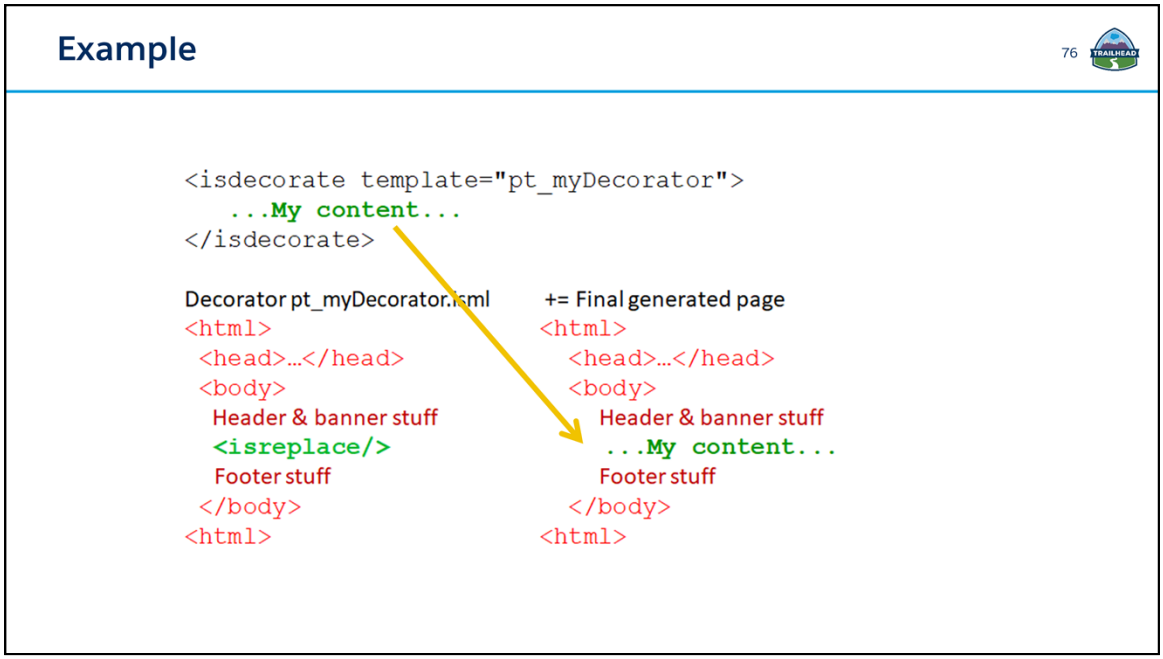





The decorator template uses `<isreplace/>` to identify where to include the decorated content. The above example shows a decorator and the area where the code is being replaced.

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

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




Here is an example:

Template using a decorator

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

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

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

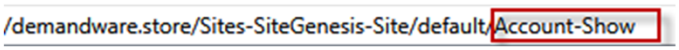
Final generated page

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

# Use the <isdecorate> Tag



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



## Exercise: Use a Decorator

78

JOIN ME

1. In UX Studio, using the Search function, locate the `product/pt_productdetails` template. Notice the different areas of the page this decorator defines.
2. Locate the `<isreplace/>` tag.
3. Open the `ShowProduct` pipeline or `JshowProduct.js` that you created earlier.
4. In your `productfound.isml` template, remove any html, body and head tags as the decorator already contains these.
5. Add the `product/pt_productdetails` decorator so it wraps the existing content on the page:
```
<isdecorate template="product/pt_productdetails">
...existing content...
</isdecorate>
```
6. Test the controller with an existing product: `ShowProduct-Start?pid=P0048` (or `JShowProduct-Start?pid=P0048`)

## Create Custom Tags with <ismodule>



Three key files

1. An isml file which sets the values of any attributes of the custom tag
2. An isml file which specifies what happens when the attributes are passed
3. Invoke the custom tag inside an isml template

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

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

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

- 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>
```

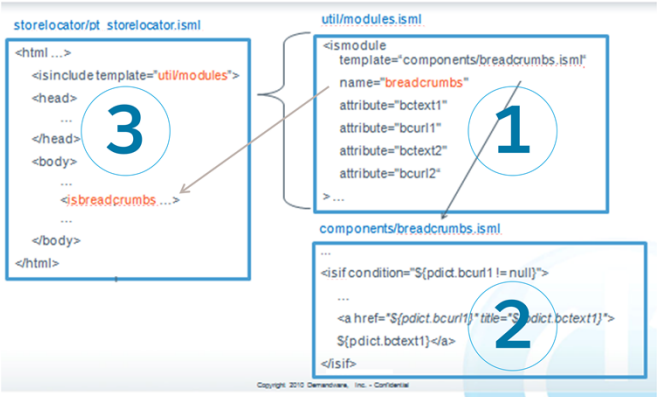
- Invoke the custom tag inside an ISML template:

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

```
</html>
```

## Putting it Together

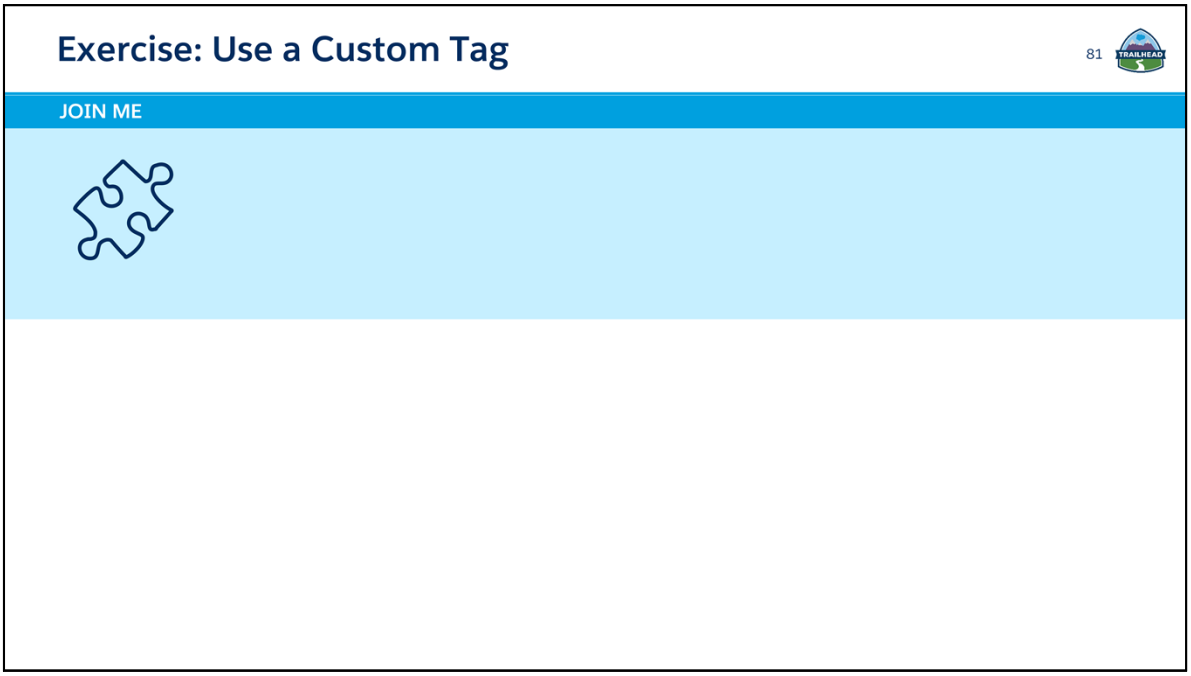





Here is how it would be organized.

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.
2. The isml file which specifies what happens when the attributes are passed.
3. Invoke the custom tag inside an isml template




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

1. Open the `util/modules.isml` template.
2. Locate the `producttile` custom tag definition. Note the different inputs defined for the `producttile` custom tag.
3. Locate the template that implements this custom tag, and study it: `producttile.isml`.
4. Open the `productfound.isml` template that you were referring to from `JshowProduct` controller.
5. Remove the remote include.
6. Change the existing local include to include the template that contains all custom tag definitions: `<isinclude template="util/modules">`
7. Invoke the <isproducttile> custom tag passing the product from the pipeline dictionary: `<isproducttile product="${pdict.myProduct}"/>`
8. Test the controller with an existing product: `JShowProduct-Start?pid= P0048`.
9. In the custom tag invocation, enable other attributes expected by the custom tag. Notice that a proper B2C Commerce script expression is required to specify true or false: `<isproducttile product="${pdict.myProduct}" showswatches="${true}" showpricing="${true}" />`
10. Test the `JShowProduct` controller.

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

Most programming languages use the keywords *if*, *else if*, and else for conditional statements.

Commerce Cloud B2C Commerce uses similar keywords, but adds is to the beginning of the syntax.

## Use Conditional Statements



Determine the location on your ISML page where you want to write your conditional statement.

Open your conditional statement with the <isif condition="..."> tag.

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

To use a conditional statement in an ISML template:

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

## Loops



<isloop>

Supporting tags:

- <isbreak>
- <isnext>

```
<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>
```

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

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

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

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

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

The attributes have the following usage:

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

## Status Variable Properties



- count
- index
- first
- last
- odd
- even

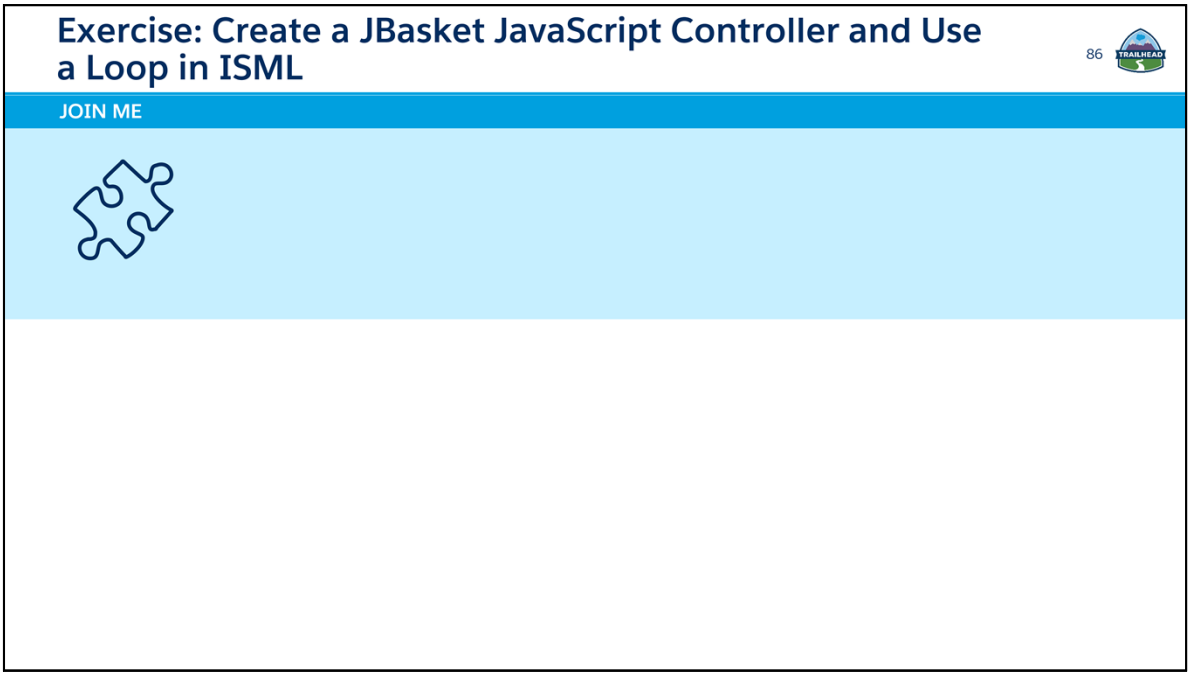
For the status variable, the following properties are accessible:

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

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

Another example of <isloop> tag is:

```
<isloop items="${order.object.shipments}" var="Shipment" status="loopState">

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




1. Review the script API (The instructor will help you with this and visit the package `dw.order`).
2. In this package find the class named `BasketMgr` and property named `currentBasket`. Study what it does. You are going to use it in your code.
3. Create a JavaScript controller named `JBasket.js`.
4. Copy and paste the following template and complete instructions described in the comment.

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

function start() {

      var basket=BasketMgr.currentBasket;

    /*use ISML to display basket on Basket.  The rendered ISML should be
    showBasket.isml
    (Use the quickcard section “Giving control to ISML” for help*/
}

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

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

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

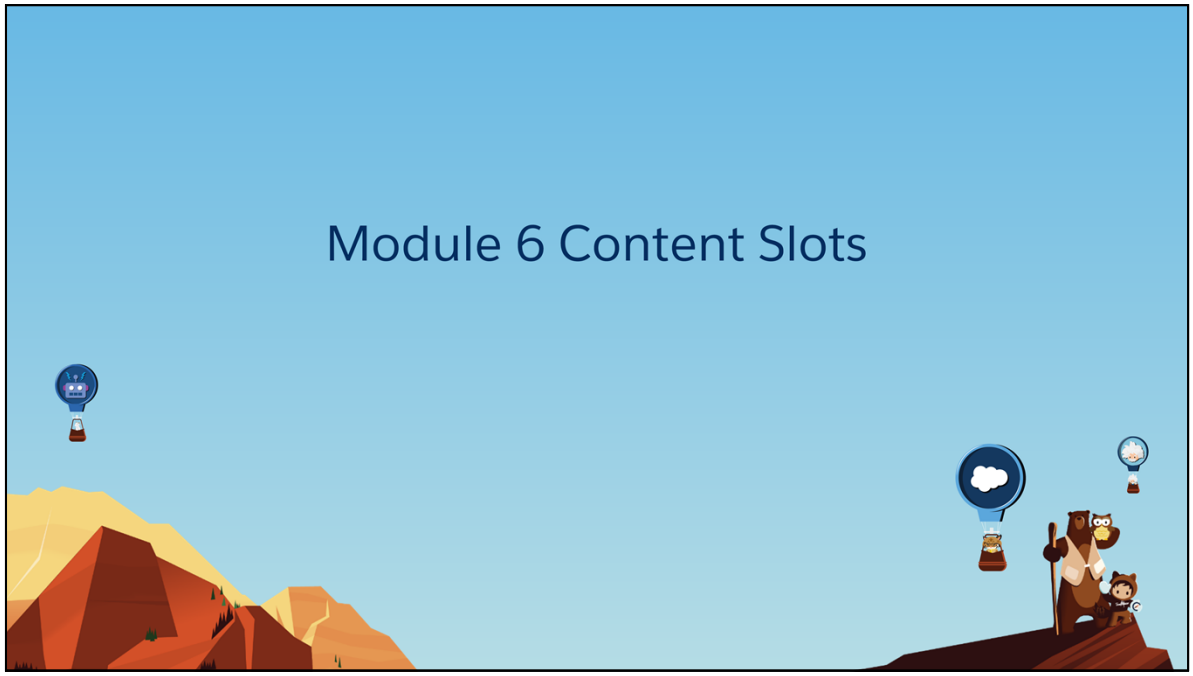
7. Open a browser to your storefront. Add products to the cart first, including a product with an option (like a TV warranty).
8. Open another browser tab and invoke the `Basket-Start` controller.
9. Add a `status` attribute in the <isloop> tag so that you can see what the count and index parameters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

## Knowledge Check 5



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?

# Module 6 Content Slots



## Module 6: Content Slots

89

**Objectives:**
Create content slots for products and images.
Use rendering templates with content slots.
Configure content slots.

**Lessons:**
6.1 Creating and Configuring Content Slots
6.2 Using Content Link Functions

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

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

## Content Types

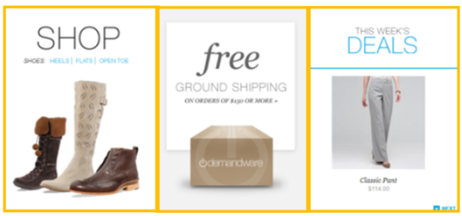


- One or many products selected by the merchant
- Category attributes (images or other visual)
- Content assets from the content library
- Static HTML and images from the static library
- PI Recommendations

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

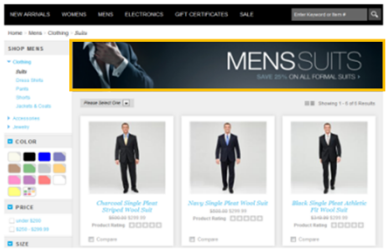
## Content Slot Types






**Global Slots**
Display on any page

**Category Slots**
Display on category pages



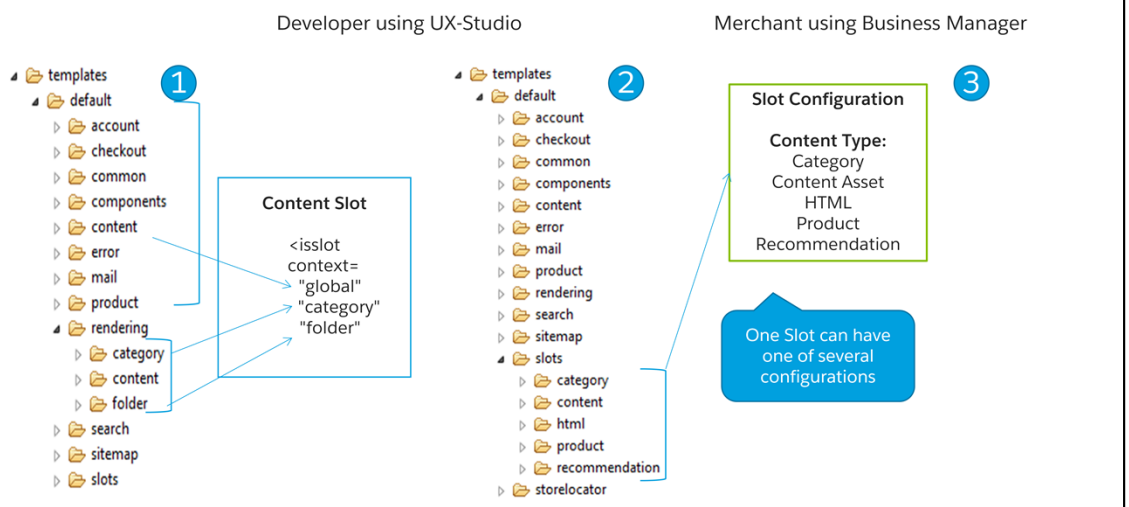

**Folder Slots**
Display on content folder pages

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

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

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

# Lesson 6.1: Creating and Configuring Content Slots




Creating a content slot requires a collaborative effort:

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

## Developer Task - Use <isslot>



Global slot example

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

Category slot example

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

Folder slot example

```
<isslot id="fldr-landing-slotbanner" context="folder"
     description="whatever…"
     context-object="${pdict.ContentSearchResult.folder}"/>
```

You create a content slot inside a template using the `<isslot>` tag. The tag must be located exactly where it should appear on the page.
Note: The attribute “description” is required for the content slot to work. If you leave out the description, the page won't render, an error occurs, and the message in the Request Log won’t give you much helpful information.

Whenever the template is saved, the new content slot automatically displays in the list of slots under **Site > Online Marketing > Content Slots** (this occurs because Commerce Cloud B2C Commerce scans any template for the use of the `<isslot>` tag).

## Developer Task - Create Slot Rendering Template



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

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

The slot displays the 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.

## Developer Task - Use slotcontent and <isprint>



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

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

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

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

```
<isif condition="${slotcontent != null}">
```

Then it loops through the slotcontent.content (the content provided for the slot):

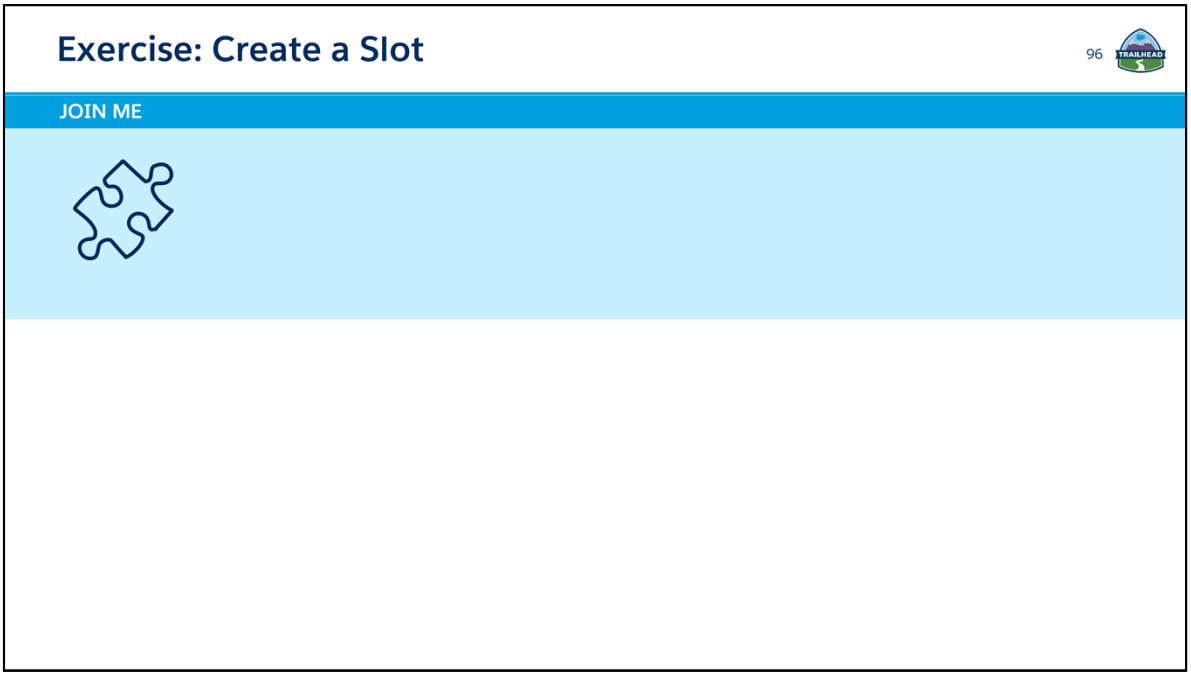
```
<isloop items="${slotcontent.content}" var="markupText">
    <isprint value="${markupText.markup}" encoding="off"/>
</isloop>
```

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

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

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

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

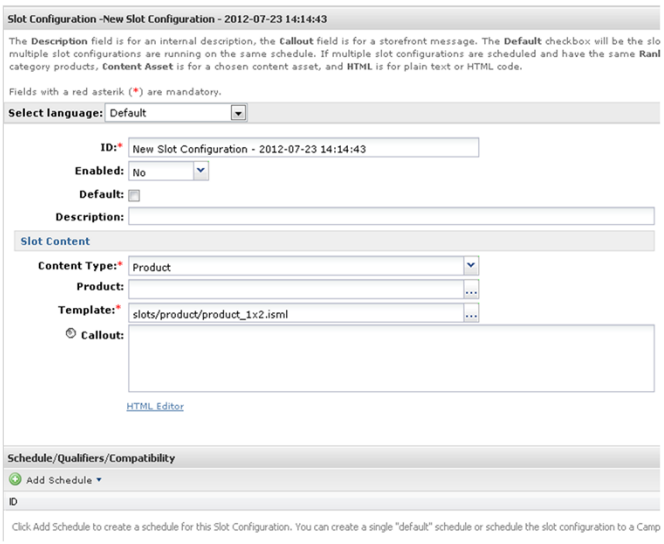



Create a banner slot containing an image on the `nohits.isml` template. This template displays when a search does not return any products.

1. Use the storefront search box and search for a product which does not exist.
2. Investigate which template is used to generate that page (which tool would you use to find that out?).
3. Copy the found template from the storefront cartridge to the exact same location into your cartridge.
4. Before the `no-hits-footer div`, add a global slot:
```
<isslot id="search-no-hits-banner"
description="recommendations banner for search no results page"
context="global" />
```
5. Study the `htmlslotcontainer.isml` rendering template that is used to render HTML-type slots.

# Merchant Task - Create Content Slot Configurations






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

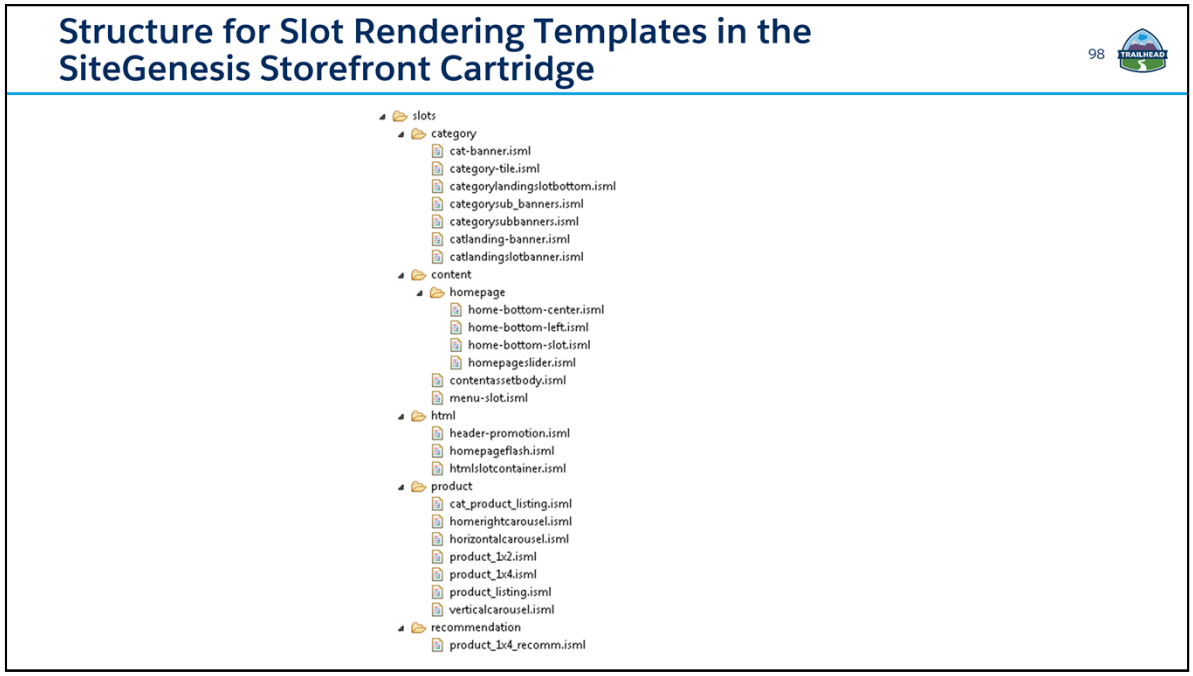
The merchant selects the type of content, for example, Product or HTML. Different fields display depending on the content type selected, for example:

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

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

## Structure for Slot Rendering Templates in the SiteGenesis Storefront Cartridge






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

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

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

### Value Attribute

```
$staticlink$
$url()$
$httpUrl()$
$httpsUrl()$
$include()$
```

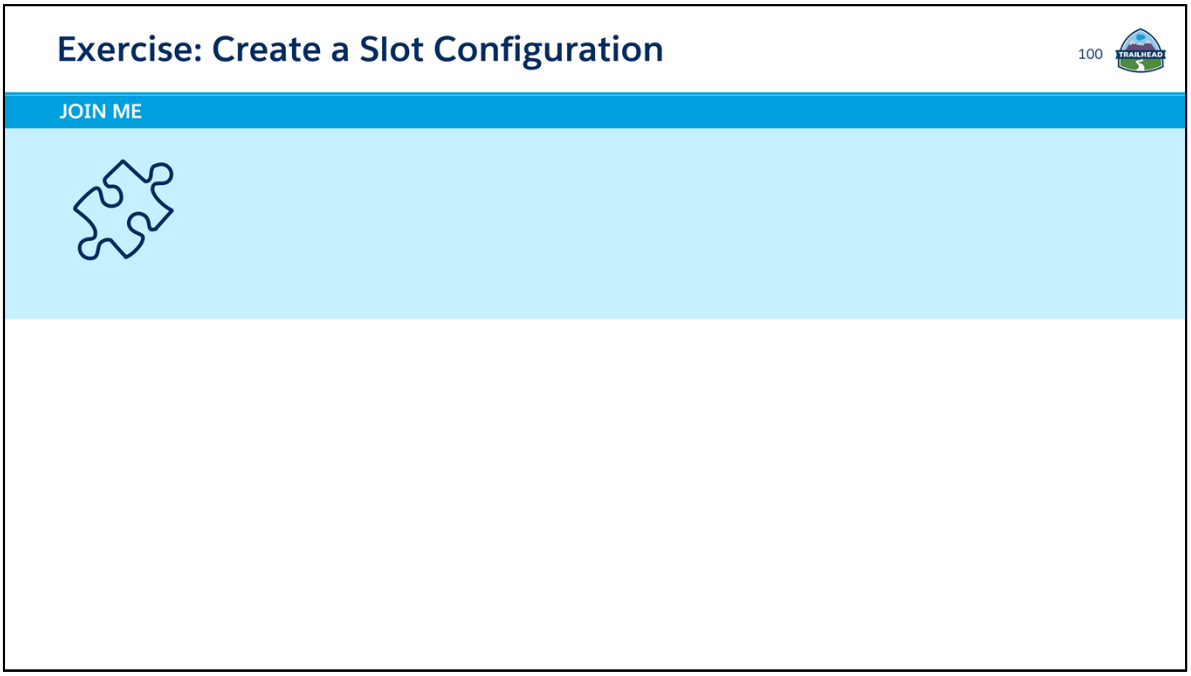
```
href="$url('Page-Show', 'cid', '2-day-
          shipping-popup')$"
```

B2C Commerce uses attributes of type HTML in many places: content assets, content slots with HTML-type content, product descriptions, etc. You can also add an attribute of type HTML to any system object where you may need to show HTML. These attributes are represented by the class `dw.content.MarkupText`.

**Note:** When using HTML in content assets or content slots, avoid hardcoding hyperlinks to pages or images in the storefront. They are instance-specific (e.g., Staging) and would have to be changed every time after a replication. Instead, B2C Commerce offers the following Content Link Functions for use in attributes of type HTML:

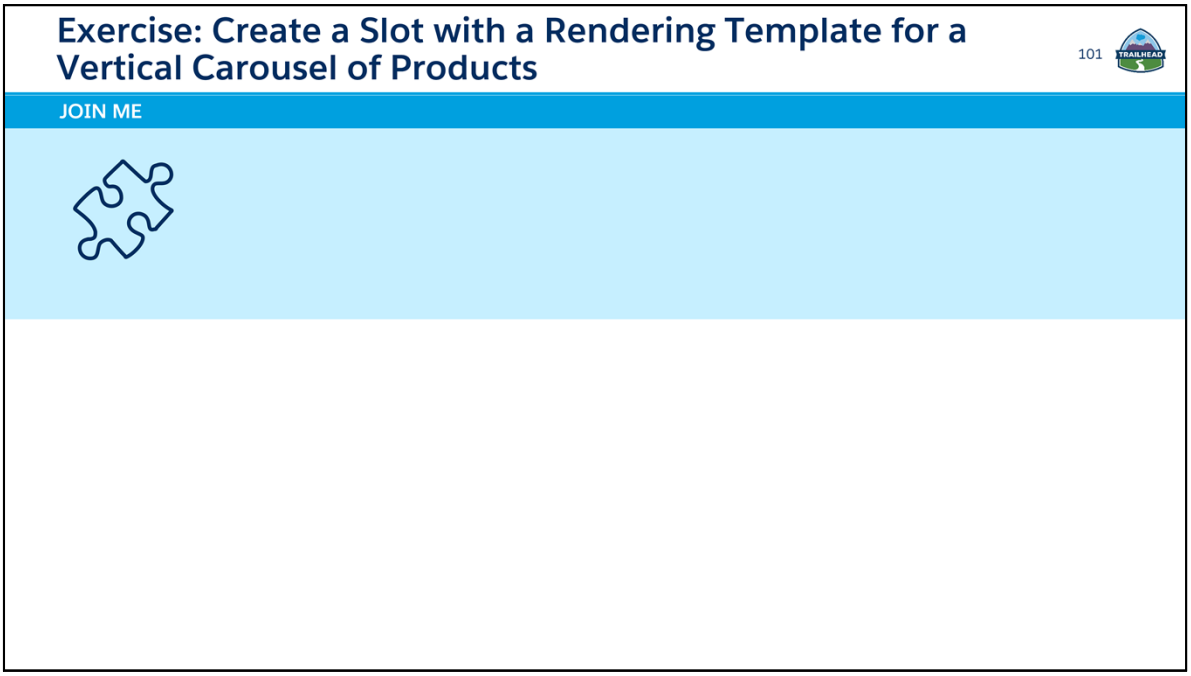
- `$staticlink$` - Creates a static link to an image.
- `$url()$` - Creates an absolute URL that retains the protocol of the outer request.
- `$httpUrl()$` - Creates an absolute URL, with the http protocol.
- `$httpsUrl()$` - Creates an absolute URL, with the https protocol.
- `$include()$` - Makes a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

Here is an example of a function that creates a hyperlink to the `Page-Show` controller passing `cid=2-day-shipping-popup` in the query string:
`href="$url('Page-Show', 'cid', '2-day-shipping-popup')$"`




Complete the configuration for the content slot created previously.

1. In Business Manager, select **Site > Online Marketing > Content Slots**.
2. Locate the new `search-no-hits-banner` slot in the global section. Create a new configuration for the slot.
3. Provide an ID: banner-for-everyone.
4. Enable it.
5. Make it the default.
6. Select HTML for the content type.
7. In the HTML editor:
   a. Click the Insert/Edit Image icon.
   b. Click **Browse Server.**
   c. Locate the `/images/slot/` directory and select it.
   d. On the Upload File section, find the `nohits.png` image in the `contentslot` cartridge, `static/default` folder, and upload it.
   e. After uploading, select the image.
   f. The generated HTML should look like this:
   ```
   <p><img width="700" height="100" src="nohits.png?$staticlink$" alt=""
   /></p>
   ```
8. Select `slots/html/htmlslotcontainer.isml` as the rendering template for the slot.
9. Click **Add Schedule > Default Schedule** to ensure that the slot displays continuously.
10. Click **Apply** to save the configuration.
11. Test the slot by searching for some non-existent product: the `nohits` page should display with the new slot visible.




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

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

```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<iscache type="relative" minute="30" varyby="price_promotion"/>
<isinclude template="util/modules"/>
<h2>${Resource.msg('global.carousel.featuredproducts','locale',null)}</h2>

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

```
            <div class="productcarousel">
           </div><!-- END: productcarousel -->
        </li>
    </ul>

    <a class="jcarousel-prev" href="${'#'}"></a>
    <a class="jcarousel-next" href="${'#'}"></a>
</div>
<!-- END: verticalcarousel -->
```

7. Create a template named `pt_productsearchresult_UI.isml` in the following location in the training cartridge:
   `training/cartridge/templates/default/search`
8. Add the following line to point to a new CSS file that redefines the vertical carousel styling:
   `<link href="${URLUtils.staticURL('/css/verticalcarouselx4.css')}" type="text/css" rel="stylesheet"/>`
9. Copy the whole folder 'static' from the contentslot cartridge to your training cartridge.
10. Copy the `pt_productsearchresult_nohits.isml` from the storefront cartridge into your cartridge to the same location (create the folder structure if it is not present).
11. The event handler for our buttons is in the Namespace named `storefront`. So modify the script block to match the following:

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

12. In Business Manager, select **Site > SiteGenesis > Online Marketing> Content Slots**.
13. 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).
14. Go to the storefront from Business Manager in the browser. Search for some non-existent product like `MyBestPants`.
15. Verify that the `nohits.isml` shows both the previous banner and multiple products in a vertical carousel.

# Knowledge Check 6



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

# Module 7 Commerce Cloud B2C Commerce Script

## Module 7: Commerce Cloud B2B Commerce Script



**Objectives:**

- Describe the Commerce Cloud B2C Commerce Script syntax.
- Describe the B2C Commerce Script API packages.
- Use B2C Commerce Script in ISML.
- Debug B2C Commerce Script in UX Studio.
- Use the Resource API and resource bundles.

**Lessons:**

7.1 Creating and Configuring Content Slots
7.2 Script and JavaScript Controller Debugging
7.3 Resource API and Resource Bundles

Commerce Cloud B2C Commerce Script is the server-side language used for coding in Commerce Cloud B2C Commerce.
It is based on JavaScript, which is standardized as ECMAScript. It implements ECMA-262 and the ECMA-357 standard, also known as ECMA for XML or E4X.
It supports all JavaScript language extensions by Mozilla known as JavaScript 1.7 as well as optional type specification (from JavaScript 2.0/ECMA 4th edition proposal and ActionScript).

Use Commerce Cloud B2C Commerce Script to access data about the system, such as: products, catalogs, prices, etc. You write B2C Commerce Script in controllers and in ISML templates for expressions or inside `<isscript>` tags.

# B2C Commerce Server Script – ISML Example

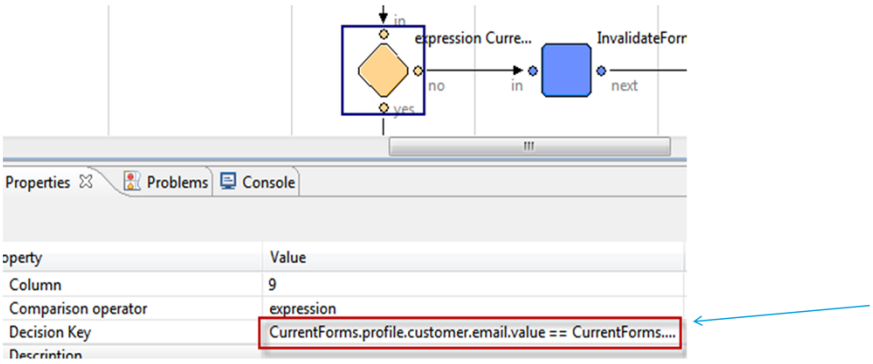


```
<isscript>
    var cat = pdict.ProductSearchResult.category;
    var path = new dw.util.ArrayList();
    while( cat != null && cat.parent != null )
    {
        if( !cat.online )
        {
            cat = cat.parent;
            continue;
        }
        path.addAt( 0, cat );
        cat = cat.parent;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

No change with controllers

# B2C Commerce Server Script – Decision Node Example






Not applicable with controllers

## Lesson 7.1: Commerce Cloud B2C Commerce Script API



Script APIs organized into packages that include classes and methods for Commerce Cloud objects.

The objects and methods in all packages besides `TopLevel` need to be imported or fully qualified.

| Package | Overview |
|---|---|
| **`TopLevel`** | General purpose classes, including data types. The `TopLevel` package is available implicitly to all scripts and does not have to be fully qualified when its objects and methods are used. |
| **`dw.campaign`** | Classes for managing campaigns , promotions, and A/B tests. This class is covered in the "Working with the `dw.campaign` Package" module. |
| **`dw.catalog`** | Classes for managing catalogs, categories, products, and prices. This class is covered in "Working with the `dw.catalog` Package" module. |
| **`dw.content`** | Classes for managing content assets, library folders, and the content library of the current site, including content searches. |

Each new B2C Commerce update includes a well-documented API. The Script is available under the Studio Help menus. The ISML documentation is available in the B2C Commerce documentation.

Salesforce continually updates clients to the latest version. Deployments happen globally on Tuesday and Thursday between 2 and 7 am local POD time. The current version number displays at the bottom of the Business Manager screen and it corresponds to *Year.Deployment*, for example, version 18.8 represents the eighth deployment in 2018.

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

The Global Release Process ensures that all Salesforce 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.

## Script APIs



| Package | Overview |
|---|---|
| `dw.customer` | Classes for managing customer data including customer profiles, credentials, customer groups, order history, product lists, and payment instruments. |
| `dw.io` | Classes for input/output supporting XML streams, CSV files, text files, and random access files. |
| `dw.net` | Classes for handling FTP, HTTP, SFTP, WebDAV, and email operations. |
| `dw.object` | Classes for managing system and custom objects, and their attributes in the Commerce Cloud framework. |
| `dw.order` | Classes for managing shopping carts (baskets), orders, product and shipping line items, and payment processors. This class is covered in the "Working with the `dw.order` Package" module. |
| `dw.system` | Classes that manage hooks, internal objects, jobs, logs, pipeline dictionaries, requests, responses, sessions, sites, site preferences, status, and system. |
| `dw.util` | Utility methods for managing objects such as arrays, calendars, collections, currency, dates, hash maps and sets, lists, locales, maps, and sets, templates, and UUIDs, among others. |
| `dw.web` | Classes for managing web sessions and pages. The `dw.web` package is available implicitly to all scripts and does not have to be fully qualified when its objects and methods are used. |

API Packages

The B2C Commerce Script API is organized in packages, just like Java. Unlike Java, inheritance is not possible from these classes or packages when you create a script. You can only use the properties and methods of these classes in your scripts.

In Commerce Cloud B2C Commerce Script, the TopLevel package is the default package. It is similar to `java.lang` in Java. It does not need to be imported in scripts. It provides standard ECMAScript classes and extensions, such as: `Error`, `Date`, `Function`, `String`, `Math`, `Number`, `XML`.

The `TopLevel.global` class contains many of the common constants, and properties used in scripts. Some properties are: `customer`, `request` and `session`.

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

# Generic API Packages



| Package | Overview |
|---|---|
| **dw.crypto** | Encryption services using JCA; DES, Triple-DES, AES, RSA, etc.<br>Classes: `Cipher, MessageDigest` |
| **dw.io** | Input and output<br>Classes: `File, FileReader, CSVStreamReader, XMLStreamReader`, etc. |
| **dw.net** | Networking<br>Classes: `FTPClient, HTTPClient` |
| **dw.object** | System base classes and custom objects<br>Classes: `PersistentObject, ExtensibleObject, CustomObjectMgr`, etc. |
| **dw.rpc** | Web services related APIs<br>Classes: `WebReference, Stub` |
| **dw.system** | System functions<br>Classes: `Site, Request, Session, Logger` |
| **dw.util** | Similar to the `java.util` API: `collections`, `maps` and `calendar` classes |
| **dw.value** | Immutable value objects<br>Classes: `Money, Quantity` |

## Use B2C Commerce Server Script in ISML

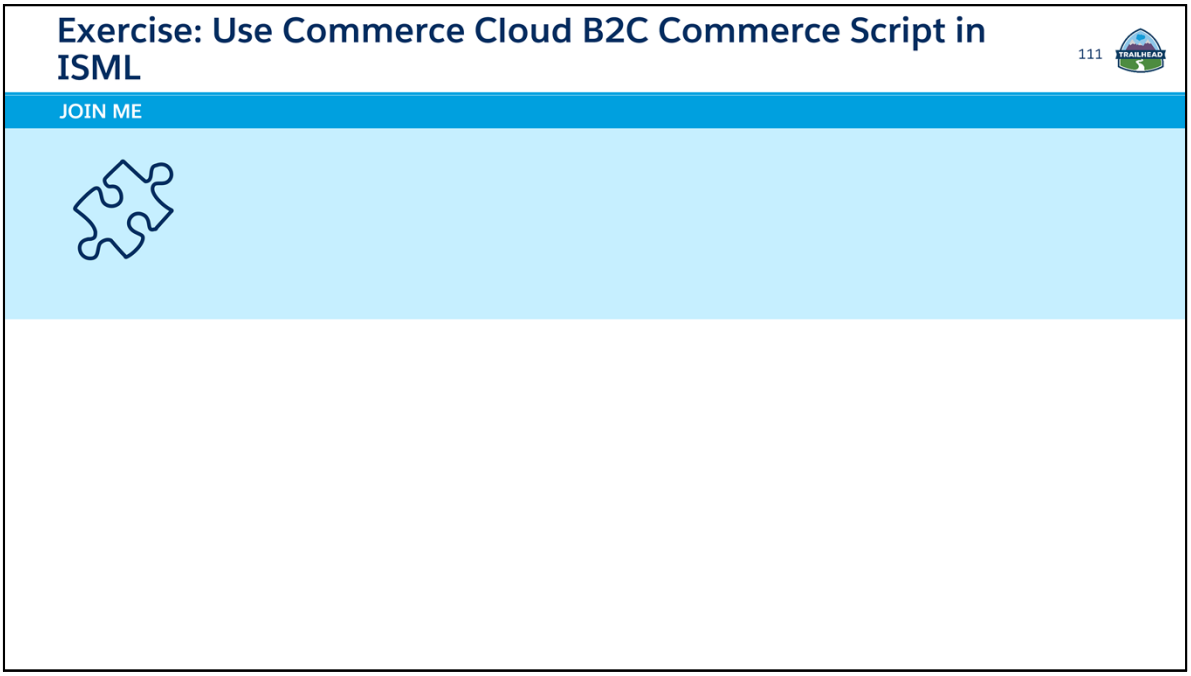


```
<iscomment>
    This template displays a 3-level category tree as top navigation.
    Only categories marked with showInMenu are shown.
</iscomment>

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

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

You can embed Commerce Cloud B2C Commerce Script into ISML by using the `<isscript>` tag. This example uses B2C Commerce Script to get the root category of a current site's navigation catalog and the category named 'sale'.

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

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

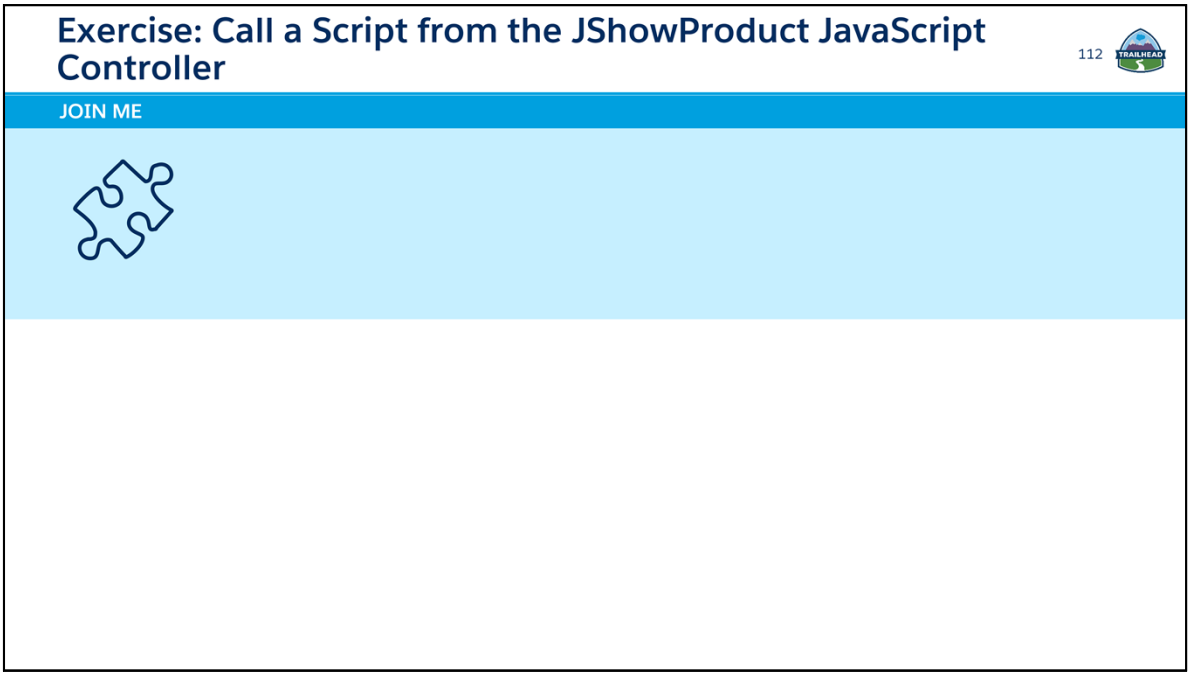



1. Create a new JavaScript controller called `JDScript`.
2. Create a new ISML template named `dscript.isml` and display it using the controller.
3. Using the `dw.customer.CustomerMgr` class, print the registered customer count.
4. Test your controller in the storefront.

**Calling a script with JavaScript Controller**
A script can be invoked by using 'require' to get Script and then invoking method on it. For example the following piece of code can invoke the script method `doJobForMe(…)`

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

**Using the * wildcard to fetch it using the cartridge path:**
Option 1: ~ indicates the current cartridge from where the script is being invoked.
Option 2: If a script is invoked from another cartridge, then that cartridge still has to be in the cartridge path and you have to mention * in the require statement.
Option 3: If a script needs to be invoked from another cartridge but not using the first-come-first-serve functionality of the cartridge path, the cartridge can be hardcoded in the require statement.




1. You can keep a backup copy of the `JShowProduct` that you previously created.
2. Copy `jsolutions/cartridge/scripts/ProductFinder.js` to your `training` cartridge in the same location.
3. In the present controller, remove all the previous code and copy and paste the following template

```
'use strict';
/** @module controllers/JShowProductCallingScript */

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

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

    var product=/* Use the quickcard section “Invoking a Script” again to invoke
the method on ProductFinder */

    if (product==null) {
            ISML.renderTemplate(
                    'productnotfound.isml',
                    {  Log:'product with id '+parameterId+' not found'  }
            );}
       else{
            ISML.renderTemplate(
                    'productfound.isml',
                   {   myProduct:product  }
             );
       }}
exports.Start = guard.ensure(['get'], start);
```

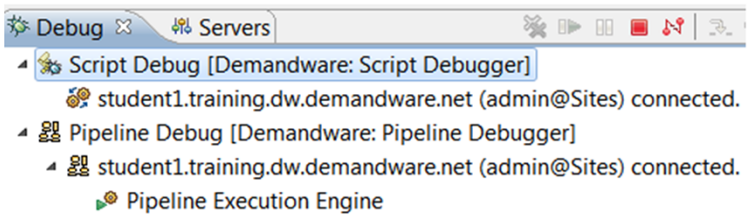
4. If not done already, modify your `productnotfound.isml` template so that it displays the contents of the Log as follows: `${pdict.Log}`

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

## Lesson 7.2: Script and JavaScript Controller Debugging

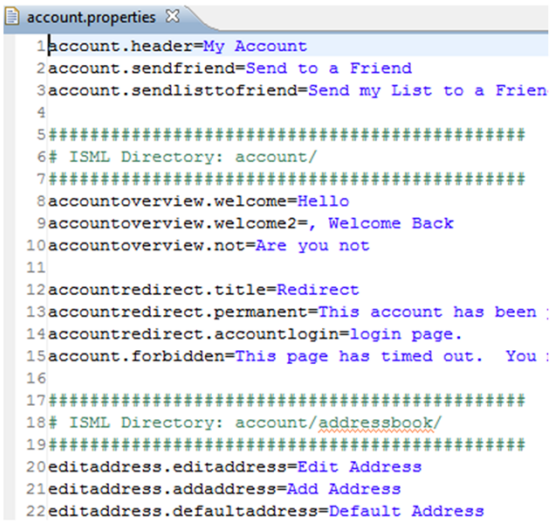


```
19 function handleForm() {
20     var TriggeredAction = request.trigg
21       response.getWriter().println('Hel
22       if ( (TriggeredAction != null) &&
23         //response.getWriter().println(
24             ISML.renderTemplate('newsle
25                 CurrentForms     : sessi
```




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

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

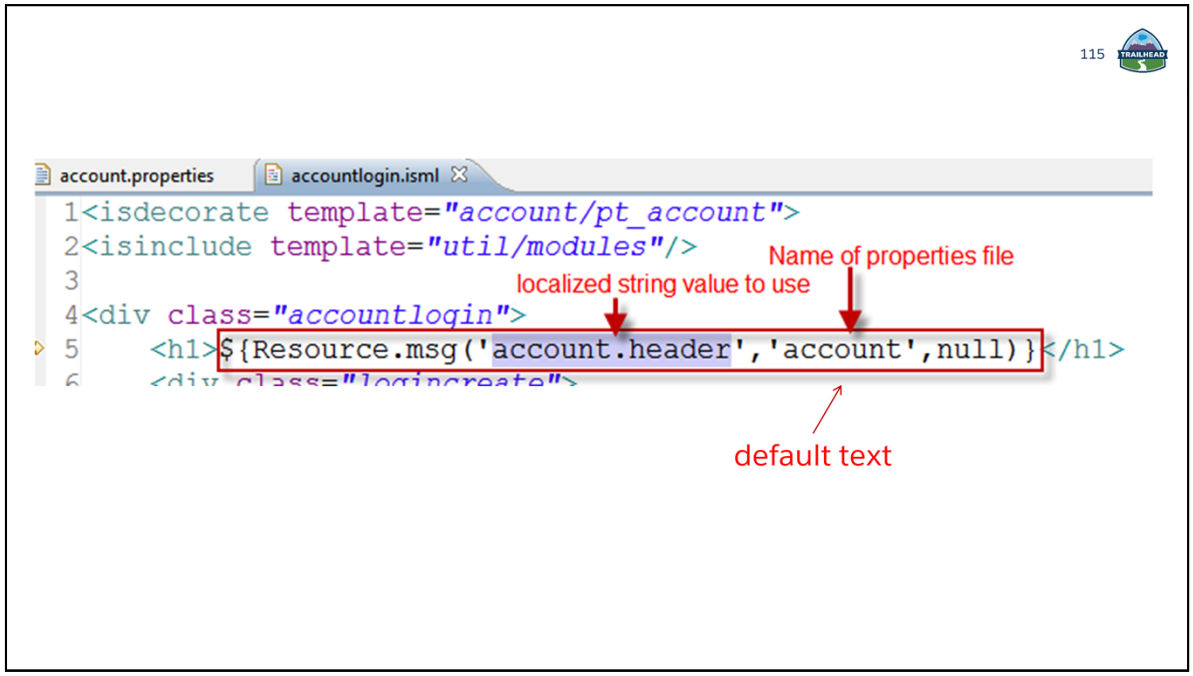



In storefront code, avoid hard-coding text strings that become visible to the user. Titles, labels, messages, button and field names should all be externalized by using resource bundles (a.k.a. properties files.). If you do not want to duplicate ISML templates in order to create locale-specific templates,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

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

Resource Bundle Editor is a very useful tool from the Eclipse Marketplace. It allows consistent handling of multi-locale bundle files.

**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”).

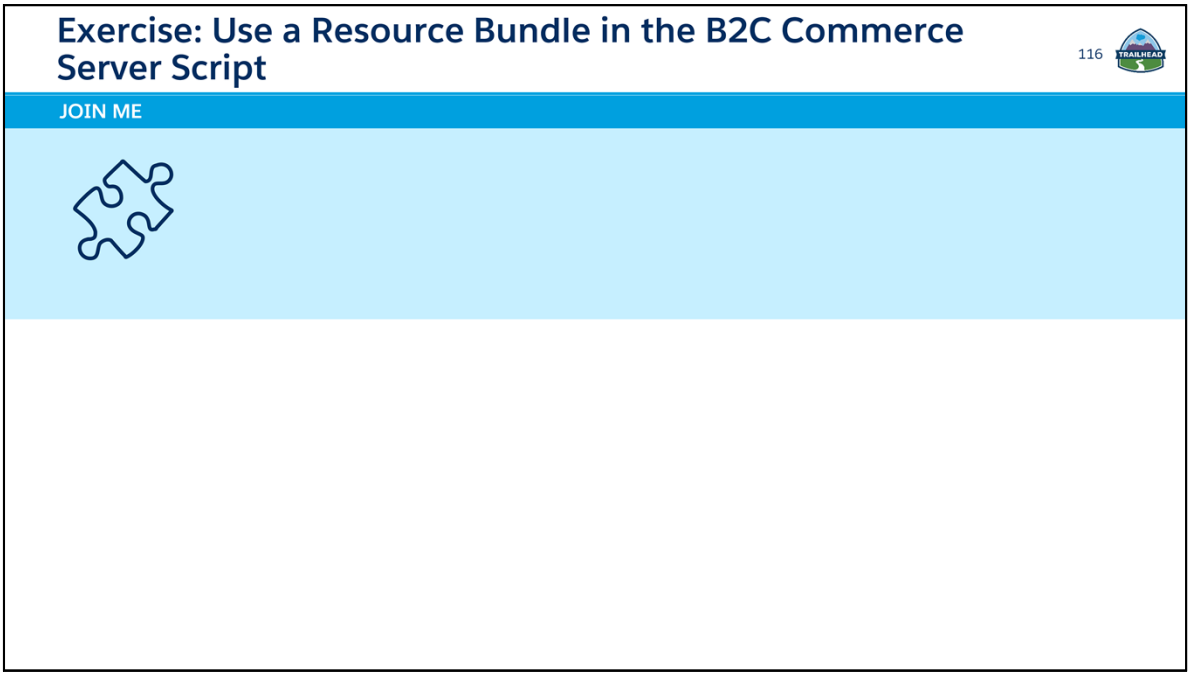



The resource bundles contain **key=value** pairs where the key might be compound (`key.subkey`) and the value is a hard-coded string that uses Java MessageFormat syntax to implement parameter replacement. Bundles are stored in each cartridge within the `/templates/resources` directory.

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 in any resource bundle. Instead of the `null` you can also show a string to display on the storefront in case the key could not be found.

Another useful method is the `dw.web.Resource.msgf(key : String , bundleName : String , defaultMessage : String , pdict.<Object>.<property>...)`. Using this method, you can specify a key with placeholders which can be dynamically replaced by the parameters specified in the `args` argument of the method. For example, this usage of the method:
`${Resource.msgf('singleshipping.wishlist', 'checkout', null, owners.get(addressKey).profile.firstName )}`
will be paired with the following Java MessageFormat definition in the resource bundle to allow the first name of the wishlist's owner to show up as **Stefan's Wishlist**:
singleshipping.wishlist={0}\'\'s Wishlist




1. Open `controllers/JShowProduct.ds`. Inside this controller, after checking if the product is null, add the following line of code to pick up a string `productnotfoundMsg` from the resource bundle named `myBundle.properties`:
   ```
   var errorMsg=dw.web.Resource.msgf('productnotfoundMsg', 'myBundle', null, parameterId);
   ```
2. Using the quickcard as a guide (section "Giving control to ISML"), edit the next line to use the ISML to render the template `productnotfound.isml` and pass the JSON code `{message:errorMsg}`.
3. Create a file `/templates/resources/myBundle.properties` with the following content (create the folder structure if not already there).
   ```
   productnotfoundMsg=The product with the id {0} is not found
   ```
4. Create a file `/templates/resources/myBundle_fr.properties`.
5. Change the encoding of this file to UTF8 to support French characters. Right-click `myBundle_fr.properties` and change the default encoding to UTF-8.
6. In myBundle_fr.properties enter:
   ```
   productnotfoundMsg = Le produit avec l\'\'ID {0} ne est pas trouvé
   ```
7. In Business Manager, select **Site-SiteGenesis > Site Preferences > Locales.** Check 'fr' and click **Apply**.
8. Run the `ShowProduct` pipeline or `JShowProduct` controller as you have been running before. Your URL will be similar to:
   `https://studentXX.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/JShowProduct-Start?pid=452345`
   Note: Replace the xx with your student id.
   You should see the message of the product not being found in English language
9. In the URL, replace 'default' with 'fr'.
10. You should see the results in French.
11. Internationalize the product name. In Business Manager, select **Products & Catalogs > Products.**
12. Search for the product with id 'P0048'.
13. Click the product link and lock it for editing.
14. Notice that the name of the product in 'default' language is 'Laptop Briefcase with wheels (37L)'. Change Select Language dropdown to 'French'. The name entry will be now blank. Paste 'Laptop Briefcase avec des roues (37L)' without quotes in the name. Click Apply.
15. In your isml that displays your product (`productfound.isml`), enter the following to print the product name (remove the earlier text):
    ```
    ${Resource.msgf('productfoundMessage', 'myBundle',null, pdict.myProduct.name)}
    ```
16. In `templates/resources/myBundle_fr.properties` add the following:

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

17. In `templates/resources/myBundle.properties` add the following:
    `productfoundMessage=The product is found and the name is {0}`
18. Run the `ShowProduct` pipeline or `JShowProduct` controller with the parameter pid=P0048.
    Your URL will be similar to:
    `https://studentXX.training-na02.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default/JShowProduct-Start?pid=P0048`
    Note: Replace the XX with your student id.
    You should see the product page with product name in English.
19. In the URL, replace ‘default' with 'fr‘.
    You should see the results in French.

Note: If a breakpoint could not be set because of an incorrect Eclipse package, install additional web tools using the respective links:
http://download.eclipse.org/webtools/repository/mars or
http://download.eclipse.org/webtools/repository/neon.

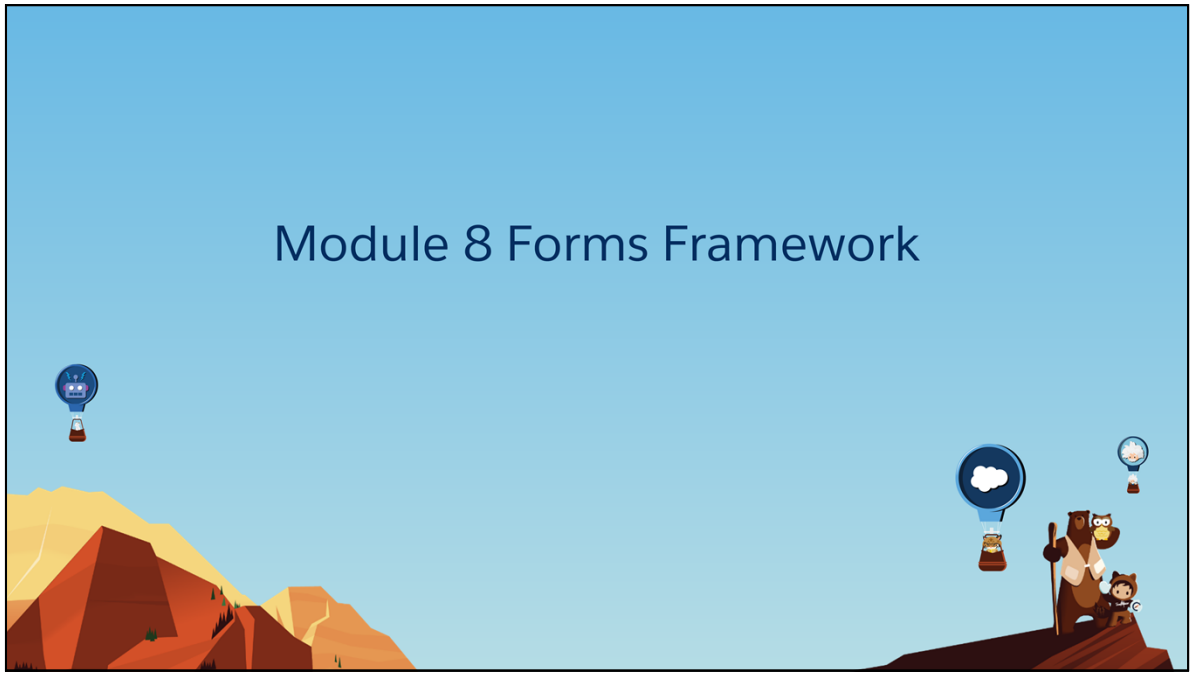
# Knowledge Check 7



**True or false:**

1. You can rely on the cartidge path logic when requiring a script from another cartridge into a script file.
2. If cartridgeA invokes a script from cartridgeB, cartridgeB does not need to be in the cartridge path.

# Module 8 Forms Framework



## Module 8: Forms Framework



**Objectives:**

- Describe the concepts and usage of the Commerce Cloud B2C Commerce Forms framework.
- Create a new form and implement it in a pipeline.

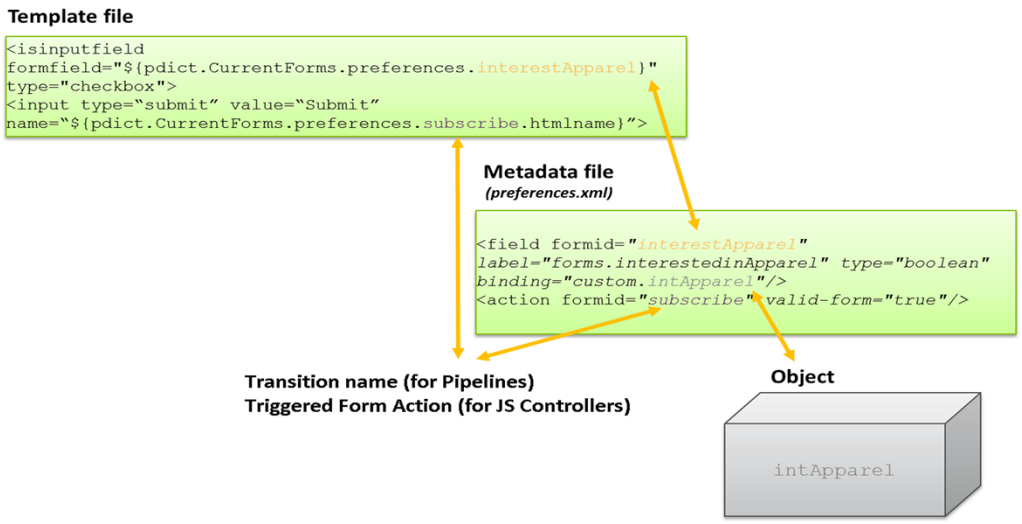
**Lessons:**

8.1 XML Metadata File
8.2 ISML Form Template
8.3 Pipeline Elements

Commerce Cloud B2C Commerce provides tools to simplify form display and processing. Use the B2C Commerce Forms framework to control how consumer-entered values are validated by the application, rendered in a browser, and possibly stored on a server.

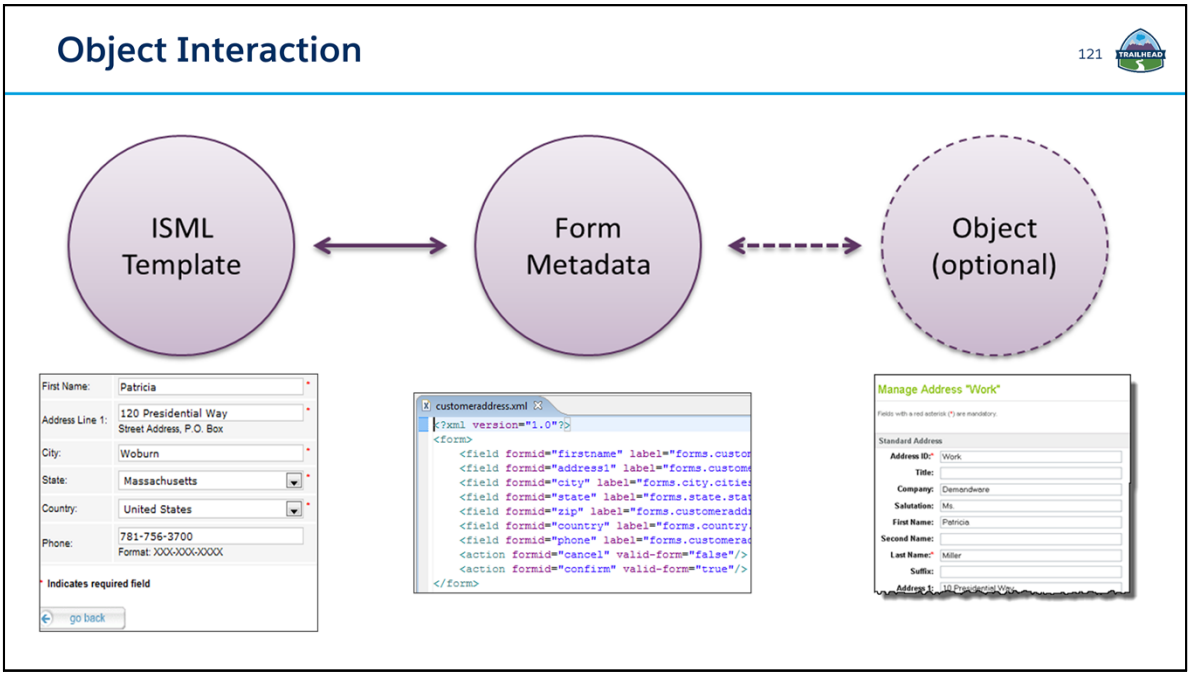
## Forms Framework






To use the B2CCommercelForms framework, you need the following files:

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




There are three objects that interact when working with B2C Commerce forms:

- XML metadata file: located in the `cartridge/forms/default` directory. It describes the fields, labels, validation rules and actions that apply when the field is used in an ISML template.
- ISML template: it uses the form metadata fields and actions to show an HTML form to the user.
- Object (optional): this object represents a single system or custom object in the `pdict`, and it can be used to pre-fill the metadata file as well as to store submitted form data to the database.

Example given this form metadata XML file.

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

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

## Lesson 8.1: XML Metadata File



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

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

The form metadata file uses the above xml elements.

# Field Element May Use the Following Attributes



| Attributes | Description |
|---|---|
| `formid` | Required: unique ID to identify the field for ISML templates and controllers. |
| `Type` | Required: data type for field (see table below). |
| `label` | Usually a key to an externalized string in the `forms.properties` resource bundle. |
| `description` | Description for field, might be used in tooltips. |
| `min-length, max-length` | Restricts the field length for data entry. |
| `min, max` | Valid range for integer, number and dates. |
| `range-error` | Message shown if value provided does not fall within the specified range. |
| `regexp` | Regular expression for string fields: email, phone, zip code, etc. |
| `parse-error` | Message shown when the data entered does not match the regex. Usually a key to an externalized string. |
| `mandatory` | Field is required via server-side validation when true. |
| `missing-error` | Message shown if the primary key validation error is generated in a pipeline. |
| `value-error` | Shown if an element is invalidated in a pipeline. |
| `Binding` | Used to `match` field to a persistent object attribute. |
| `Masked` | Specify # of characters to mask. |
| `Format` | Format for display of dates, numbers, etc. |
| `whitespace` | Specify whitespace handling (none or remove). |
| `timezoned` | Optional flag for date objects (true or false). |
| `default-value` | Pre-defines a value for a field. |
| `checked-value` | Value when field is checked in a form. |
| `unchecked-value` | Value when field is unchecked in form. |

# Field Types Can be as Follows



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

## Example of a Simple Form Metadata File



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

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

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

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

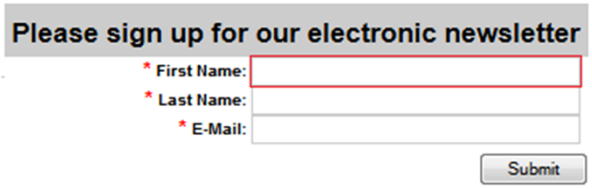
In this example, the fields `fname`, `lname` and email store the information needed to send a newsletter to a non-registered user. The fields are:

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

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

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

**Note:** Although it is not a requirement, it is a best practice to use lower-case letters when naming your xml forms.




`session.forms.`*<form metadata file>.<formid>*

OR

`pdict.CurrentForms.`*<form metadata file>.<formid>*

```
<isinputfield formfield=
      "${session.forms.newsletter.fname}"
      type="input"/>
```

Define an ISML template with the same tags needed for a valid HTML form: `<form>...</form>`

You can implement your own form action by specifying a controller URL, but that would circumvent the Forms framework.

`<form action="${URLUtils.continueURL()}" method="post">`The method `dw.web.URLUtils.continueURL()` ensures that the form gets submitted back to the controller that displayed the form template.

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

The custom tag uses the `fname` `formid` from the metadata file and builds an HTML label using the `forms.properties` file to pull the text for the `forms.contactus.firstname.label` key. It also creates an HTML input field to the right of the label with the necessary client-side JavaScript to enforce required fields, as shown.

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

The final requirement in the ISML template is to implement the button that matches the action in the form metadata. For this, 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 controller determines which form action to process.

## Lesson 8.3: Pipeline / JavaScript Controller Elements

A controller that uses the Digital Forms framework has a distinctive pattern that uses these elements:

- `ClearFormElement` pipelet or `<metadata>.clearFormElement()` function to clear an existing form object in the `pdict` using a specified form metadata file.
- `InvalidateFormElement` pipelet or `<metadata>.<Element>.invalidateFormElement()` function invalidates the specified `FormElement`.

Use the ISML object to display ISML form, and to perform server-side validation.

To create a form using the form framework:

- Create an xml metadata file that will hold your form data.
- Create an ISML template that will display a form to a visitor.
- Create a controller to display and process the form.

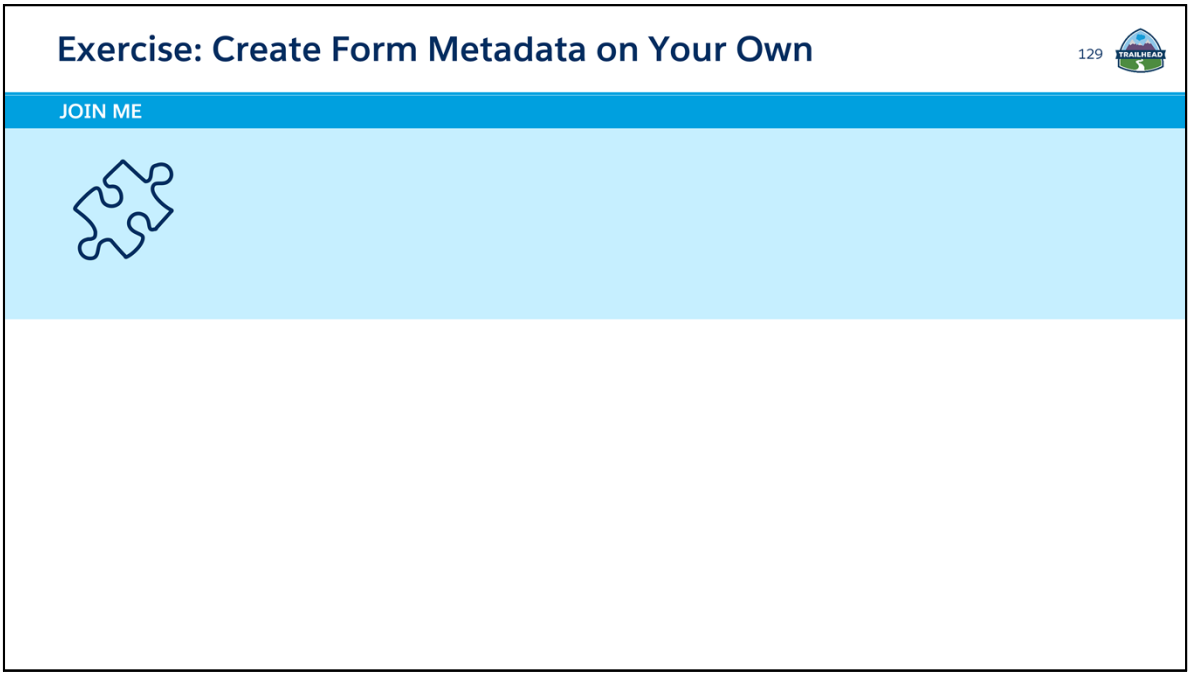
## Exercise: Use the Forms Framework



JOIN ME

**Note**: Perform steps 1- 4, if you did not complete them from the previous exercise.

1. Define form metadata to store newsletter subscription data.
2. Study the fields and action in the `newsletter.xml` file from the `solutions` cartridge.
3. Save `newsletter.xml` into your cartridge at exactly **the same location** as in solutions.
4. Create `templates/resources/forms.properties` and externalize the keys in `newsletter.xml`. For example: `forms.contactus.firstname.label=Enter your first name please`.
5. Define a template to capture form data.
6. Study the use of the `<isinputfield>` custom tag in the `newslettersignup.isml` template in the `jsolutions` cartridge.
7. Study the use of the `URLUtils.httpsContinue()` method. What will this method accomplish in the context of the form action?
8. Save `newslettersignup.isml` from the solutions cartridge into your cartridge under similar directory structure (`templates/default/newsletter`).
9. Create a template to display the form values submitted.
10. Save `newslettersuccess.isml` from the solutions cartridge into your cartridge: this displays a "Thank you <fname> <lname> for signing up" under similar directory structure.
11. Create resource bundle `templates/resources/locale.properties` and externalize the keys used in `newslettersignup.isml` and `newslettersuccess.isml`.
    For example:
    `global.newslettersignup=Please sign up for our newsletter`
    `global.newsletterthanks=Thank you **{0} {1}** for signing up!`
12. Create a JavaScript controller named JNewsletter.js. Copy code from JNewsletter_exercise.js into it. Follow the instructions in the comments to complete the code.
13. Adjust the links in `newslettersuccess.isml` to point to `JNewsletter` controller.
14. Execute the code.




To add rows, right-click in any cell and select **Insert Rows Above (or below)**.

In this exercise, you will capture customer interests related to categories and products on the site. 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 modify this form metadata file that captures marketing and personal information from a registered visitor. The modification must use the following:

| Formid | Label | Data type | Binding | Externalized Strings |
|---|---|---|---|---|
| interestApparel | forms.interestedin Apparel | boolean | custom.interestApparel | Are you interested in Apparel ? |
| interestElectronics | forms.interestedin Electronics | boolean | custom.interestElectronics | Are you interested in Electronics ? |
| newsletter | forms.interestedin Newsletter | boolean | custom.newsletter | Are you interested in Newsletter ? |

None of the choices are mandatory.

2. Add an `apply` action that does not require validation. You will not use this metadata until a later exercise.

# Knowledge Check 8

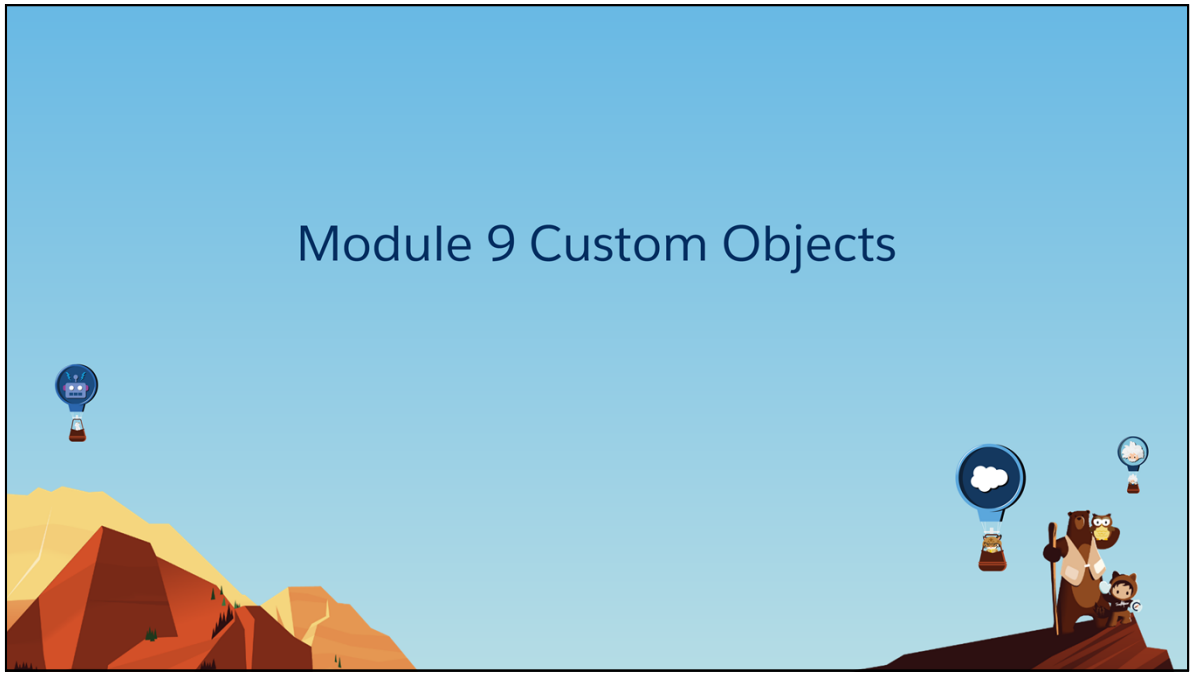


**True or false:**

1. The `<isinputfield>` is a custom tag used to populate form field attributes.
2. The following piece of code can display and submit a form

```
ISML.renderTemplate('registerForBenefits', {
        displayForm : session.forms,
        Subscription : co
});
```

# Module 9 Custom Objects



## Module 9: Custom Objects



**Objectives:**

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

**Lessons:**

9.1 Using Commerce Cloud B2C Commerce Script to Create Custom Object Instances

9.2 Custom Logging

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

Custom Objects extend the Commerce Cloud B2C Commerce data model. They are basically a new table in the database where you specify the primary key and storage attributes (columns) that suit your business needs.

**Note:** Always first consider if you can use a B2C Commerce System object (Product, Catalog, etc.) instead of creating a custom object. Although you can create custom objects, they are best used to store static data (like configuration parameters), not for uncontrolled amounts of data (like analytics). Custom objects searches can be slow if the data is large. You should consider data growth and cleanup in your Custom Objects. Commerce Cloud B2C Commerce Governance has quotas around custom object API usage and data size which will be enforced in the future.

## Custom Object Creation

133

Custom Object Definitions
- Business Manager

Custom Object Instantiation (Creation)
- Business Manager
- Programmatically

You create custom objects at the organization level; therefore, they are available for use in all storefronts within the organization. You use two Business Manager modules to define and manage your custom objects:
- Custom Object Definitions: facilitates naming, primary key and column specification. It is located in **Administration > Site Development.**
- Custom Object Editor: facilitates instance creation and editing. It is located in **Site - <site> > Custom Objects > Custom Object Editor**.

When defining the Custom Object, specify the storage scope of the instances: site or organization.
- Organization Custom Objects can be used by any site.
- Site Custom Objects are created by one site and cannot be read by another.

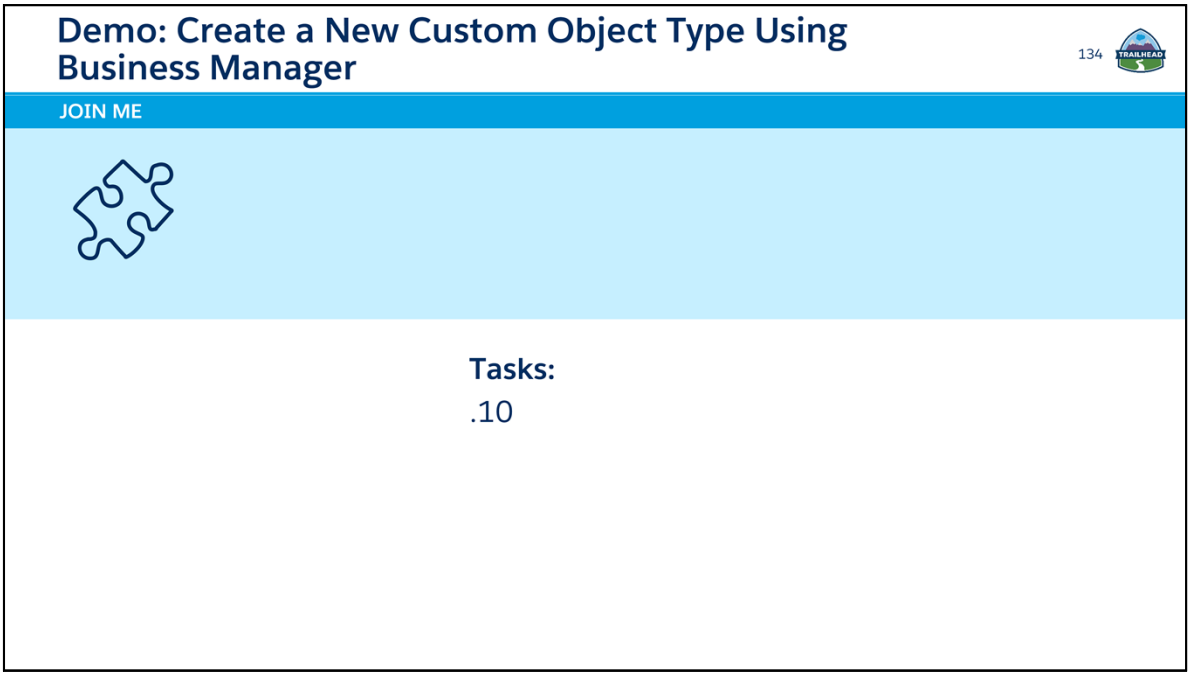
The Custom Object type itself is always available to the entire organization. Also, you can specify if you want Custom Object instances to be replicable. This means you can copy them from Staging to Production during the replication process.

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

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

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

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




1. Log into Business Manager.
2. Select **Administration > Site Development > Custom Object Definitions.**
3. Click **New** to create a new Custom Object type.
4. Fill in the required fields for the Custom Object type:
   - **ID**: the unique ID of the object type. It cannot contain spaces.
   - **Key Attribute:** This is the unique key for the custom object type.
   - **Data Replication**: Specify whether the custom object type data will be replicable to other instances.
   - **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 become available.
6. Click the Attribute Definitions tab. Notice the default values created with your Custom Object type. These values cannot be changed once they are created.
7. To create the attributes (values you wish to capture in the table), click **New**.
8. In the ID field, specify a unique name. In the Value Type drop-down, select the type of data being entered for the attribute.
9. Click **Apply**.
10. Click the **Back** button to add another attribute.
11. When you are finished adding attribute definitions, create an Attribute Group. Click the **Attribute Grouping** tab.
12. In the **ID** field, enter a name for your grouping. In the **Name** field, enter a name. Click **Add**.
13. Add field attributes to the group. Click the **Edit** link.
14. To the right of the ID field, click the ellipses to select field attributes.
15. Select the attributes you wish to add from the list by clicking in the checkbox next to each one. Click **Select**.

You can now view, add, and edit new instances of the custom object type you just created in the Custom Object Editor section.

## Demo: Create a New Custom Object Instance Manually Using Business Manager



JOIN ME

**Tasks:**

.10

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

## Exercise: Create a Custom Object Definition



JOIN ME

**Tasks:**

.10

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

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

## Lesson 9.1: Using B2C Commerce Script to Custom Object Instances



The B2C Commerce Server Script API provides the following classes in the **dw.object** package, among others:

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

All “Objects” represent persistent data.

This is the inheritance tree for the CustomObject type:
`Object > dw.object.PersistentObject > dw.object.ExtensibleObject > dw.object.CustomObject (or dw.object.SystemObject)`

Custom Objects persist in the database. An administrator or developer can add custom attributes added to them in Business Manager. Many commonly used classes such as `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 use of the `CustomObjectMgr` class enables creation of an instance of a Custom Objects by providing the Custom Object type and the primary key:

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

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

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

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

## Database Transaction Handling



Two approaches to database transaction handling:
- Implicit
- Explicit

There are two approaches to database transaction handling in Commerce Cloud B2C Commerce:
- Implicit – The script automatically starts the transaction and then commits or rolls back if Commerce Cloud B2C Commerce determines it to be appropriate.
- Explicit – the transaction is controlled in the script. The developer explicitly indicates in the code when the transaction should begin, rollback, or commit.

# Implicit Transaction Control



```
var Transaction = require('dw/system/Transaction');
Transaction.wrap(function()
{
      //Your code
});
```

For example:

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

# Explicit Transaction Control



```
var Transaction =
      require('dw/system/Transaction');

Transaction.begin();
      your code

try
{
      more code
      Transaction.commit();
}
catch (ex)
{
      Transaction.rollback();
}
```

## Create a Custom Object via Script API



To create a custom object programmatically, follow these steps:

1. Create a *custom object* ***type*** in Business Manager before creating a *custom* ***object*** programmatically.
2. Create a script that uses the dw.object.CustomObjectMgr class to create a custom object.

This can be B2C Commerce Server script, or CommonJS script.

## Exercise: Create a Custom Object Using Implicit Transaction Handling



JOIN ME

Create a JavaScript controller named `JNewsletterV2.js`. Copy code from `JNewsletterV2_exercise.js` into it. Follow these instructions to complete the code:

1. Copy `MyModel.js` from the `JSolutions` cartridge to your `training` cartridge (in similar location) and study how it is creating the object.
2. Copy `newsletter/newslettererrorV2.isml` and `newslettererrorV2` from `JSolutions` cartridge into your cartridge in the similar location. Adjust the links in `newslettererrorV2.isml` to point to `JNewsletterV2.js` controller.
3. Execute the controller and see if objects are being created.

Multiple severities and categories of logging (defined by Apache log4j open source project):

- ✓ Debug
- ✓ Info
- ✓ Warn
- ✓ Error
- ✓ Fatal

```
var logger = Logger.getLogger("logFilePrefix","category" );
logger.debug("Input params received in pipelet
              firstName: {0}\n lastName: {1}\n email: {2}",
              args.firstName, args.lastName, args.email);
try
{
 … do something…
}
catch (e)
{
  logger.warn(“error description: {0}", e.causeMessage );
}
```

dw.system.Logger.getLogger() factory method

Commerce Cloud B2C Commerce 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 enable the developer to capture debug messages at different levels of granularity. The severity levels are:
Debug < Info < Warn < Error < Fatal

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

The developer can define as many levels of categories and subcategories as needed. Commerce Cloud B2C Commerce does not impose a certain categorization; the developer determines the organization. For example:

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

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

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

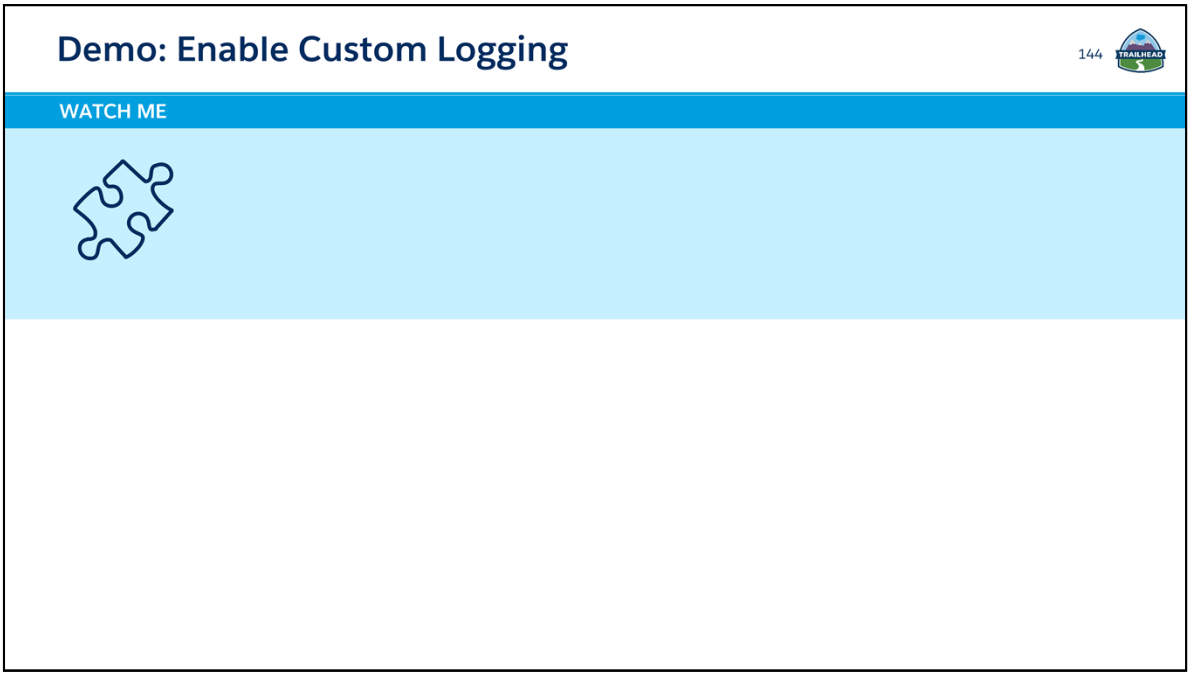
- Warn, Error and Fatal messages are logged for "product" and all its sub-categories.
- Debug and Info are logged for "product.import" and all its sub-categories.

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

```
var logger = Logger.getLogger("logFilePrefix","category" );
logger.debug("Input params received in pipelet
              firstName: {0}\n lastName: {1}\n email: {2}",
              args.firstName, args.lastName, args.email);
try
{  … do something…  }
catch (e)
{  logger.warn(“error description: {0}", e.causeMessage );  }
```

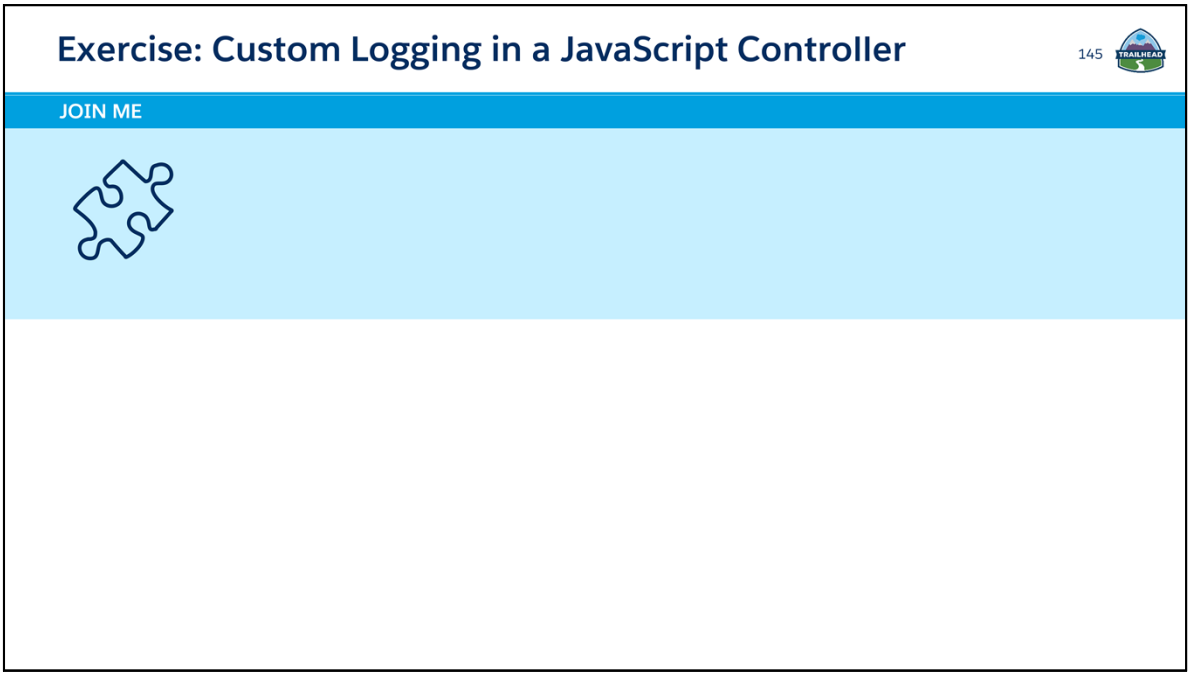
Use the Logger object to write a message for a specific severity level: `Logger.error(String msg)`.

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.




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

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




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

1. Modify your latest version of `JNewsletter` to write to the custom log files.
   a. Use require to get Logger from `dw.system` package and instantiate a Logger class.
   b. Add code to write useful information to the logs. Include a debug level message for all submissions and an error level message in the `catch` block. Use the two string signature with these arguments: (`"NewsLogs"`,`"newsletter"`).
   c. If you need help, use the sample code provided after this exercise.
2. Enable logging for debug messages.
   a. In Business Manager, select **Administration > Operations > Custom Log Settings**.
   b. In the Log Category field, enter `newsletter` and set Log Level to `DEBUG`.
   c. Click **Add**.
3. Select Debug in Log Files. (Do not deselect any of the options already checked.)
4. Click **Save**.
5. Test your controller with a new email address and then a duplicate.
6. View the latest custom log files.
   a. Go to **Administration > Site Development** > **Development Setup > Log Files**.
   b. Verify the messages appear with the most recent timestamp. The name of the file should start with `custom-NewsLogs`
7. Verify that the messages also appear on the request log.

**Code Sample**

```
var logger = require('dw/system/Logger').getLogger("NewsLogs","newsletter");
logger.debug("Input params firstName: {0} lastName: {1} email: {2}",
    newsletterForm.fname.value, newsletterForm.lname.value,
    newsletterForm.email.value);
```

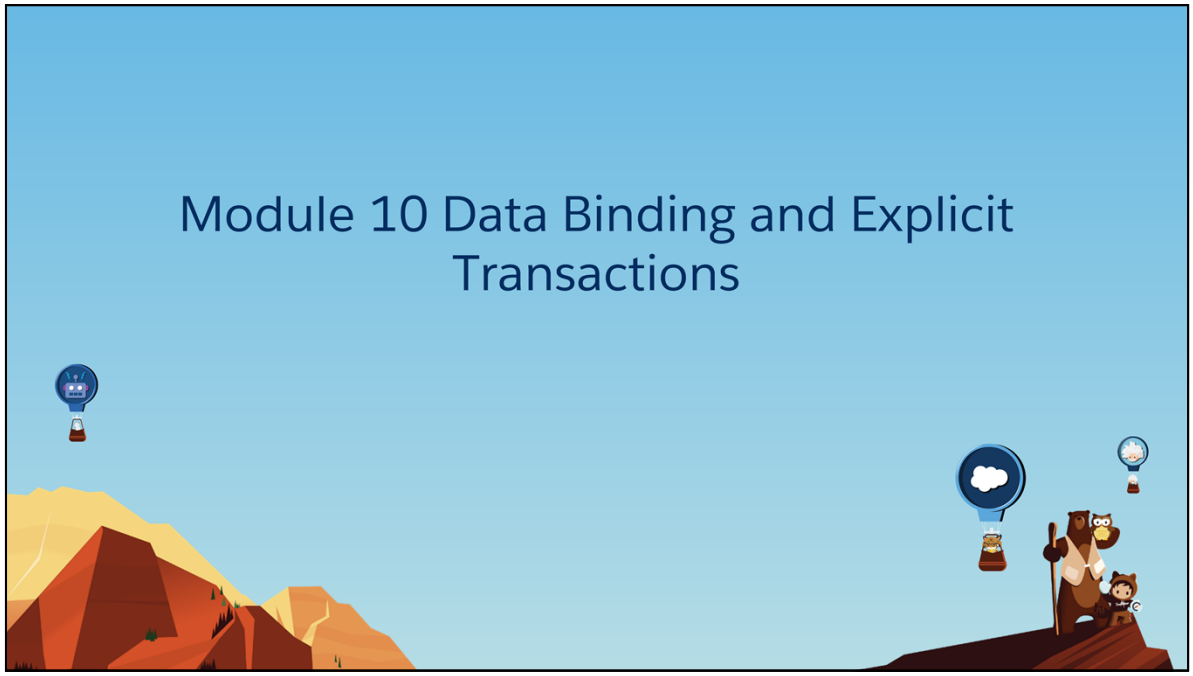
# Knowledge Check 9



**True or false:**

1. Custom objects are the only way to store custom data in Commerce Cloud B2C Commerce.
2. The “custom” keyword is required to access custom attributes of an object.
3. A custom object needs a primary key.
4. Custom object instances can only be created in Business Manager.

# Module 10 Data Binding and Explicit Transactions



# Module 10: Data Binding and Explicit Transactions



**Objectives:**

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

**Lessons:**

10.1 Data Binding with Forms and Objects

## Lesson 10.1: Data Binding with Forms and Objects



```
<?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>
```

| | ID | Name |
|---|---|---|
| 🔒 | **UUID** | UUID |
| 🔒 | **creationDate** | Creation Date |
| 🔑 | **email** | |
| | **firstName** | First Name |
| 🔒 | **lastModified** | Last Modified |
| | **lastName** | Last Name |

The Commerce Cloud B2C Commerce 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 B2C Commerce API. The reverse mechanism is also supported: pre-populating a form object with data from a persistent object.

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

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

When the information is stored in a custom object in the code, you can manage that transaction explicitly in the code using `dw.system.Transaction`. The following methods can then handle the transaction.

```
Transaction.begin(..)
Transaction.commit(..)
Transaction.rollback(..)
```

Transactions can also be implicit. This means that the transactions will not have to be explicitly begun or committed. They will be handled by the Commerce Cloud B2C Commerce. The following is an example code for implicit transaction.

```
Transaction.wrap(function(){
// work with business objects here
});
```

## Exercise: Create a Custom Attributes on a System Object



JOIN ME

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

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

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

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

Note: Follow the instruction in the comment to make the line fully executable and working (use Script API as the guide if needed).

6. Use the `copyTo` method (use the quickcard section “Handling Forms”) to store `newsletterForm` to the object created above.
7. Adjust `newslettersuccessV2.isml` to point to the controller JNewsletterV3.js.
8. Execute the JavaScript controller.
9. In Business Manager, select **Merchant Tools > Custom Objects > Custom Object Editor**. Determine if the object was created.

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



JOIN ME

Create a new `JEditPreferences.js` controller to pre-fill the form with the logged-in customer preferences. Once the customer changes his/her preferences, save the data to the database.
**Note**: Perform Step 1-3 only if you have not so in the previous exercise.

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

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

8. Edit the Form Metadata to add bindings and externalization of strings.
9. Open the `preferences.xml` form metadata file.
10. Externalize the keys in preferences.xml in templates/resources/forms.properties file for example: `forms.preferences.apparel=Are you interested in Apparel ?`

Likewise externalize the other two keys.

11. 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"/>
```

12. Copy the `editpreferences.isml` from the `customerpreferences` cartridge to your own cartridge under similar directory structure (`templates/default/account/user`). Make sure the formfields are matching the formids of the `preferences.xml` metadata file.

13. Create a JavaScript controller named `JEditPreferences.js`. Copy code from `JEditPreferences_exercise.js` into it. Follow the instructions in the comments to complete the code.

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

# Knowledge Check 10



**True or false:**

1. To implement Explicit Transactions you use the `beginTransaction()` method in the controller.
2. You can rollback a transaction in the code using implicit transaction.

# Module 11 Site Maintenance

## Module 11: Site Maintenance

154

**Objectives:**

- Use the “Pipeline” Profiler and implement page caching.
- Use the JavaScript Controller Profiler.

**Lessons:**

11.1 Site and Page Caching

11.2 Site Performance

Commerce Cloud B2C Commerce provides tools that you can use to improve site performance as well as replicate code and data.

**Note:** In the Business Manager, the JavaScript Controller Profiler is referred to as the Pipeline Profiler.

**Note:** The *Developing for Commerce Cloud B2C Commerce II* course provides additional information on this topic.

## Lesson 11.1: Site and Page Caching



Commerce Cloud B2C Commerce controls caching on a per page basis, via the ISML template for the page. Set caching on a page using the `<iscache>` tag:

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

Commerce Cloud B2C Commerce follows these rules when using the tag:

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

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 slows down the entire site for all users (including job processing) and not just for the requested pages.

## <iscache> Parameters

- `status = "off|on"` *deprecated*
- `type = "relative | daily"`
- `hour = integer`
- `minute = integer`
- `varyby="price_promotion"`

## Use the <iscache> to set the following parameters



| Parameter | Description |
|---|---|
| `type = "relative \| daily"` | Relative enables you to specify a certain period of time, in minutes and hours, after which the page will be deleted from the cache. Daily enables you to specify an exact time when the page will be deleted from the cache. |
| `hour = integer` | Indicates either the caching duration or the time of day. If the `type` attribute is set to `daily`, the `hour` value must be an integer, ranging from 0 to 23. If `type` is set to `relative`, all integer values greater than 0 are valid (the default value is 0, meaning either the page is never cleared from the cache or only the `minute` attribute is relevant). |
| `minute = integer` | Indicates either the caching duration or the time of day. If the `type` attribute is set to `daily`, the `minute` value must be an integer ranging from 0 to 59. If `type` is set to `relative`, all integer values greater than 0 are valid (the default value is 0, meaning either the page is never cleared from the cache or only the `hour` attribute is relevant). |
| `varyby= "price_promotion"` | Enables you to mark a page as personalized: this does not mean that the page is unique for a person but rather that different versions of the same page showing different prices, promotions, sorting rules or AB test segments will be cached by Commerce Cloud B2C Commercel. For example, this parameter is necessary for product pages since a customer belonging to a customer group might get special promotions that other customer groups don't get. While the ISML template is the same, the generated pages vary, and therefore caching every version of the page benefits performance. For performance reasons, a page should only be **marked with the** `varyby` **property if the page is really personalized**; otherwise, the performance can unnecessarily degrade. |

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

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

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

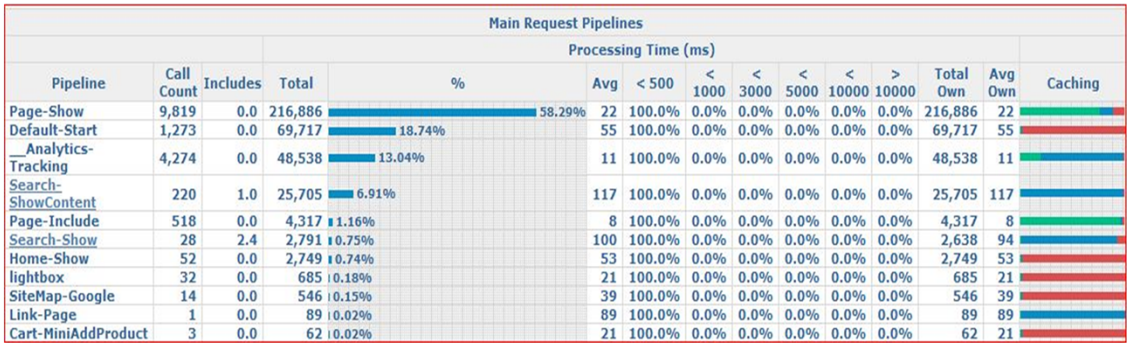
Portions of pages can be cached separately. You can assemble a page from snippets with different caching attributes using remote includes. Each part:

- Must be a result of a pipeline request to the application server.
- Is included using the `<isinclude url="">` or the `<iscomponent pipeline=….>` syntax.
- Can have different cache times or no caching at all.

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

## Study Page Analytics to Determine Caching Problems





Main Request Pipelines

Processing Time (ms)

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

To access B2C Commerce caching metrics, select **Site > Analytics > Technical Reports**. Shown is the **Pipeline Performance** report.

These types of analytics are only collection on Production instances, not Sandboxes. In this example, it reveals that the `Home-Show` controller (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 B2C Commerce customers, the two critical metrics to focus on from a performance perspective are the average response times of `Search-Show` and `Product-Show` controllers. These controllers are used across all customers and are the main components of most pages on B2C Commerce installations.

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

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

## Page-Level Caching



```
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
```

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

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

# Page Caching Example



```
var ISML = require('dw/template/ISML');

let Calendar = require('dw/util/Calendar');
let cal = new Calendar();

cal.add(Calendar.MINUTE, 30);
response.setExpires(cal.getTime());

ISML.renderTemplate('cachedpage');
```

## Exercise: Page-Level Caching



JOIN ME

1. Create an ISML template named `cachedpage.isml` that has caching enabled for 30 minutes:
   `<iscache type="relative" minute="30" />`
2. Add a `Date` object to the page that prints the current time:
   `<isprint value="${new Date()}" style="DATE_TIME" />`
3. Create a new pipeline named `Caching` or a javaScript controller named `JCaching` to display the above template.
4. Test the template in your SiteGenesis storefront. Refresh your page. Does the time change on refresh?
5. Enable caching on your SiteGenesis site. Retest the template. You may need to wait a minute before you see the page has been cached.

## Partial Page Caching Example



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

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

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

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

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

## Exercise: Partial Page Caching



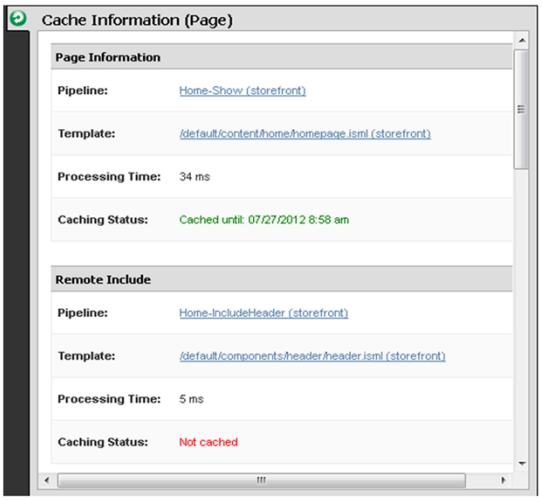
JOIN ME

1. In the template `cachedPage.isml`, add a remote include call to the `Product-IncludeLastVisited` (pipeline or controller) `<iscomponent pipeline="Product-IncludeLastVisited" />`
2. Invalidate the cache in Business Manager.
3. Go to another tab and visit a few products (3 at most).
4. Refresh the `Caching-Start` pipeline or JCaching-Start controller
5. Visit more products on the other browser.

   Result: the time remains unchanged while the last visited products change every time a new product is visited.
6. Once you have finished this exercise, do not forget to turn your caching off again in Business Manager (Administration > Sites > Manage Sites > SiteGenesis > Cache)

# Use 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.

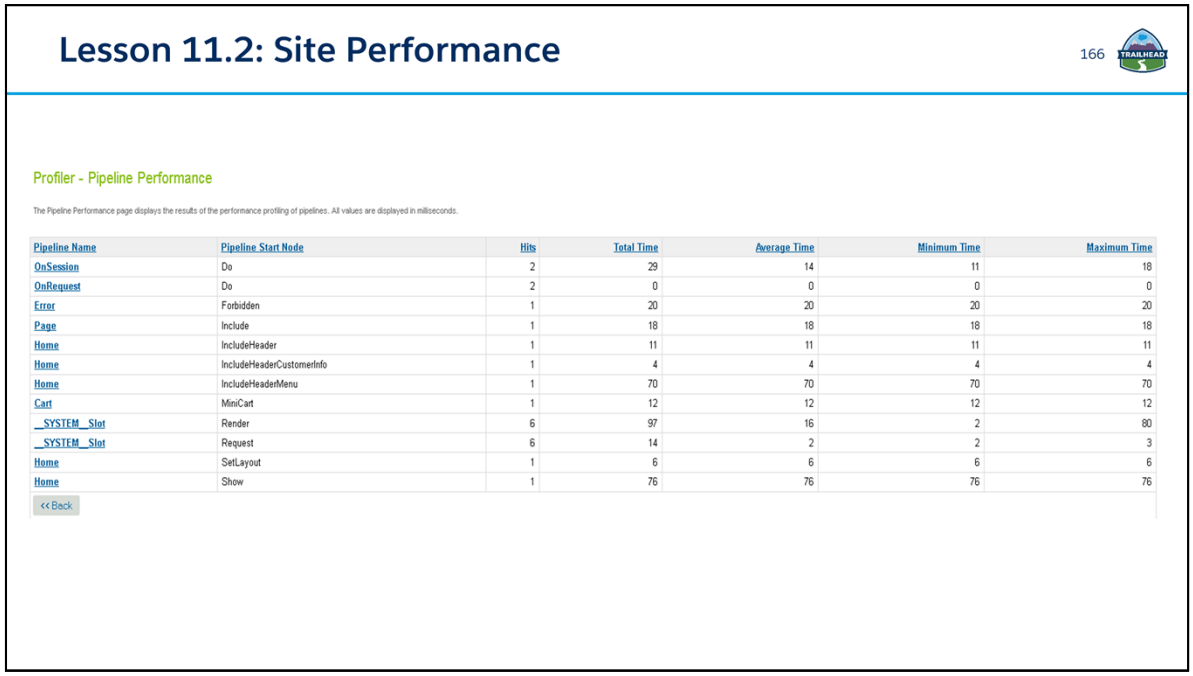
The page now shows special icons that you can click to reveal how the whole page and its remote includes are cached.

## Exercise: Use the Cache Information Tool



JOIN ME

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?



Profiler - Pipeline Performance

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

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

<< Back

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

To track the performance of a pipeline using the Pipeline Profiler:

1. In Business Manager, select **Administration > Operations > Pipeline Profiler**.
2. Reset previously collected statistics and turn on the Pipeline Profiler.
3. Browse specific pipeline in storefront.
4. Return to profiler and analyze the collected data.
5. Click the link for that site where you want to capture data:
    a. The pipeline profiler displays a high-level view of response times per script or pipeline, such as hits, total time for a page to be generated, average time, etc.
    b. Look for scripts with high average run times and high hits. These are the first areas to focus on performance improve.
    c. To view more detailed data for a specific script, click the script name.
6. Test the script or a different one again.
7. While the pipeline profiler runs you have access also to captured script data.
8. Turn off the profiler and analyze the results.
9. If you make modifications to the script, retest to verify if performance has improved.

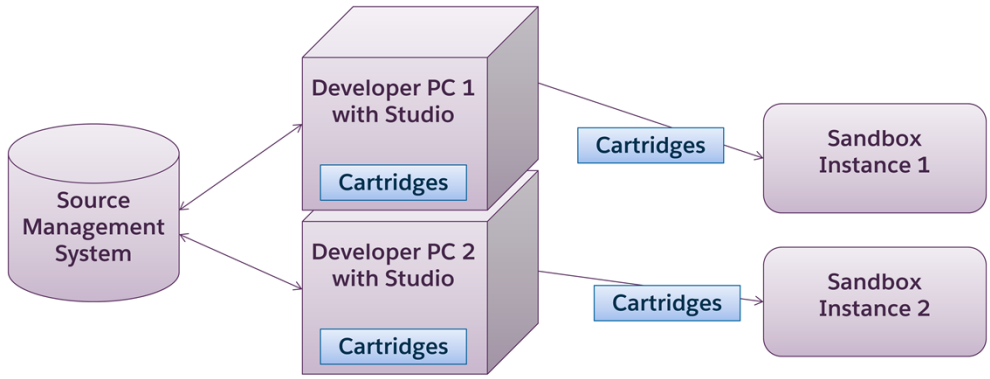
## Lesson 11.3: Code Replication



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

## Code Replication Overview (1)





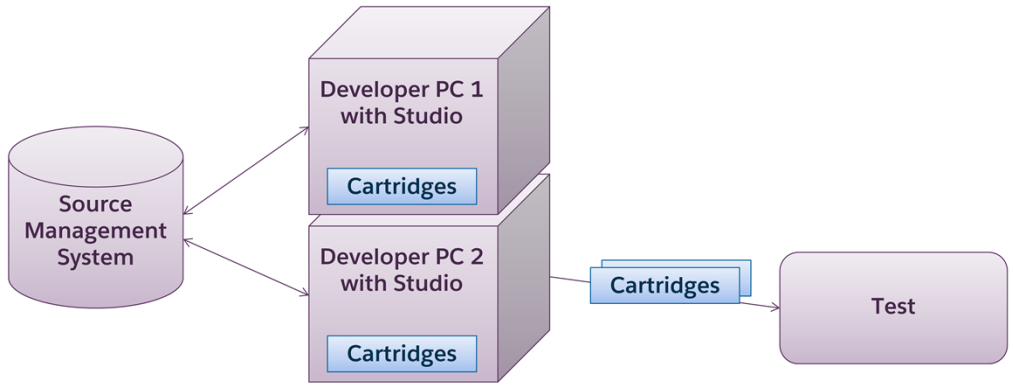

In a typical development environment, a source management system is used for code version control. Each developer uses their own sandbox for development, while checking in their code to a source management system.

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

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

## Code Replication Overview (2)



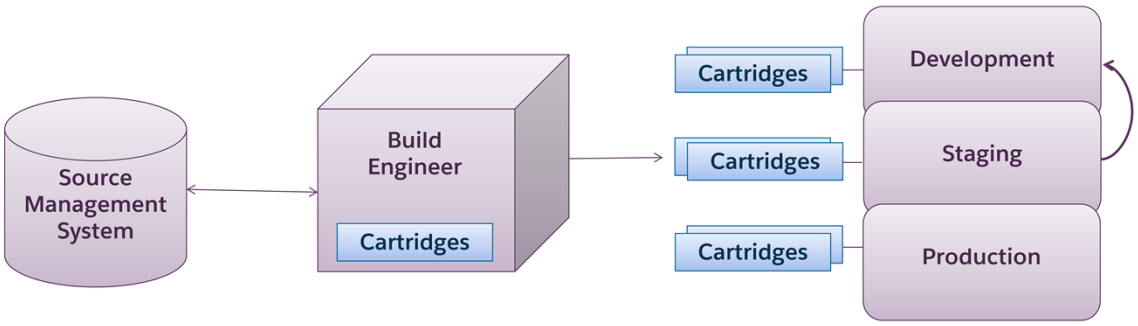



Next, the developer *uploads* custom cartridges with UX Studio or WebDAV client using 2-factor authentication and tests the storefront in Staging. A rollback to a previous version is available.

For major code changes, it is recommended to use a sandbox for testing. To test in a sandbox, export the site data to the global directory from staging and import it into your sandbox using the Site Import/Export module in Business Manager.

## Code Replication Overview (3)



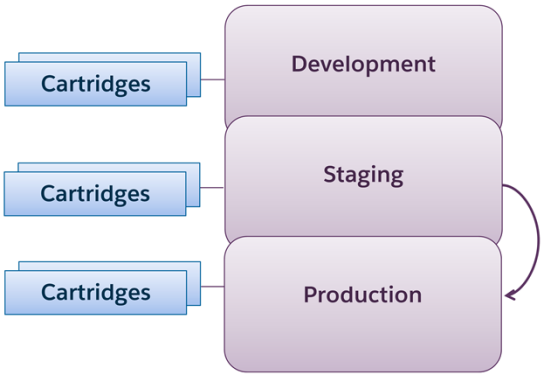



When you need to test code metadata (site preferences, new attributes, etc.), the build engineer replicates from Staging to Development.

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

## Code Replication Overview (4)

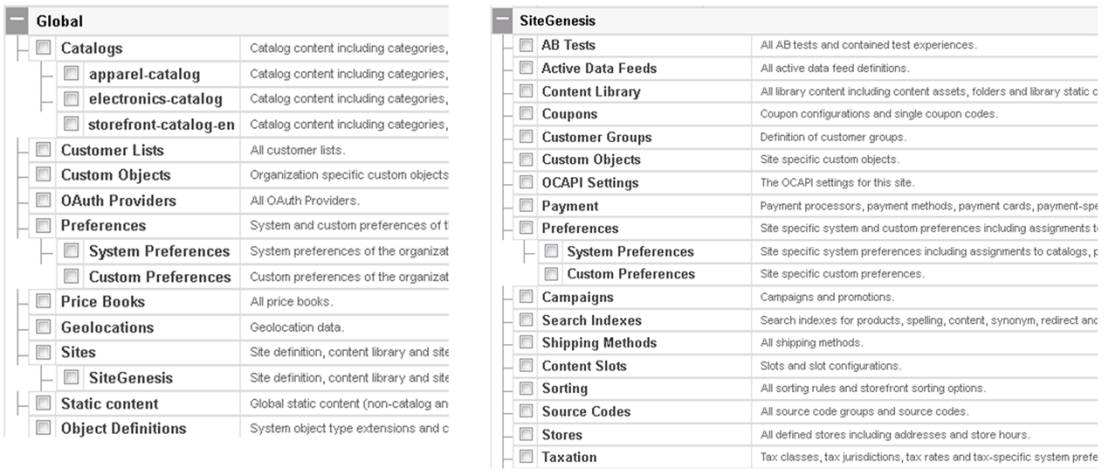





The last step in code replication is *moving* code from Staging to Production using Business Manager.

To replicate code from Staging to Development or Staging to Production:

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




Data replication promotes 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

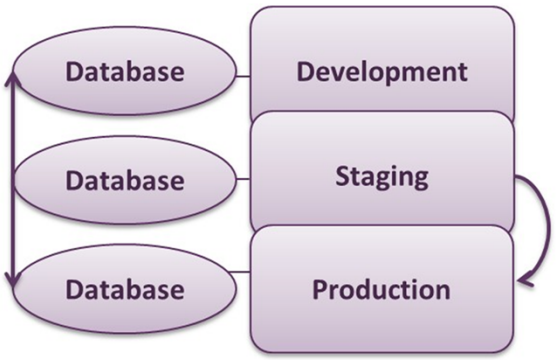
Data Replication has two phases:
- Transfer – Long running processes where data is copied from Staging into shadow tables and folders on Production. No changes are shown in storefront.
- Publishing – Very fast process. Changes in shadow tables and folders become active, the page cache is purged, and the new version is shown in storefront.

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

To view the progress of a replication, monitor the staging logs on the staging and production instance.

## Data Replication from Staging






Like code replication, you set up data replication in Business Manager. The process is similar, except you select the data that you want to replicate.
Just as code replication can only occur between Staging and Development or Staging and Production, data replication is a one-way process from Staging to the other primary instances.

To replicate data from Staging to Development or Staging to Production:

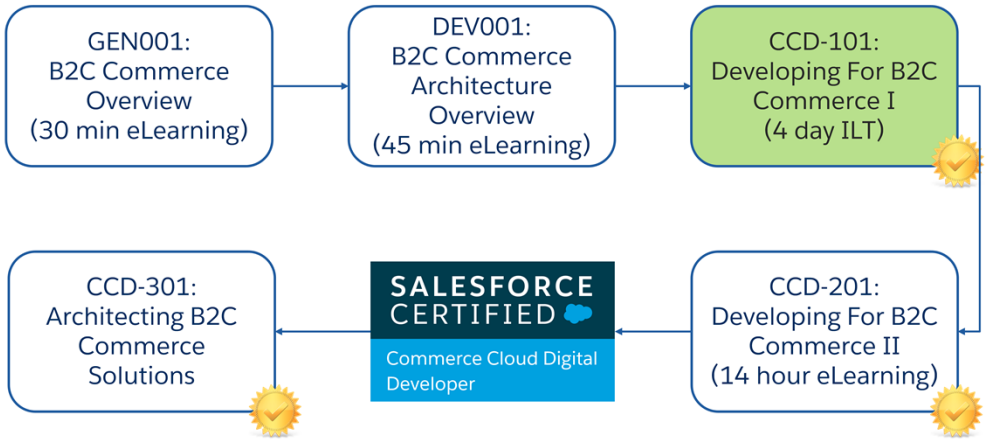
1. Log into the Staging Business Manager with an account that has code replication permissions.
2. Select **Administration > Replication > Data Replication.**
3. Click **New** to create a new data replication process.
4. Specify the target for the data replication process: Development or Production.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

# Knowledge Check 11



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

# Commerce Cloud Developer Track




## Thanks for Attending!

Your opinion matters, and we want to hear from you. Navigate to our Class Survey to give us your feedback.

## What's Next?

**LEARN ALL THE SKILLS YOU NEED**

Build on your skills with self-paced learning or another expert-led class.
https://sfdc.co/learnsalesforce

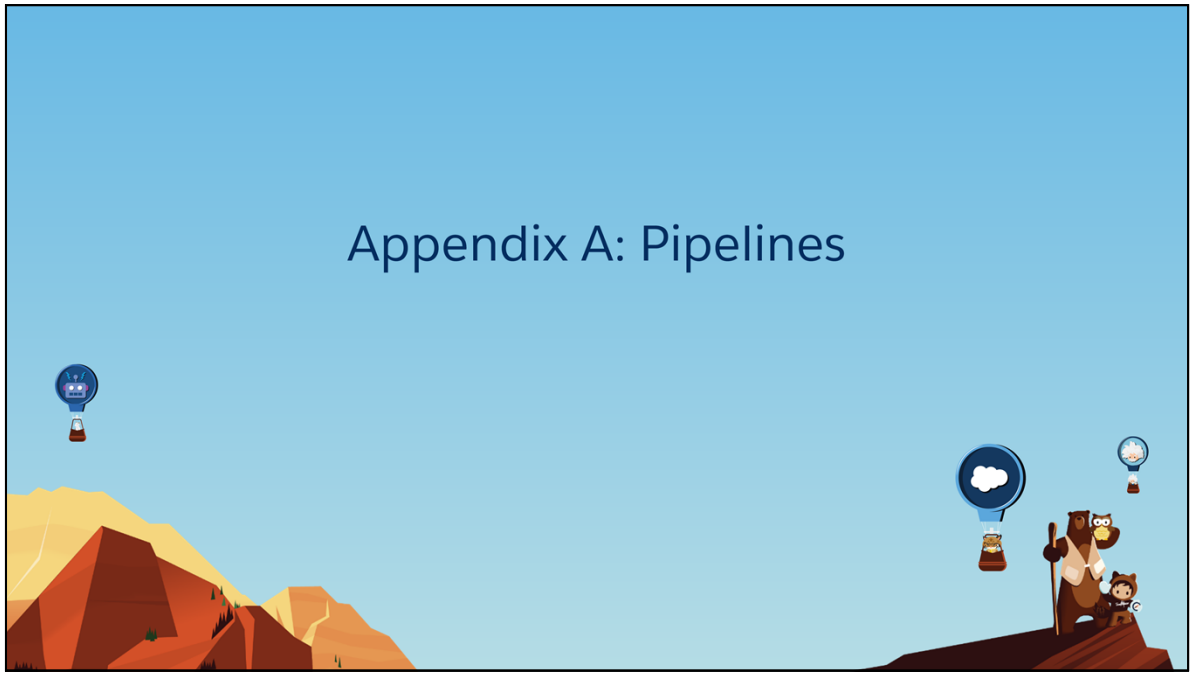
**EARN SKILL-BASED CREDENTIALS**

Get rewarded for the skills you learn and get industry-wide recognition for your expertise.
https://trailhead.salesforce.com/en/super_badges
http://certification.salesforce.com/

**CONNECT WITH FELLOW TRAILBLAZERS**

Follow us in the Trailblazer Community, and on social.
Community: https://sfdc.co/TrailheadCommunity
Twitter: @Trailhead
Facebook: /SalesforceTrailhead

# Appendix A: Pipelines



## Appendix A: Pipelines

178

**Objectives:**

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

**Lessons:**

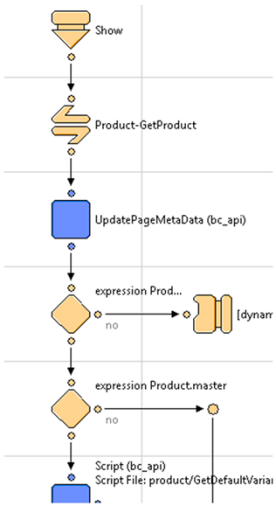
Lesson A.1: Pipeline Overview
Lesson A.2: Creating a Pipeline
Lesson A.3: Call Nodes and End Nodes
Lesson A.4: The Pipeline Dictionary
Lesson A.5: Troubleshooting with the Pipeline Debugger
Lesson A.6: Pipelets

Note: In general, JavaScript controllers replace pipelines throughout Commerce Cloud B2C Commerce. If you are creating a new site, Salesforce strongly recommends the use of JavaScript controllers. However, if you are working with an existing site that has pipelines, it is important to understand how pipelines work so that you can maintain and extend them if needed.

**Cartridge Folder Structure**
Cartridges can contain either controllers and pipelines or controllers alone. If you have controllers and pipelines in the same cartridge, and they have the same name, the platform uses the controller and not the pipeline. Even if they are in different cartridges and have the same name, the platform uses the controller in the path and not the pipeline.

## Lesson A.1: Pipeline Overview






A pipeline is a logical model of a particular business process, similar to a flow chart. UX Studio provides a visual representation of the process within the Eclipse IDE. This example shows the Product-Show pipeline that renders the product detail page on the SiteGenesis site.
Pipelines are stored in XML files in the file system, both locally and on the server. You define and store pipelines within the context of a cartridge.
When the storefront application references a pipeline in a cartridge, it searches for the pipeline in the cartridge path and uses the first one it finds. When a pipeline with the same name exists on two cartridges, the first one found in the path is used. Therefore, use unique pipeline names to ensure that the framework locates the correct pipeline.

# Pipeline Elements (1)



There are many pipeline elements available.

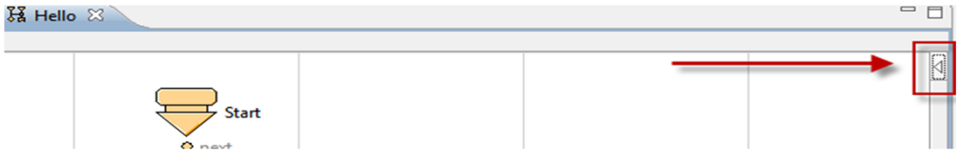
| Element | Icon | Description |
|---|---|---|
| Start Node |  | Begins the logical branch of a pipeline. |
| Interaction Node |  | Use when a request requires a page as a response. |
| Transition Node |  | Define a path along a pipeline between pipeline nodes. |
| Call Node |  | Invoke a specified sub-pipeline. After the sub-pipeline execution, the workflow returns to the calling pipeline. |
| End Node |  | Terminate a sub-pipeline and return to the calling pipeline. |
| Jump Node |  | Use when the pipeline forwards the request to another pipeline. |
| Join Node |  | Provide a convergence point for multiple branches in workflow. |

# Pipeline Elements (2)



| Element | Icon | Description |
|---|---|---|
| Interaction Continue Node |  | Process a template based on user action via a browser. |
| Script Node |  | Execute Digital scripts. |
| Eval Node |  | Evaluate an expression. |
| Assign Node |  | Assign values to new or existing Pipeline Dictionary entries, using up to 10 configured pairs of dictionary-input and dictionary-output values. |
| Stop Node |  | Terminate a sub-pipeline and calling pipelines; stops execution immediately. Use in pipelines that execute a batch job. |
| Loop Node |  | Use to loop through an iterator. |
| Pipelet Placeholder |  | Placeholder for a script node. |
| Decision Node |  | Evaluate a condition and navigate to a different branch in the pipeline. |

## Lesson A.2: Creating a Pipeline




A new pipeline needs at least one Start node and one Interaction Node. In UX Studio, when creating a new pipeline, access the pipeline palette from the Pipeline Editor.
Note: If the palette is not visible, click the button on the upper-right corner of the editor.

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

**Interaction Node**
This node specifies the template to display in the browser. If the Dynamic Template is:
- 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.
- 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. To create a transition between two nodes click and drag your mouse between two nodes in a pipeline.
To create a simple pipeline using a Start node and an Interaction node:
1. From UX Studio, click **File > New > Pipelin**e. The Create Pipeline dialog displays.
2. Provide a name that describes its business purpose.
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 pointer down over the white dot at the bottom of the Start Node. Drag-and-drop your mouse pointer over to the white dot at the top of the Interaction Node. Release your mouse. A

Transition Node connects the two elements.

7. Click the Interaction Node twice (**not double-click**). This displays an 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. Click **OK.**
10. Save the pipeline: **CTRL+S**.

## Exercise: Create a Pipeline



JOIN ME

1. In UX Studio, select **File > New > Pipeline** in the **training** cartridge.
2. Name the pipeline **Hello**. Do not specify a group. Keep **View** as the pipeline type.
3. Use the palette to 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. In the pipeline’s Interaction node, specify the template name **hello.isml** in its properties.
7. Double-click the Interaction node to verify that it opens the hello.isml template.
8. Save both the pipeline and the template.

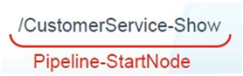
## Execute a Pipeline



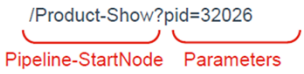
Execute pipelines from the browser via an HTTP(S) request or via Call or Jump Nodes. Note: If the pipeline Start node is set as Private it can only be called via a Call or Jump node.

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 this syntax:




To execute a public pipeline from the storefront:

1. Open your storefront in a browser window.
2. At the end of the URL add the `default/` directory, then enter your pipeline name 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
```

## Exercise: Execute a Pipeline



JOIN ME

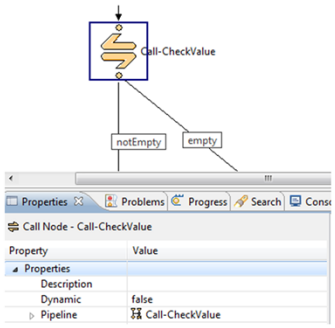
1. Test your pipeline in the storefront: `http://student1.training.dw.demandware.net/on/demandware.store/Sites-SiteGenesis-Site/default`
2. Close the Storefront Toolkit on the upper-left corner of the page to view the output.
3. Bookmark this URL to use it for future pipelines invocations during this class.

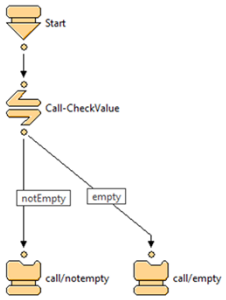
## Invoke a Pipeline

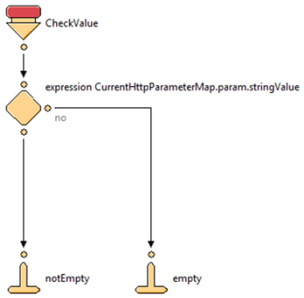


To invoke a pipeline:
1. Open your sandbox storefront.
2. Invoke a `Test-Start` pipeline (which has not been created).
3. Open the **Request Log** to view the error message.

Call nodes 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 executes, the workflow returns to the calling pipeline by means of an End node. It behaves like a function call where the function might return one or multiple values.

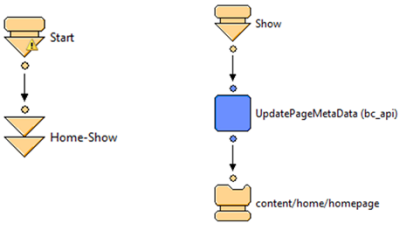
A Call node requires only a Pipeline-Start node to invoke. You can provide this information as a fixed configuration value or from a pipeline dictionary key.

**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 following example, if the CheckValue sub-pipeline returns a **notEmpty** value, then that transition is followed on the Start pipeline. Additionally, a Call node is used to invoke another pipeline. At the end of the execution of the called pipeline is a Decision node that checks whether a value called **param** has been passed in a URL string. The End nodes return control back to the Call node.

To use a Call node with a Transition node:
1. In UX Studio, open the pipeline where you want to add the Call node.
2. Select the Call node from the palette and drag it over the transition node where you want it. Note: Be sure the transition node turns red before you release the 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 dialog, select Show Pipelines from the drop-down. In the first field, enter the name of the pipeline that you want to find. The lower boxes will populate with available pipelines and Start nodes.
6. Click OK.
7. 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.

## Jump Nodes






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

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

This information can be provided either:

- As a fixed configuration value
- From a pipeline dictionary key (covered later)

An example of using Jump nodes is the `Default` pipeline. This is a special pipeline that the system calls 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.

## Lesson A.4: The Pipeline Dictionary



- The main data container.
- A hashtable with key/value pairs.
- Passed across sub-pipeline calls.

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

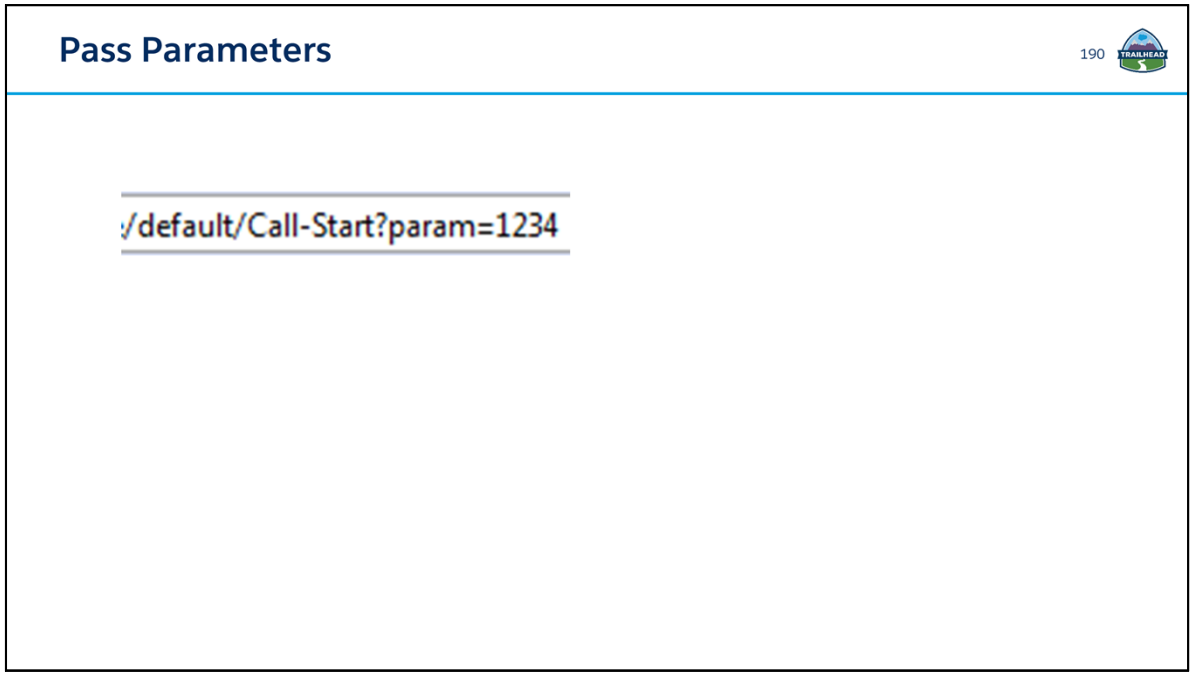
The pipeline dictionary or `pdict` is the main data container for each pipeline execution. It is created and initialized when a pipeline is invoked and remains in memory as long as the pipeline executes.
The structure of the pipeline dictionary is a hashtable with key/value pairs. The default keys in the `pdict` include:

- 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.

## Pass Parameters

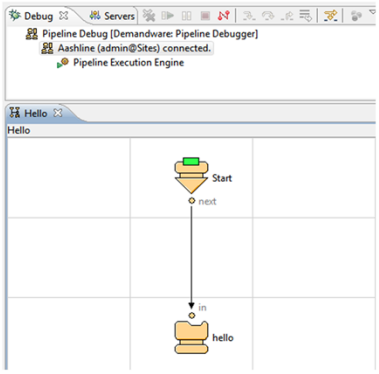





You can pass parameters to the pipeline dictionary using a URL query string:
The parameters will be added to the `CurrentHttpParameterMap` object inside the `pdict`. You can see the values stored in the pipeline dictionary when you run the pipeline debugger.
In this example, the `CurrentHttpParameterMap` 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.
**Note:** For more information on the different classes mentioned here, consult the Digital Script and Pipeline APIs in the Help menu.

## Lesson A.5: Troubleshooting with the Pipeline Debugger

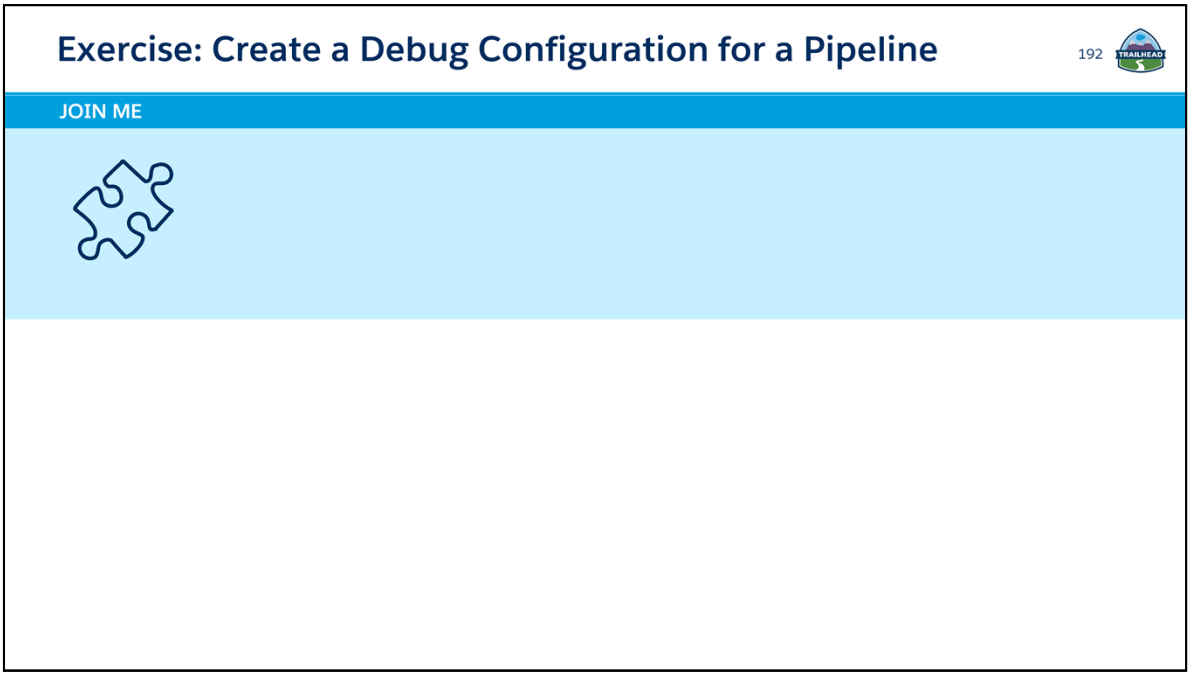






When testing your pipelines in the storefront, if you receive an error on execution, you can use the Request Log tool as well as the Studio Pipeline Debugger to troubleshoot your pipeline.

**Pipeline Debugger**
The Pipeline Debugger operates at the pipeline level, not at source code level. It provides step-by-step tracking for pipeline execution and for examining the pipeline dictionary at runtime.
The Debugger requires a running Commerce Cloud B2C Commerce system and a properly configured Remote Server connection. You also need to create a debug configuration before running the Debugger. To execute the pipeline debugger properly, you need a pipeline with breakpoints set on at least one node. When you launch a pipeline debugger, the breakpoint color changes from a semi-transparent green to solid green.




To create a debug configuration:

1. In UX Studio, go to the main menu and select **Run> Debug Configurations** or select **Debug Configurations** from the drop-down menu under the green bug icon:
2. Double-click the **UX Studio: Pipeline Debugger** to create a new configuration.
3. Enter a configuration name: **Pipeline Debug**.
4. In the **Server Configuration** section, click **Select**. Select the server connection you wish to debug then click **OK.**
5. In the **Site to Debug** section, click **Select**. Select the site to debug. Click **OK.**
6. Click **Apply** to save the configuration.
7. Click **Debug** to start the debugger.

## Set Breakpoints Where the Debugger Will Pause



- To set breakpoints where the Debugger will pause
- To debug a pipeline using a debug configuration
- To start a debugging session

To set breakpoints where the Debugger will pause:
1. In UX Studio, open the pipeline to debug.
2. Click the pipeline node where you want the Debugger to pause during execution.
3. Right-click and select **Add/Remove Pipeline Node Breakpoint** from the pop-up menu.

To debug a pipeline using a debug configuration:
1. In UX Studio, open the Debug perspective so that you can view a debugging session. Use one of the following methods:
- Window > Open Perspective > Other > Debug
- Click the Open Perspective icon on the upper-right corner and select Other > Debug.

To start a debugging session:
1. From the Debug icon, select **Debug Configurations.** Double-click the **Pipeline Debug** configuration.
2. Verify that the Pipeline Execution Engine is running.
3. In a browser, launch the pipeline to debug.
4. Click the UX Studio icon on the taskbar, it should blink since the breakpoint was triggered. Sometimes the OS will switch the context to UX Studio automatically, but this is rare.
5. In UX Studio, the execution stops at the breakpoint, as indicated by the vertical bars:

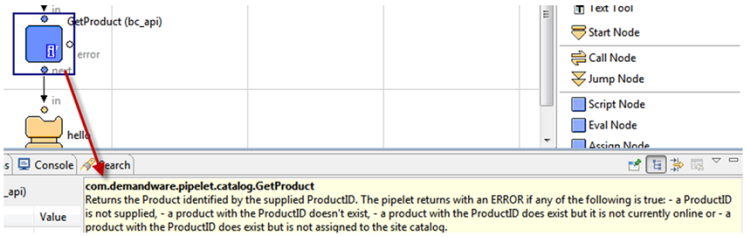
The **Debug Perspective** toolbar enables you to step through the pipeline. Or use the corresponding keyboard shortcuts:

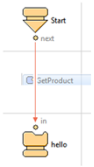
## Exercise: Use the Debugger for a Pipeline



JOIN ME

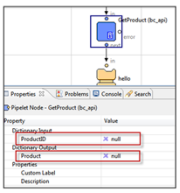
1. Create a pipeline debug configuration called **Pipeline Debug**.
2. Add a breakpoint on the `Call-Start` pipeline.
3. From the storefront, invoke the pipeline.
4. View the variables window when the debugger stops at the breakpoint.
5. Using the F5 key, step through the debugger. What are the values stored in the `pdict`? Rerun the **Call-Start** pipeline but this time add a parameter at the end of the string:
   `/Call-Start?param=1234`
6. Check the values of the `CurrentHttpParameterMap` after the Start Node.
7. Continue stepping thru the pipeline and observe changes in the `pdict`.

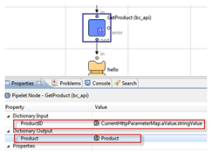



Drag and drop the pipelet onto the Transition node

Pipelets are the building blocks that implement storefront business logic in pipelines. Commerce Cloud provides pre-coded pipelets (in the bc_api cartridge) which implement common functionality to manage storefront objects, such as products, baskets, orders, and customers.





Undeclared Values Declared Values

- Pipelets are the building blocks that implement storefront business logic in pipelines.
- Commerce Cloud provides pre-coded pipelets (in the bc_api cartridge) which implement common functionality to manage storefront objects, such as products, baskets, orders, and customers.

In this section, you will use the pipeline dictionary to print information to a page using Pipelets.
A pipelet executes an individual business function within a Digital pipeline.
Pipelets are pre-coded pieces that manage storefront objects, such as products, baskets, orders, and customers. Pipelets are provided by Salesforce, but you can also use other types of pipelets from the palette such as:

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

Commerce Cloud Digital Pipelets are available in UX Studio via the Pipelets view. They belong to the `bc_api` cartridge, which downloada as part of the Digital API the first time you connect to a server.
There is a published API available under UX Studio Help menus or in the Commerce Cloud B2C Commerce documentation.
Each Digital 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.

To access and use a Digital API pipelet:

1. In UX Studio, open or create the pipeline where you wish to add a pipelet.
2. Open the Pipelet view tab.
3. Drag and drop the pipelet onto the Transition node between the two nodes where you want the pipelet to execute. Be sure the Transition node turns red when you hover your mouse pointer over it. Otherwise, the pipelet will not be connected to the node.
4. 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 shown, a GetProduct pipelet creates a product object by using a ProductID value. The value for the input comes from a stored variable in the pipeline dictionary. The product object output will need a name value.
5. Save your pipeline.