



SDL Digital Experience Accelerator Developer (Java)

Quick Start Guide

Updated: 06 September 2016

Version 1.3 – Document owner: Richard Hamlyn



Contents

GUIDE INTRODUCTION	3
INTRODUCTION TO MODEL, VIEW CONTROLLER (MVC).....	4
INTRODUCTION TO DXA	6
LAUNCHING YOUR DXA WEB APPLICATION IN ECLIPSE	12
CREATING A NEW MODULE IN DXA	20
MANAGING RESOURCES	30
APPENDIX A – INSTALLING MAVEN	32
APPENDIX B – INSTALLING TOMCAT IN ECLIPSE	35
AVAILABLE COURSE OPTIONS (CLASSROOM).....	38

Guide introduction

This short introduction to development with SDL Digital eXperience Accelerator (DXA) will introduce you to the basic concepts and knowledge required to commence development work.

The guide will:

- Introduce MVC development (optional).
- Introduce Digital eXperience Accelerator (DXA).
- Guide you to setting up a DXA development environment.
- Teach you how to create Views and Models in DXA.
- Introduce you to Resources in DXA.

The guide is split in to two main parts. Chapters 2-3 provide theoretical background information on MVC and DXA. If you are already comfortable with either of these topics then you should jump to Chapter 4 to start the practical sessions. Chapters 4-6 are practical sessions that will enable you to create your own development environment and commence some introductory development tasks that will provide a clear insight to the simplicity of development with SDL DXA.

Pre-requisites:

- JDK 1.7+, Eclipse IDE, with Tomcat server and Maven 3.2+ installed.
- Basic knowledge of MVC.

Notes:

DXA is also available in .Net but this document does not cover this.
You will connect to an SDL Web environment that has been pre-implemented with DXA.
You do not need to know any details of the Content Management System (CMS) to develop; however, the steps that have been pre-implemented in the CMS are detailed for your knowledge and background information.

If you want to know more about DXA you can attend one of the SDL Training courses by contacting learn@sdl.com.

Questions or help

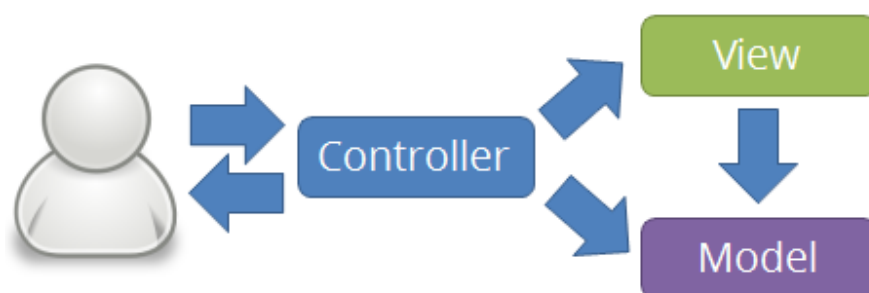
The guidance in this document has been tested and should always run if you follow the steps; however, if you have any questions or you require any further help then please post to the SDL Community (<http://community.sdl.com>). Please remember to identify the guide, including the fact that it is an Quick Start Guide (Java) that you are referring to.

Introduction to Model, View Controller (MVC)



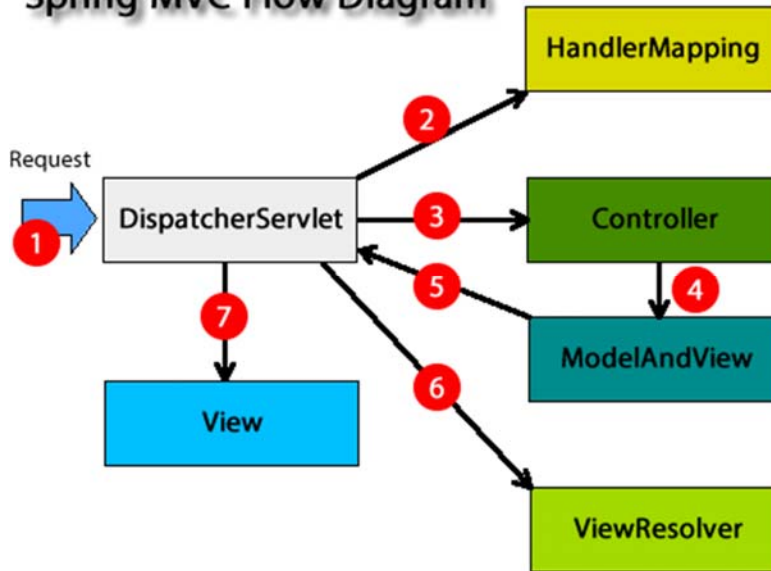
- **Spring MVC Framework**
 - The Model, View, Controller (MVC) methodology provides three logical layers:
 - **Model**
 - Domain objects that are processed by the service layer (business logic) or persistent layer (data source).
 - **View**
 - Displays data, normally as a JSP page written with the Java Standard Tag Library (JSTL).
 - **Controller**
 - URL mapping and interacts with service layer for business processing and returns a model.
 - MVC separation helps team development as you can work on **views**, **models** and **controller** logic in isolation.

Model, View, Controller (MVC)



The flow of a Java Spring MVC application is shown here:

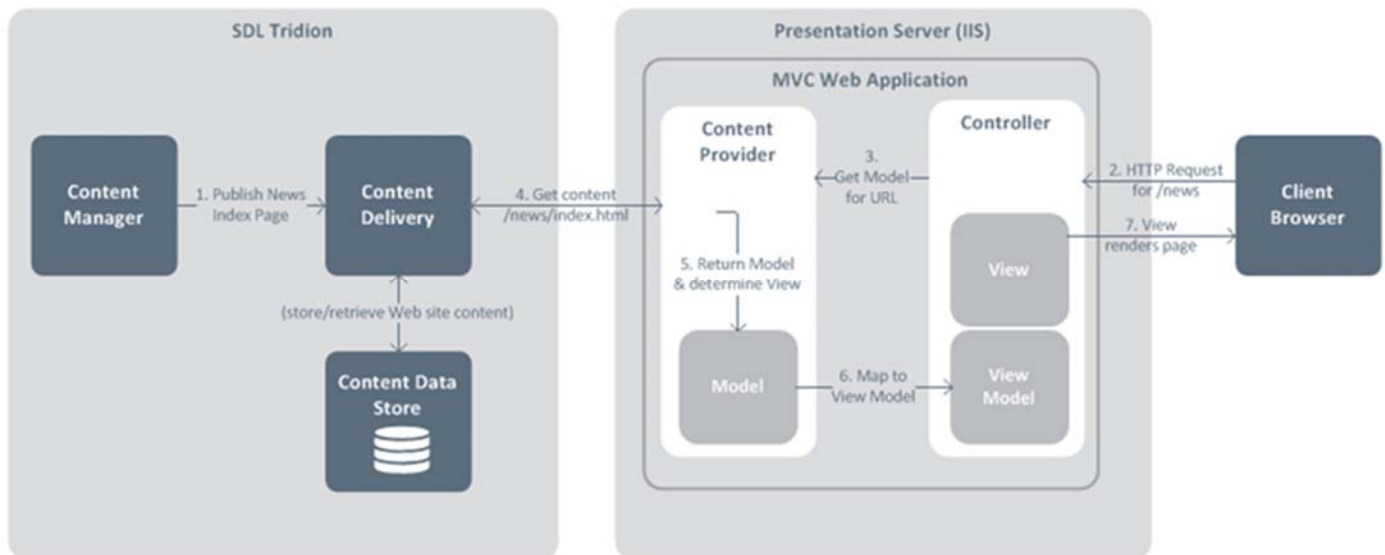
Spring MVC Flow Diagram



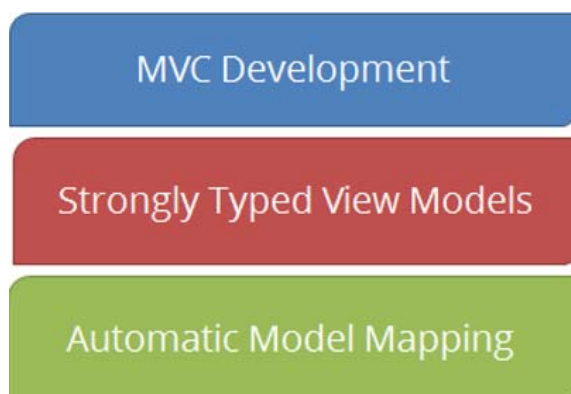
1. Request received by DispatcherServlet.
2. DispatcherServlet contacts the HandlerMapping to find out which Controller class name is associated with the request.
3. The request is transferred to the Controller for processing.
4. The Controller runs the request by executing the appropriate methods.
5. The ModelAndView object is returned (Model_data and View_name) back to the DispatcherServlet.
6. The DispatcherServlet sends the model object to the ViewResolver to get the view page.
7. Finally, the DispatcherServlet will pass the Model object to the View page to display the result.

Introduction to DXA

- **Digital eXperience Accelerator (DXA):**
 - Provides an optimized and accelerated way to deliver SDL driven sites.
 - Empowers users to design, create and publish sites quickly.
 - Provides a foundation for an organization to build on.
 - Provides rich user experience.
 - Encourages a standard approach to creating and managing Content Managed websites.
- **Architecture**

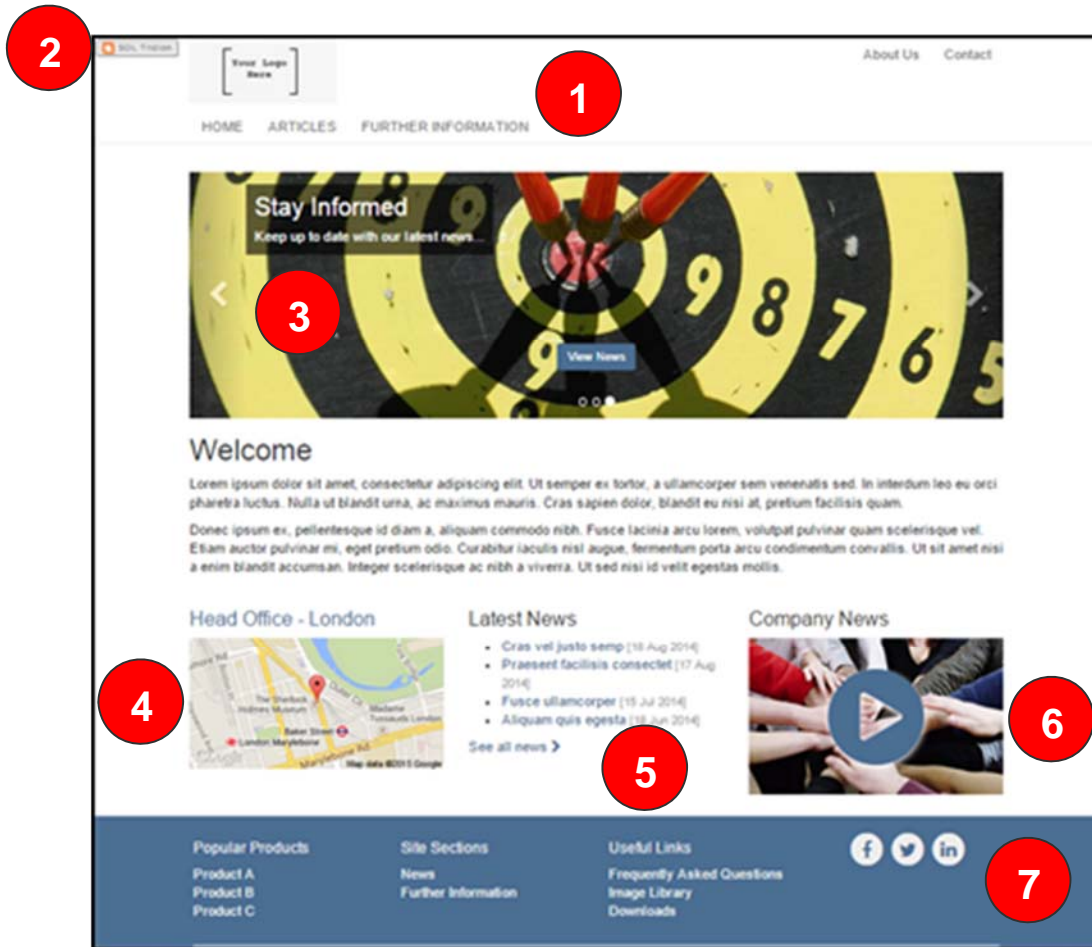


- **Key Functionality**



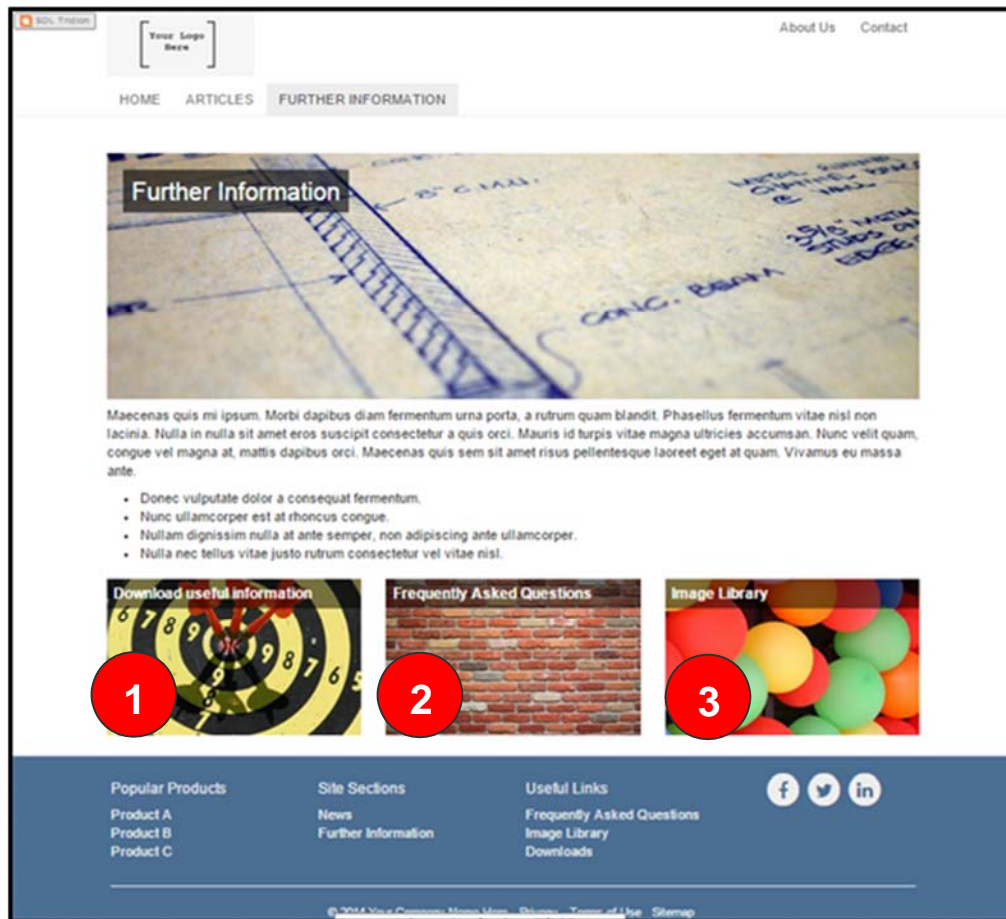
- Web Application
 - ASP.Net MVC 5
 - Requires a Content Data Store
 - XPM implemented by default
- BluePrint
 - Default Publication hierarchy (Linear)
- Security Model
 - Pre-defined
 - Developer
 - Editor
 - Site Manager
- Implemented in Content Delivery
- Uses JSP Views
- Content Manager TBBs are used to:
 - Publish Navigation
 - Configure the site
 - Provide resource data assets
 - Provide HTML design assets
- Integration
 - Most common social networks
 - YouTube
 - Google Maps
 - Google Analytics
- Navigation and sitemap
 - Top and left navigation
 - Breadcrumb
 - Sitemap and Google sitemap

○ Homepage



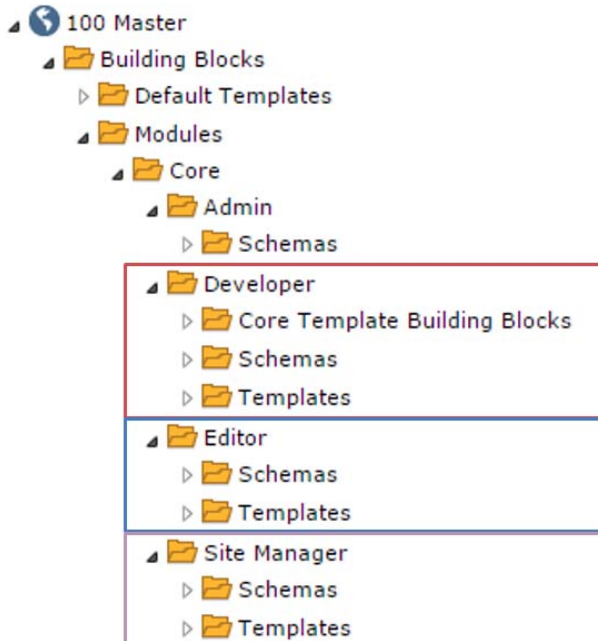
1. Header
2. Experience Manager (XPM)
3. Rotating Banner
4. Google Maps integration
5. Dynamic List
6. YouTube extension
7. Footer

- **Further Information page**



1. Download area
2. FAQ
3. Image library

- **Default Schemas** (Content structure that the MVC data model will map to)



Schemas are created in folders linked/restricted to roles



Developer

Edit site layout and functionality

Editor

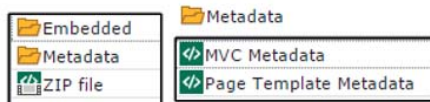
Edit content

Site Manager

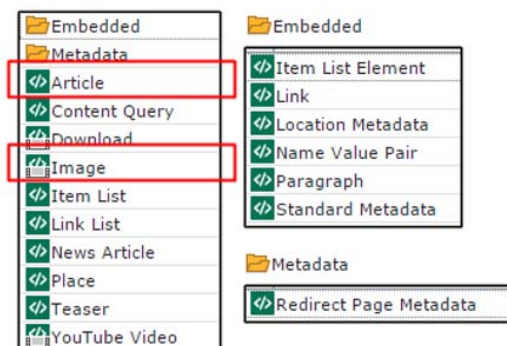
Edit content and configuration



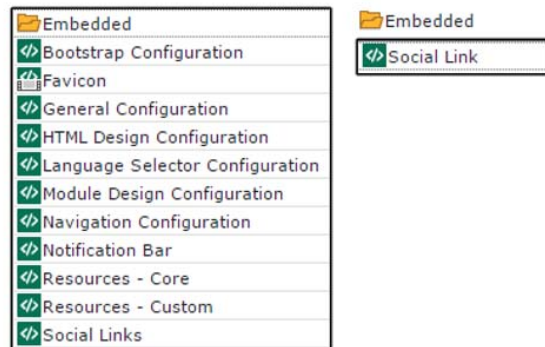
Developer



Editor



Site Manager



Article and Image Schemas have fields mapped to Schema.org schemas

Schema.org

Some DXA Schemas are mapped to Schema.org structure

Schema.org:

Common set of schemas for structuring website data in pages
Enables Search Engines to index and understand your site data
Implemented by all major search engines, including:



Provides search engine benefits, such as a Google results site search

[SDL: The Leader in Global Customer Experience Solutions](#)

www.sdl.com/ ▼

Bring your brand to the world & the world to your brand with integrated software & services for Language Translation & Digital Content Management by SDL.



Launching your DXA Web Application in Eclipse

In this task, you will launch the latest DXA Web Application from Maven and setup your development environment in the Eclipse IDE.



If you do not have Maven 3.2+ installed on your machine, you should perform the steps in Annex A to do so before progressing.

1. Assuming you have Maven defined in your 'Path' Environment Variable (see Annex A), open a Command Line Interface (from any location).
 - a. In the Command Line Interface enter 'mvn archetype:generate' and hit enter. This will load all the available archetypes from Maven.
 - b. You now need to filter the list of archetypes. In the filter option enter 'dxa' as shown below:

```
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 828: dxa
```

- c. You will now see a filtered list of archetypes similar to the below. Enter the number for the dxa archetype you want to build. In this example it is '1':

```
Choose archetype:
1: remote -> com.sdl.dxa:dxa-webapp-archetype (Maven Archetype for DXA web application)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): :
```

- d. You want to build the latest DXA Web Application, so you enter '4' as shown below:

```
Choose a number or apply filter (format: [groupId:]artifactId, contains): : 1
Choose com.sdl.dxa:dxa-webapp-archetype version:
1: 1.3.0
2: 1.4.0
3: 1.4.1
4: 1.5.0
Choose a number: 4: 4
```

Continued...



- e. Now update your archetype details as defined below:

groupId: **com.sdl.dxatraining**
artifactId: **dxatraining**
version: Hit enter to use the default value
package: Hit enter to use the default value

Note!

You must use 'dot' separation notation in the groupId otherwise your build will fail.

Enter 'Y' to continue and hit enter.

```
Define value for property 'groupId': com.sdl.dxatraining
Define value for property 'artifactId': : dxatraining
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': com.sdl.dxatraining: :
Confirm properties configuration:
groupId: com.sdl.dxatraining
artifactId: dxatraining
version: 1.0-SNAPSHOT
package: com.sdl.dxatraining
Y: : Y
```

- f. The Web Application will now download to the location shown in the basedir (defaults to the location you operate the Command Line from), highlighted below:

```
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
dxa-webapp-archetype:1.5.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\rhamlyn
[INFO] Parameter: package, Value: com.sdl.dxatraining
[INFO] Parameter: groupId, Value: com.sdl.dxatraining
[INFO] Parameter: artifactId, Value: dxatraining
[INFO] Parameter: packageName, Value: com.sdl.dxatraining
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\rhamlyn\dxatrai
ning
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 14:08 min
[INFO] Finished at: 2016-08-25T16:11:03+01:00
[INFO] Final Memory: 16M/325M
[INFO] -----
```

- g. Still in your Command Line Interface, change the directory to the root of your new Web Application package using '**cd dxatraining**'. See below:


```
C:\Users\rhamlyn>cd dxatraining
C:\Users\rhamlyn\dxatraining>
```

- h. Now you can build your package. At the Command Line Prompt run the following command '**mvn clean package -P web8**' as shown:

Once the package builds, you will see the results and a '**Build Success**' message:

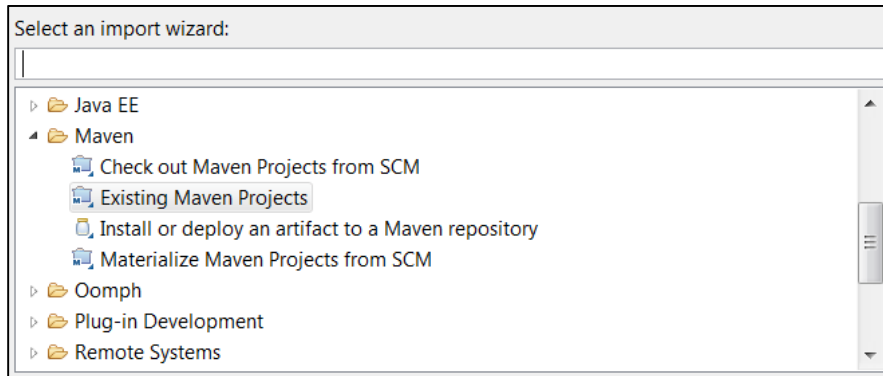
```
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] --- maven-war-plugin:2.4:war (default-war) @ dxatraining ---
[INFO] Packaging webapp
[INFO] Assembling webapp [dxatraining] in [C:\Users\rhamlyn\dxatraining\target\dxatraining]
[INFO] Processing war project
[INFO] Webapp assembled in [1291 msecs]
[INFO] Building war: C:\Users\rhamlyn\dxatraining\target\dxatraining.war
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.486 s
[INFO] Finished at: 2016-08-25T16:18:09+01:00
[INFO] Final Memory: 29M/324M
[INFO] -----
```

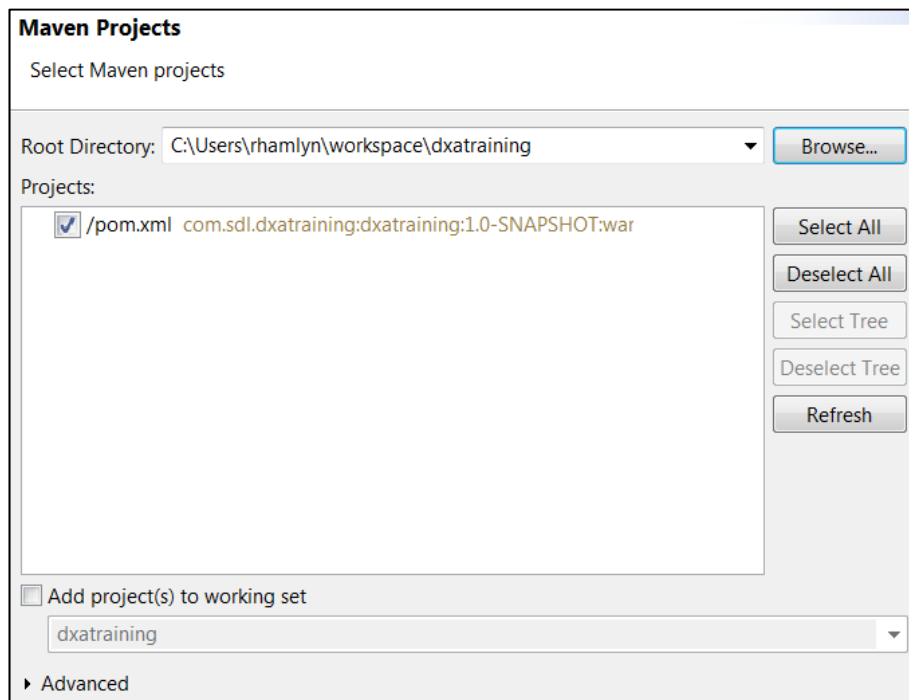
- i. You can now close the Command Line Interface.

2. Import your new DXA Web Application to Eclipse.

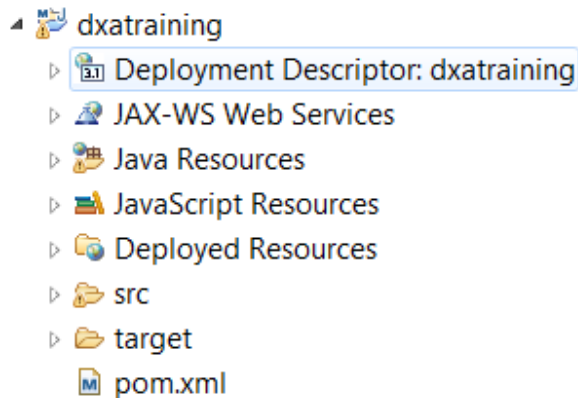
- a. Navigate to the location of your Web Application (detailed in basedir previously).
- b. Copy the entire '**dxatraining**' folder and paste it to your Eclipse Workspace. Default location '**c:\users\username\workspace**'.
- c. Open your Eclipse IDE.
- d. Select '**File**' and then 'Import'. From the popup window, select '**Maven**' and then '**Existing Maven Projects**' as shown below:



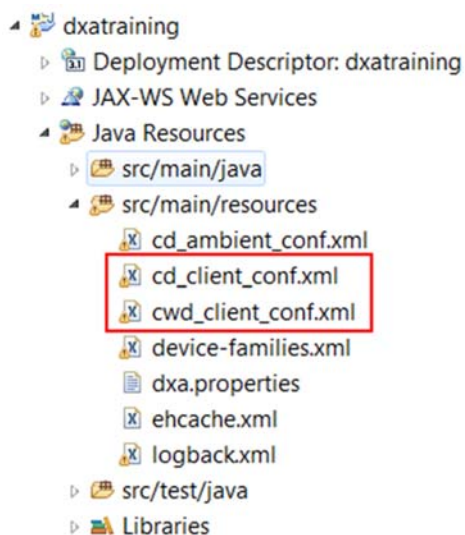
- e. Browse to your Maven Web Application that you just pasted in to your Eclipse Workspace:



- f. Click **'Finish'** to import. You will see the following project in the Eclipse Project Explorer:



- g. You now need to configure the Web Application to talk to the Content Management System that you will be using to provide the content.
- h. In the Project Explorer, open the two files **cwd_client_conf** and **cd_client_conf**.



- i. Update both files as shown below:

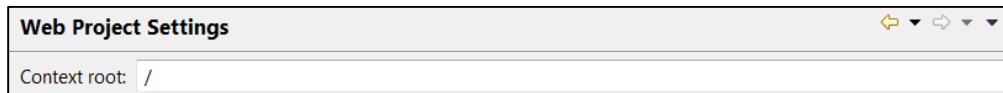
```
<!-- Specify your CIS Environment's Discovery Service URL below -->
<DiscoveryService ServiceUri="http://sdl-training.com:8082/discovery.svc"/>
```



There is a read only Content Service running at the above endpoint that you can use with your web application.

- j. Save and close both of the config files.

- k. Right-click on the project node in the Project Explorer and select '**Properties**'. A new window will open. Scroll to the bottom of the new menu and select '**Web project settings**'. Change the Context URL to '/' as shown:



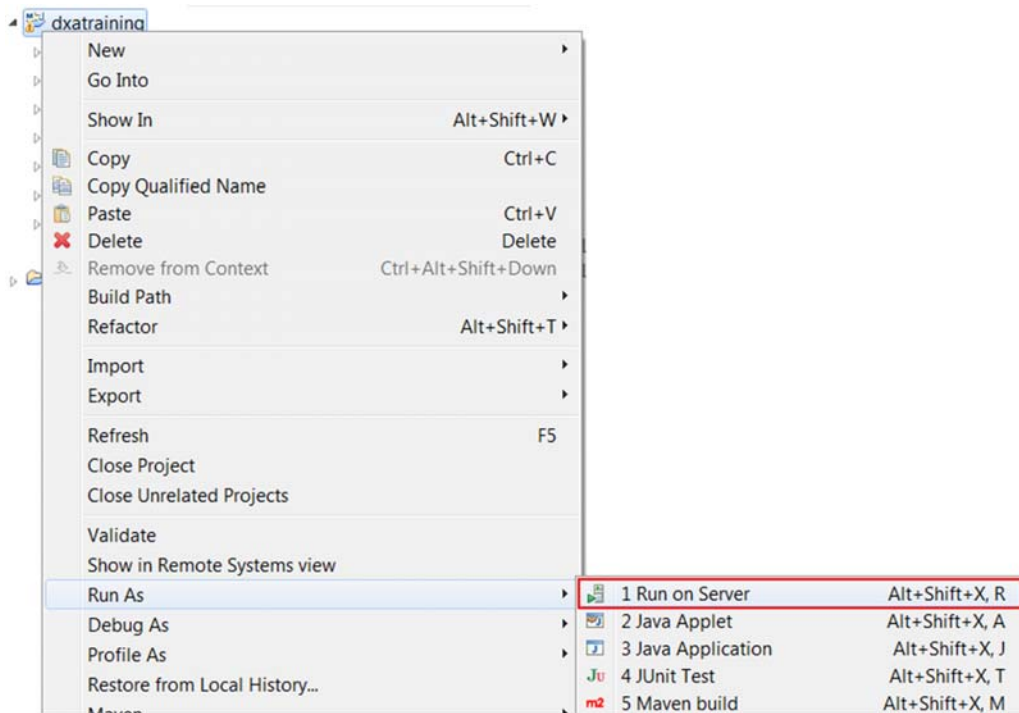
- i. Click '**OK**'.
- l. **Optional:** If you want to put your Web Application's logging in to Debug mode then open the **logback.xml** file and update the level as shown:

```
<configuration scan="true" scanPeriod="1 minute">
  <!-- Properties and variables -->
  <property name="log.pattern" value="%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n"/>
  <property name="log.history" value="7"/>
  <property name="log.level" value="DEBUG"/>
  <property name="log.encoding" value="UTF-8"/>
  <property name="log.folder" value="../logs/DXA"/>

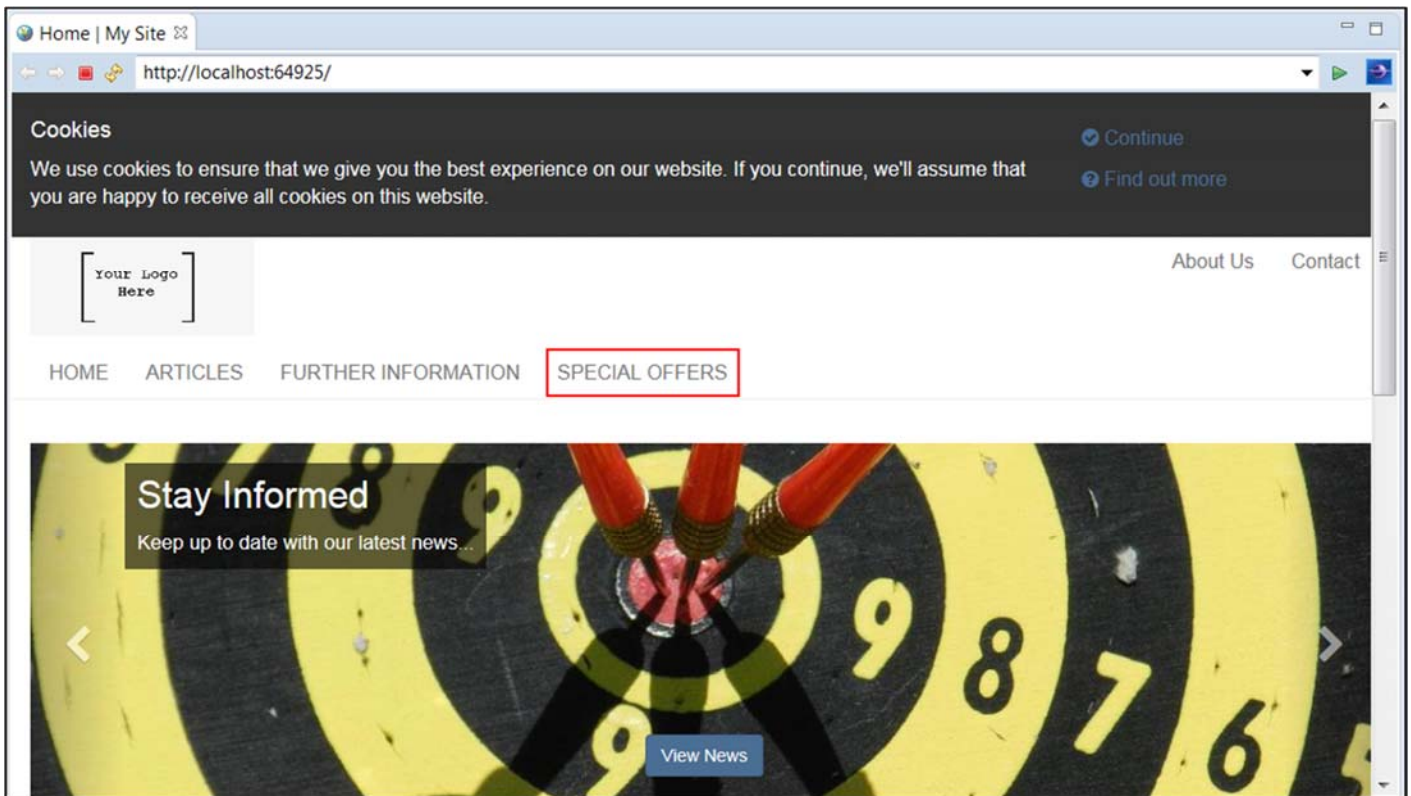
  <!-- Appenders -->
```

- i. Save and close the logback.xml file.
- m. Now you can run your project, assuming you have TomCat installed in Eclipse².
- n. Right-click on your project node and select '**Run as**' – '**Run on server**'.

² See Appendix B – Installing TomCat for Eclipse.



- o. Select your Tomcat server and click '**Finish**'.
- p. You should see the following DXA homepage load in your Eclipse window:



You are now ready to start development in DXA!

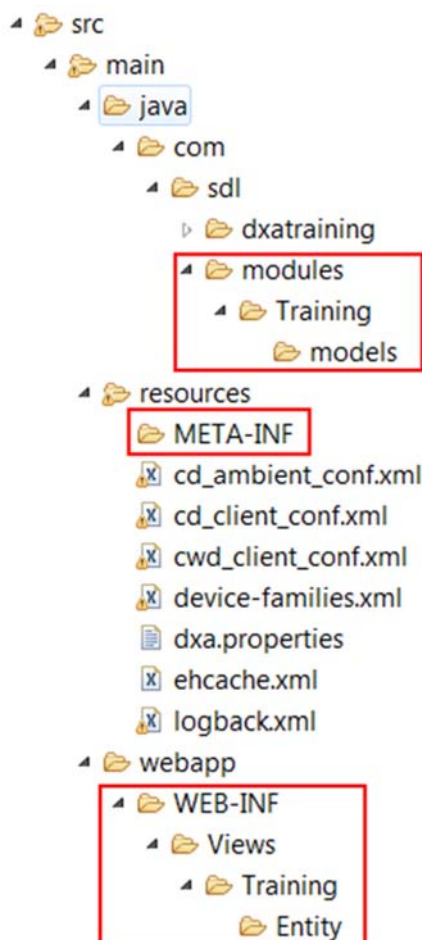


The above is a screen shot of the DXA homepage inside of an Eclipse development environment. The highlighted Special Offers area of the site is not a part of standard DXA but it is an extension implemented in the CMS by the authors. You will now create a **Model** and **View** for the new Special Offers part of the site.

Creating a new Module in DXA

In this task, you will create a new module to extend the DXA site and you will create a new Special Offer Model and View.

- I. Creating a module in your Web Application project.
 - a. In your Eclipse Project Explorer, create the following file structure to represent your new module.



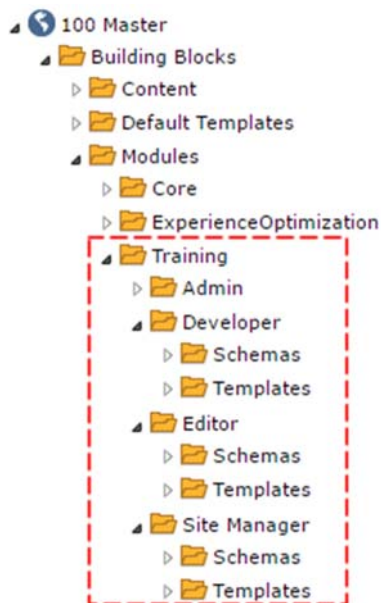
Note!

Modules help separate your code and functionality from that of the Core in DXA.

2. Create a View Model and View for the new Special Offers that the Business will manage from the CMS.



The following Module was created in the CMS already and this maps to our Module that you are creating next:



Information only!

You do not need to do anything with the content in the CMS.



Do not worry if you do not understand the CMS elements of this, it is added to the document to explain what has already been setup but is not essential information for a Developer. You can simply assume that the business have already implemented the Module on the Content side of the site and you are providing the Delivery side.

The following Schema exists in the CMS and you will map this to a Model in your web application next. This part is for information only and you do not need to do anything with Schemas.

General

General	Design	Metadata Design	Source	Workflow	Info
* Name:	<input type="text" value="Special Offer"/>				
* Description:	<input type="text" value="Special Offer"/>				
* Schema Type:	<input type="text" value="Schema"/>				
* Root Element Name:	<input type="text" value="SpecialOffer"/>				
Translate Name:	<input type="checkbox"/>				
Aggregate Translation Items:	<input type="checkbox"/>				

Design (Fields)

	XML Name	Description	Type
*	headline	Headline	Text
*	image	Image	Multimedia Link
*	description	Description	Text
*	link	Link	Component Link

Information only!

You do not need to do anything with Schemas in the CMS.

Metadata (Fields)

	XML Name	Description	Type
*	expiryDate	Expiry date	Date

This means you need a Model that can bind to CMS Components based on this Schema with the following fields:

headline: *Special Offer title*

image: *Jpeg image*

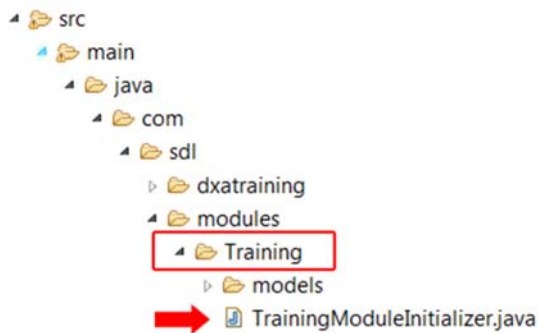
description: *Information about the Special Offer*

link: *A link to a related article to that of this Special Offer*

expiryDate: *The expiration of the Special Offer*

Inside the CMS, the Content Editors for the Special Offers promotion have already created some content that we can use for testing later.

3. You will now create the items in Eclipse that are required to employ the content from the CMS. Create a **Module_INITIALIZER** class for your new 'Training' module.
 - a. Navigate to your **Training** module folder, right-click and select 'New' – 'File'.
 - b. Create a new class as shown:



- c. Add the following code to your TrainingModuleInitializer class:

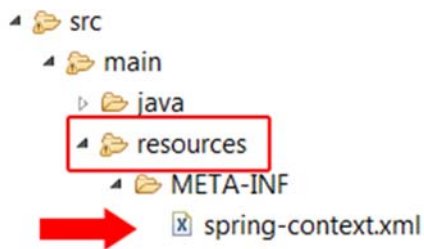
Code Box

```
package com.sdl.modules.Training;
import com.sdl.modules.Training.models.SpecialOfferModel;
import com.sdl.webapp.common.api.mapping.views.AbstractInitializer;
import com.sdl.webapp.common.api.mapping.views.RegisteredViewModel;
import com.sdl.webapp.common.api.mapping.views.RegisteredViewModels;
import org.springframework.stereotype.Component;

@Component
@RegisteredViewModels({
    @RegisteredViewModel(viewName = "SpecialOffer", modelClass =
SpecialOfferModel.class)
})

public class TrainingModuleInitializer extends AbstractInitializer{
    @Override
    protected String getAreaName(){
        return "Training";
    }
}
```

- d. Now navigate to **META-INF** in the Project Explorer.
 - e. Create a new XML file named **spring-context** as shown by right-clicking resources and selecting 'New' – 'File':



- f. Add the following XML to the spring-context file and save.

Code Box

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-
                           beans.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-
                           context.xsd">

  <!-- <context:annotation-config/> -->
  <context:component-scan base-package="com.sdl.modules.Training"/>

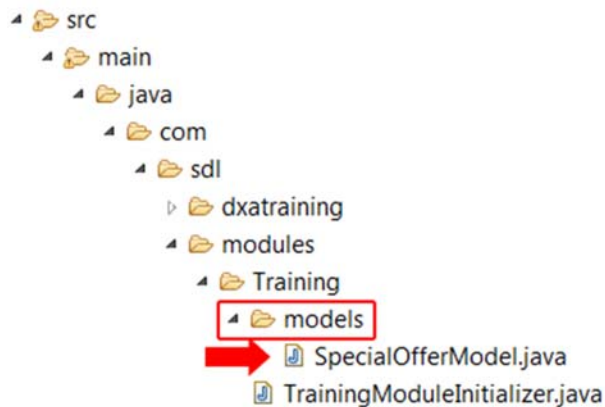
</beans>
```

4. Create the **View Model** that will map to the Special Offer Schema created by the business in the CMS.



A View Model is a simple class to represent the content you want to display. The class has strongly typed properties to enable you to easily refer to them in your View.

- a. Create a new class named '**SpecialOfferModel.java**' in your module's **models** folder as shown:



b. Add the following code and semantic mappings to the model and save:

Code Box

```

package com.sdl.modules.Training.models;

import org.joda.time.DateTime;

import com.sdl.webapp.common.api.mapping.semantic.annotations.SemanticEntity;
import com.sdl.webapp.common.api.mapping.semantic.annotations.SemanticProperty;
import com.sdl.webapp.common.api.model.entity.AbstractEntityModel;
import com.sdl.webapp.common.api.model.entity.Image;

import static
com.sdl.webapp.common.api.mapping.semantic.config.SemanticVocabulary.SDL_CORE;

@SemanticEntity(entityName = "SpecialOffer", vocabulary = SDL_CORE, prefix = "s")

public class SpecialOfferModel extends AbstractEntityModel {

    @SemanticProperty("s:headline")
    private String headline;
    @SemanticProperty("s:image")
    private Image image;
    @SemanticProperty("s:expiryDate")
    private DateTime expiryDate;
    @SemanticProperty("s:link")
    private String link;
    @SemanticProperty("s:description")
    private String description;

    public String getHeadline() {
        return headline;
    }

    public void setHeadline(String headline) {

```

```

        this.headline = headline;
    }

    public Image getImage() {
        return image;
    }

    public void setImage(Image image) {
        this.image = image;
    }

    public DateTime getExpiryDate() {
        return expiryDate;
    }

    public void setDate(DateTime expiryDate) {
        this.expiryDate = expiryDate;
    }

    public String getDescription() {
        return description;
    }

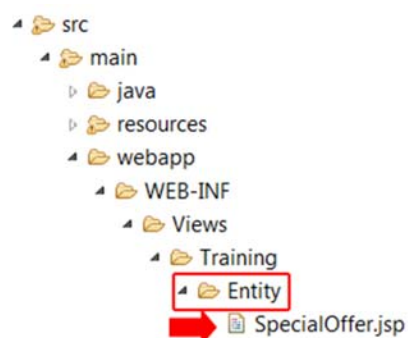
    public void setDescription(String description) {
        this.description = description;
    }

    public String getLink() {
        return link;
    }

    public void setLink(String link) {
        this.link = link;
    }
}

```

- c. Now you can create a View to display your model data.
 - i. Create a View named '**SpecialOffer.jsp**' in the location shown below:



- d. Add the following code to the **SpecialOffer.jsp** View and save:

Code Box

```
<%@ taglib prefix="dxa" uri="http://www.sdl.com/tridion-dxa" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

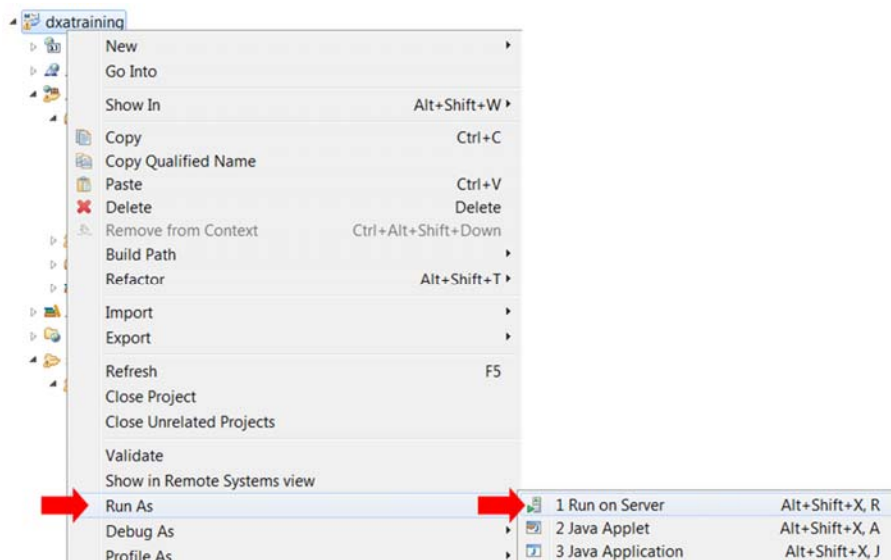
<jsp:useBean id="entity" type="com.sdl.modules.Training.models.SpecialOfferModel"
scope="request"/>

<div class="content ${entity.htmlClasses}" ${markup.entity(entity)}>

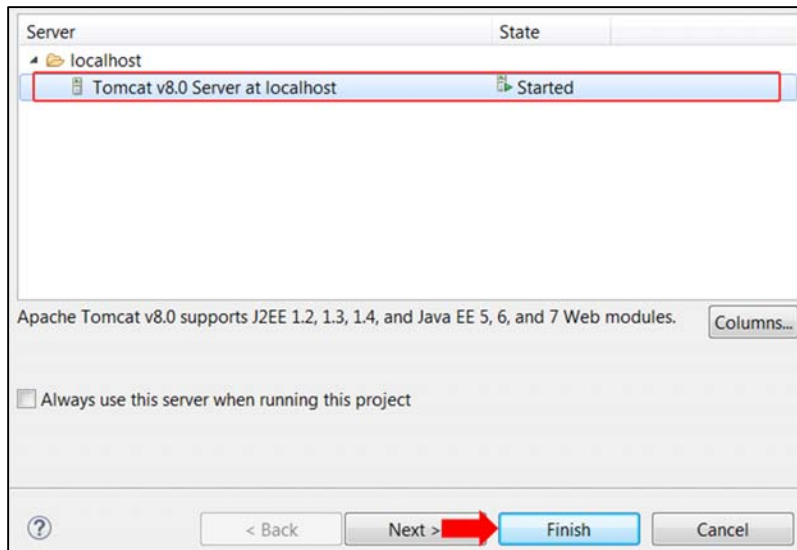
    <c:if test="${not empty entity.image}">
        <div ${markup.property(entity, "image")}>
            <dxa:media media="${entity.image}" aspect="3.3"/>
        </div>
    </c:if>

    <h1 ${markup.property(entity, "headline")}> ${ entity.headline }</h1>
    <p ${markup.property(entity, "description")}> ${ entity.description }</p>
    <a href="${ entity.link }" class="btn btn-primary"
${markup.property(entity, "link")}>Read on...</a>
    <br />
    <p class="meta small">Valid until <span ${markup.property(entity,
"headline")}> ${markup.formatDate(entity.expiryDate)}</span></p>
</div>
```

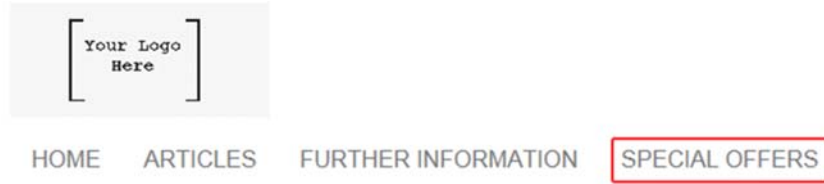
- e. You can now run your project to test everything works okay.
f. Right-click on your project and select **'Run as' – 'Run on server'**.



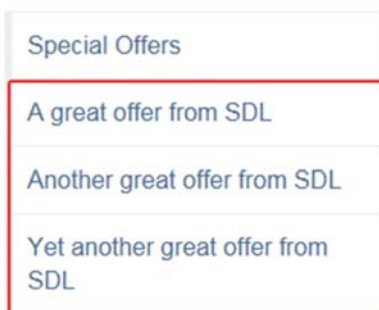
- g. Select your Tomcat server and click '**Finish**' to run the project.



- h. You will see a lot of activity in the Eclipse Console as the Web Application starts to communicate with SDL Web to retrieve your requested homepage.
i. Once the homepage loads, click on the '**Special Offers**' menu item.



- j. The Special Offers Index Page will load. However, this is using a DXA standard View so it does not mean your View is working. You need to click on an actual Special Offer to test your View. Click on one of the Special Offers.




- k. The Special Offer content created in the CMS will now load in the View you created for Special Offers. You will notice that the content reflects your Model too.


Special Offers

A great offer from SDL

Another great offer from SDL

Yet another great offer from SDL





Yet another great offer from SDL

Don't get lost with this amazing offer!

[Read on...](#)

Valid until Sunday, December 25, 2016

Managing Resources

Resources provide snippets of content used across the website. This allows the business to manage and control parts of the website that would normally require IT intervention. In this section, you will implement a resource that has already been Published by the business that will give them control over the text in a site-wide button.

1. The following Resource was created in the CMS by the business and has been Published. This means it is available to the web application.



Configuration Name:
Special Offer Resources

Settings:

+ -

* Name:
moreInfoText

* Value:
More Info

Information only!

This shows how the business would create/update a Resource in the CMS.

2. Write out Resource values in the SpecialOffer View:
 - a. Update your **"SpecialOffer"** view link (in your **Module** in the project) as follows to write out the **"moreInfoText"** resource:
 - b. Use the **Html.Resource** and replace the View's existing link with the code below.

Code Box

```
<a href="{ entity.link }" class="btn btn-primary" ${markup.property(entity, "link")}><dx:resource key="training.moreInfoText" /></a>
```

3. Run your project and navigate to one of your Special Offers.



Yet another great offer from SDL

Don't get lost with this amazing offer!

More Info

Valid until Sunday, December 25, 2016

- a. You will see that the text in the button is being injected from the Resources variable that was set by the business authors. This has now been applied site wide and is controlled by the business so they can update as and when they need to.

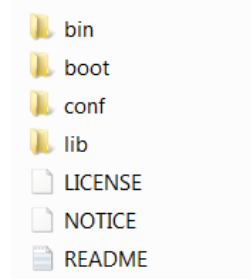
Appendix A – Installing Maven

1. You must have Maven 3.2+ installed to download DXA from Maven.
2. Download the Maven bin package from the official site at <https://maven.apache.org/download.cgi>
3. Select the bin package as shown here (note, it is okay to use a later version if one is available).

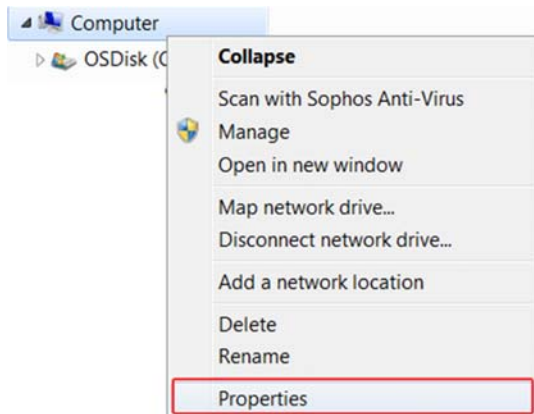
Binary tar.gz archive	apache-maven-3.3.9-bin.tar.gz
Binary zip archive	apache-maven-3.3.9-bin.zip
Source tar.gz archive	apache-maven-3.3.9-src.tar.gz
Source zip archive	apache-maven-3.3.9-src.zip

4. Once the download is complete, unzip the package and place the contents in a location that is easy to find.

c:\apache-maven



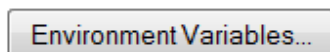
5. You need to create an environment variable so that other software can find your Maven instance.
6. Open your Computer's properties.



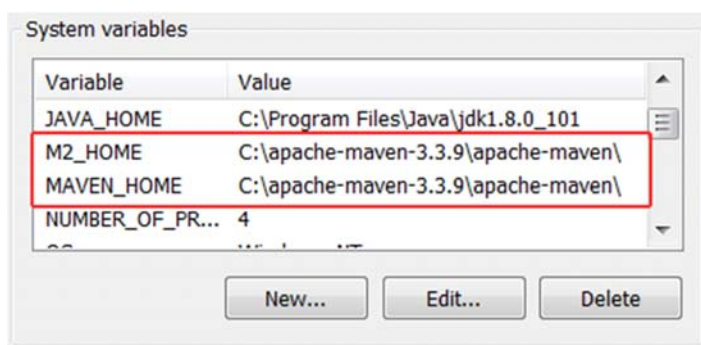
7. In the window that opens click on the Advanced system settings option:



8. Click on Environment Variables:



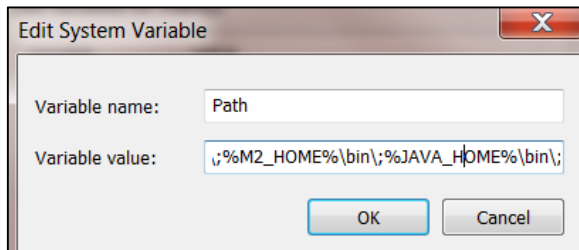
9. Add the following two System variables.





The value for each should be the root folder where you saved your instance of Maven.

10. In the System variables pane, scroll down to the **Path** entry, select and click on edit.
11. Enter the path to your M2_Home variable by adding **;%M2_HOME%\bin** at the end of the line



This will make sure that you can access Maven from any location on your local machine.

12. Test that Maven works from any location by opening your command line interface tool and running the command 'mvn -version'

```
C:\Users\rhamlyn>mvn -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T16:41:47+00:00)
Maven home: C:\apache-maven-3.3.9\apache-maven
Java version: 1.8.0_101, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_101\jre
Default locale: en_GB, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

13. You now have Maven installed and ready to use.

Appendix B – Installing Tomcat in Eclipse

1. Ensure you have a Tomcat server ready to test the Web Application and commence development exercises.
 - a. If your Eclipse environment does not have an integrated Tomcat Server already configured then follow these steps to do this now.
 - b. Download a Tomcat server from <http://tomcat.apache.org/> in this exercise you are using version 8 and 9 (you only need one of these).

- Core:
 - [zip \(pgp, md5, sha1\)](#)
 - [tar.gz \(pgp, md5, sha1\)](#)
 - [32-bit Windows zip \(pgp, md5, sha1\)](#)
 - [64-bit Windows zip \(pgp, md5, sha1\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, md5, sha1\)](#)
- Full documentation:
 - [tar.gz \(pgp, md5, sha1\)](#)

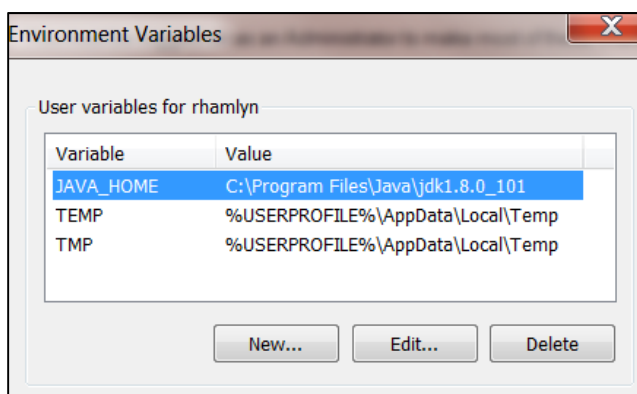


This assumes you have a 64 bit machine.

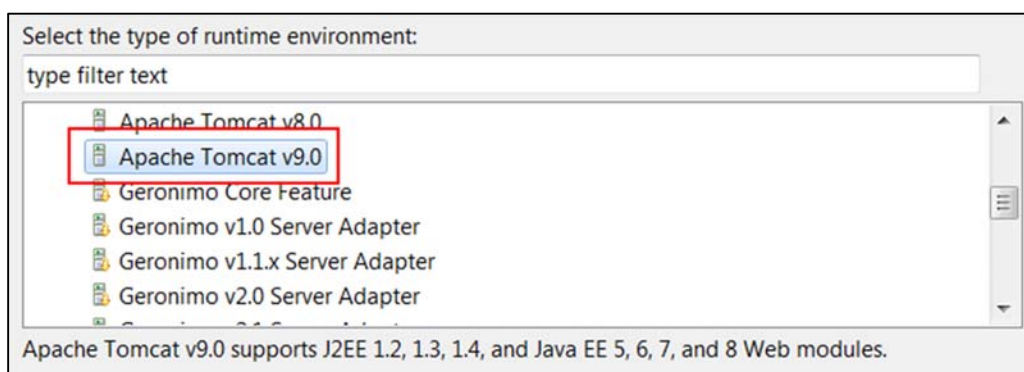
- c. Unzip the downloaded package and place it in an easy to find location on the file system i.e. **c:/Apache/Tomcat/**



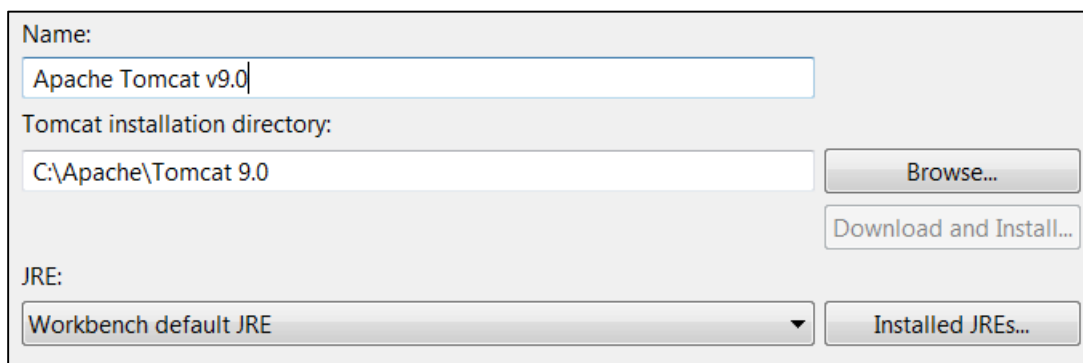
For Tomcat to run you must have an Environment Variable setup that points to your JDK (JAVA_HOME). For example:



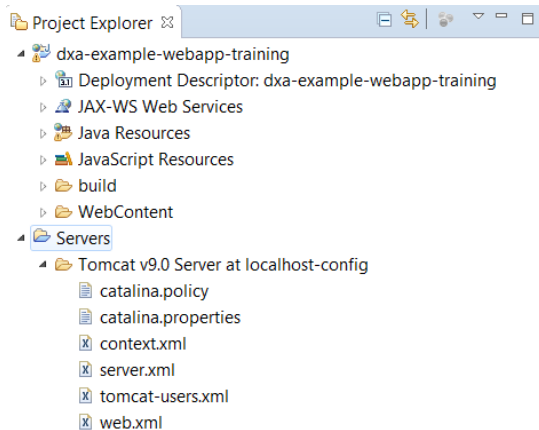
- d. Now that you have an instance of Tomcat ready to use you can implement this in Eclipse.
2. Add a server to your project so that you can run the project (only do these steps if you do not have a server already in Eclipse).
 - a. From the Eclipse menu, select '**Window**' and then '**Preferences**'. A new window will appear. Scroll down to the **Servers** option and select.
 - b. Select Runtime environment and click on the '**Add**' button.
 - c. From the popup window, select the version of Tomcat you intend to use.



- d. Click Next.
- e. In the next window browse to the root of your downloaded Tomcat instance.



- f. Click Finish and your server will be created and you will see it added to your Project Explorer:



- g. The SDL Web site that you will connect to has been defined with the following address:
http://localhost:
- h. Open the Server's **server.xml** file.



- i. Find the following entry and change the Tomcat port to **64925** so that it will map to the Web Application defined in SDL Web's Topology.

```
<Connector connectionTimeout="20000" port="64925" protocol="HTTP/1.1" redirectPort="8443"/>
```

- j. Save and close the server.xml file.

3. You are now ready to use Tomcat from within your Eclipse environment.



All logging whilst running the project will be displayed in the Console, including any errors you may come across.

Available course options (classroom)

This guide is intended to provide a quick introduction to development with Digital eXperience Accelerator and SDL Web 8. There are further options for learning that go far beyond the depth of this guide and that are available through the SDL Web Developer course provided by SDL Product Training.

SDL Web 8 Developer Course

The course provides a technical introduction to developing SDL Web powered web sites using the Model View Controller (MVC) design pattern.

The course simulates a pre-built environment created through SDL's Digital eXperience Accelerator (DXA) delivering best-practice examples and methodology to accelerate time to market.

This hands-on course introduces a developer, or technical user, to the key advantages of developing with DXA. The course uses Visual Studio or Eclipse as the chosen IDE and assumes you are familiar with terminology and basic methodology associated with developing an MVC site.

During the course, all participants will build a new site from scratch by employing DXA methodology that can be used in any implementation.

Topics covered:

- Introduction to SDL Web 8
- Introduction to MVC Module
- Areas and Modules Module
- Views and View Models Module
- Managing Resources Module
- Managing Configuration Module
- Managing Page Includes Module
- Web site layout (Page and Regions)
- DXA Templates
- Creating Custom Controllers
- Creating Functional Modules
- Automation Development
- Workflow Development

For further details please contact SDL Product Training at learn@sdl.com

Website: [SDL Training](#) (link)