04/20/2021

Contents

Sling Models for component logic

Introduction

Sling Models provide excellent support for web content authors to build pages in Adobe Experience Manager (AEM) and to easily customize the pages in AEM to their requirements. Sling Models let AEM developers access Sling content without creating their own adapters, thus avoiding a large amount of boilerplate code.

Objectives

After completing this course, you will be able to:

- Describe Sling Models
- Create a Sling Model
- Discuss the Sling Model Exporter in AEM
- Extend a core component

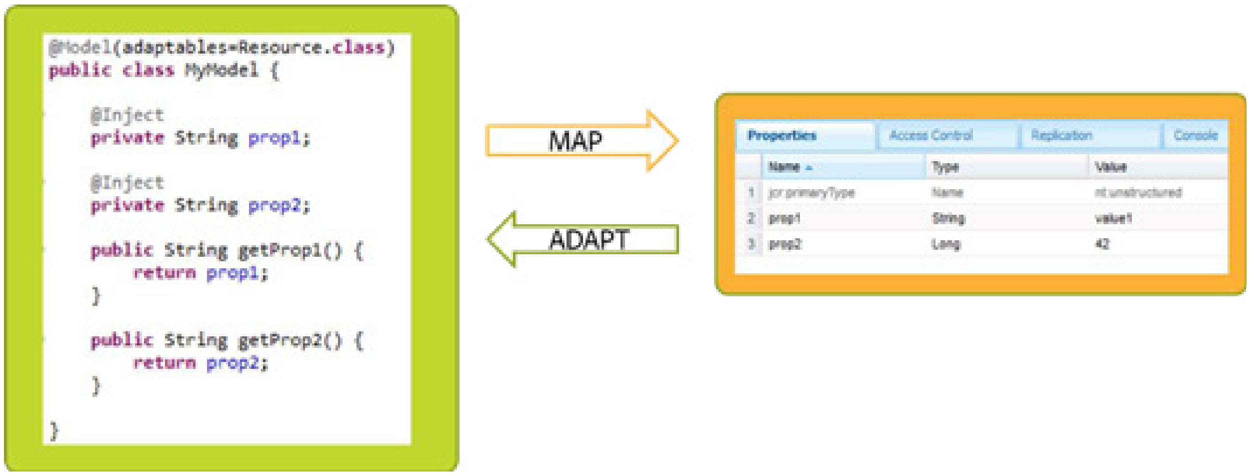
Working with Sling Models

Introduced with Sling 7 (2014), Sling Models allow you to define Java model objects (classes or interfaces) and map those objects to Sling resources (like a Java Content Repository (JCR) data structure or a SlingHttpServletRequest), or even to OSGi services, expanding the use of the adaptTo() method to any Java object.

A Sling Model can be seen as an abstraction layer that allows AEM components to consume back-end logic, just like the JS-Use, WCMUsePojo, or Sling Resource APIs would, but for complex components with the need for reusability.

Using Sling Models

A Sling Model is a Java class located in an OSGi bundle that is annotated with @Model and the adaptable class (for example, @Model(adaptables = Resource.class) for a resource).
In the example below, data members (the fields of the Java class) are annotated with @inject to map to node properties.



```
@Model(adaptables=Resource.class)
public class MyModel {

    @Inject
    private String prop1;

    @Inject
    private String prop2;

    public String getProp1() {
        return prop1;
    }

    public String getProp2() {
        return prop2;
    }
}
```

	Name	Type	Value
1	jcr:primaryType	Name	nt:unstructured
2	prop1	String	value1
3	prop2	Long	42

Benefits of using Sling Models

- Saves time from creating your own adapters (avoiding boilerplate code)
- Supports both classes and interfaces
- Allows you to adapt multiple objects—minimal required objects are Resource and SlingHttpServletRequest
- Works with existing Sling infrastructure (for example, changes to other bundles are not required)
- Provides the ability to mock dependencies with tools like Mockito @InjectMocks

Annotation Usage:

The following table shows the different types of Sling Model annotations and their usage:

Sling Model Annotation	Code Snippet	Injector
@Model	@Model(adaptables = Resource.class)	Class is annotated with @Model and the adaptable class
@Inject	@Inject private String propertyName; (class) @Inject String getPropertyName(); (interface)	a property named propertyName will be looked up from the Resource (after first adapting it to a ValueMap) and it is injected. If property is not found, it will return null.
@Default	@Inject @Default(values="AEM") private String technology;	A default value (for Strings, Arrays & primitives)

@Optional	@Inject @Optional private String otherName	@Injected fields/methods are assumed to be required. To mark them as optional, use @Optional, for example, resource or adaptable may or may not have properties.
@Named	@Named @Inject@ Named("title") private String page Title;	Inject a property whose name does not match the Model field name.
@Via	@Model(adaptables=SlingHttpServletRequest.class) Public interface SlingModelDemo { @Inject @Via("resource") String getPropertyName(); }	Use a JavaBean property of the adaptable as the source of the injection.
@Source	@Model(adaptables=SlingHttpServletRequest.class) @Inject @Source("script-bindings") Resource getResource();	Explicitly tie an injected field or method to a particular injector (by name). //Code snippet will ensure that resource is retrieved from the bindings, not a request attribute.

@PostConstruct	@PostConstruct protected void sayHello() { logger.info("hello"); }	Allows for the definition of methods to be executed after the instance has been instantiated, and all the injects have been performed.

Injector-specific Annotations

Sling Models are easily extensible, with custom injectors and annotations. To create a custom injector, simply implement the org.apache.sling.models.spi.Injector interface and register your implementation with the OSGi service registry. Here is a list of the available injectors:
https://sling.apache.org/documentation/bundles/models.html#available-injectors
Sometimes, it is necessary to use customized annotations that aggregate the standard annotations described above. This has the following advantages over using the standard annotations:

- Less code to write (only one annotation is necessary in most of the cases)
- More robust (in case of name collisions among the different injectors, you must ensure the right injector is used)
- Better IDE support

The following list of annotations maps to specific injectors:

Annotation	Supported Optional Elements	Injector
@ScriptVariable	optional and name	script-bindings
@ValueMapValue	optional, name, and via	valuemap
@ChildResource	optional, name, and via	child-resources
@RequestAttribute	optional, name, and via	request-attributes
@ResourcePath	optional, path, and name	resource-path
@OSGiService	optional, filter	osgi-services
@Self	optional	self
@SlingObject	optional	sling-object

Injectors Lookup in Web Console

You can see a list of the available injectors at: http://localhost:4502/system/console/status-slingmodels

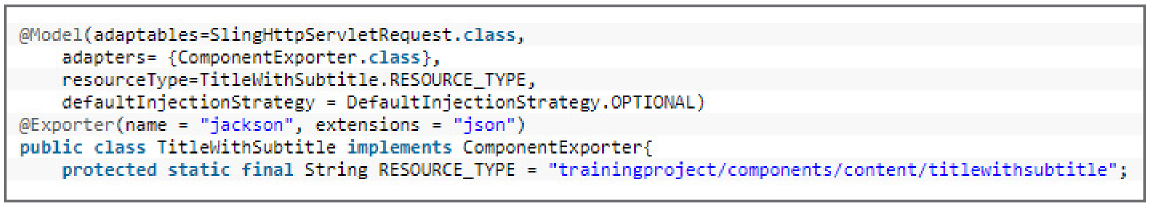
Debugging

In order to see logging specific to Sling Models, add a logger for the package **org.apache.sling.models**, set the log level to TRACE, and dump the messages in to the appropriate log file.

Sling Model Exporter

Primarily used to easily expose data to HTML Template Language (HTL) scripts with minimal code, Sling Models are becoming a major development business object in AEM.
The Sling Model Exporter feature allows new annotations to be added to Sling Models that define how the Model can be exported as a different Java object, or more commonly, serialized into a different format such as JSON, to be consumed by your front-end application (for example, headless Content Management Systems (CMSs)).

Apache Sling provides a Jackson JSON exporter to cover the most common case of exporting Sling Models as JSON objects. The exporter scans through all the getters and serializes them into JSON. The model needs to define the resourceType of the component as a modifier, in order to be bound to it.
The snippet below shows the declaration of a model for an AEM component, that gets exported with the Jackson exporter using the specific component adapter interface, com.adobe.cq.export.json.ComponentExporter:



```
@Model(adaptables=SlingHttpServletRequest.class,
    adapters= {ComponentExporter.class},
    resourceType=TitleWithSubtitle.RESOURCE_TYPE,
    defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
@Exporter(name = "jackson", extensions = "json")
public class TitleWithSubtitle implements ComponentExporter{
    protected static final String RESOURCE_TYPE = "trainingproject/components/content/titlewithsubtitle";
```

Exercise 1: Extend a core component

The AEM core components are building blocks for modern web development. Sometime these core components need more functionality and properly extending them is vital to ensure upgradability and maintainability.

In this exercise you will learn how to properly extend a sling model that is the business logic for a core component. You will also upload and observe an extended component using your new Sling Model.

This exercise includes the following tasks:

1. Install the Java code
2. Install the content package
3. Test the component

Task 1: Install the Java code

1. Open the IDE containing your Maven project for this course if not already opened.
2. In the IDE navigation on the left, find the **core** module.
3. Navigate to **core** > **src** > **main/java/com/adobe/training/core.**
4. Right-click **models** and choose **New File.**
5. Name the file as **TitleWithSubtitle.java**.
6. To add code to your **TitleWithSubtitle.java**, open up the **Exercise_Files-EC** folder for this course and navigate to **/core/src/main/java/com/adobe/training/core/models/**.
7. Open **TitleWithSubtitle.java** and copy the contents.
8. In your IDE, paste the contents into your Maven Project > **TitleWithSubtitle.java**.
9. Double-click **TitleWithSubtitle.java**. The file opens in the editor, as shown:

```
package com.adobe.training.core.models;
import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.models.annotations.DefaultInjectionStrategy;
import org.apache.sling.models.annotations.Exporter;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.Via;
import org.apache.sling.models.annotations.injectorspecific.ScriptVariable;
import org.apache.sling.models.annotations.injectorspecific.Self;
import org.apache.sling.models.annotations.injectorspecific.SlingObject;
import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
import org.apache.sling.models.annotations.via.ResourceSuperType;
import org.slf4j.Logger;
import com.adobe.cq.export.json.ComponentExporter;
import com.adobe.cq.wcm.core.components.models.Title;
import com.day.cq.wcm.api.Page;
/**
 * This title model extends the core title model: com.adobe.cq.wcm.core.components.models.Title
 * The core title model is used in both v1 and v2 core title components: core/wcm/components/title
 * In this model, we extend the title model to include a subtitle.
 *
 * In order to extend a core component model, in core/pom.xml add the dependency:
       <dependency>
           <groupId>com.adobe.cq</groupId>
           <artifactId>core.wcm.components.core</artifactId>
       </dependency>
 *
 * Note: You don›t need to add this dependency to the parent pom because it is already included.
 */

@Model(adaptables=SlingHttpServletRequest.class,
    adapters= {ComponentExporter.class},
    resourceType=TitleWithSubtitle.RESOURCE_TYPE,
    defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
@Exporter(name = "jackson", extensions = "json")
public class TitleWithSubtitle implements ComponentExporter{
    protected static final String RESOURCE_TYPE = "training/components/titlewithsubtitle";

    @Inject
    @Named("log")
    private Logger logger;

    @SlingObject
    private SlingHttpServletRequest request;
    //HTL global object in the model
    //Learn more  at Helpx > HTL Global Objects
    @ScriptVariable
    private Page currentPage;

    //property on the current resource saved from the dialog of a component
    @ValueMapValue
    private String subtitle;

    //Core Title model we are extending
    @Self @Via(type = ResourceSuperType.class)
    private Title coreTitle;
    //Method called when the model is initialized
    @PostConstruct
    protected void initModel(){
        //setup properties that are extending the title model
        if(subtitle == null){
            subtitle = "";
        }
    }
    //Next 2 methods required to support JSON display for this Component
    public String getTitle() {
        return coreTitle.getText();
    }
    public String getLinkURL() {
        return coreTitle.getLinkURL();
    }

    public String getSubtitle() {
        return subtitle;
    }

    public boolean isEmpty() {
        //Verify there is a subtitle
        if(!subtitle.isEmpty()) {
            return false;
        }
        //Verify a title was entered from the dialog and
        //Page.title or Page.PageTitle are not being used
        String uniqueTitle = coreTitle.getText();
        if (!uniqueTitle.equals(currentPage.getTitle())
            && !uniqueTitle.equals(currentPage.getPageTitle())) {
            return false;
        }
        return true;
    }
   @Override
    public String getExportedType() {
        return request.getResource().getResourceType();
    }

}
```

In order to develop against the core components Sling Models, we need to add a dependency to the core components bundle. Open **training** > **core** > **pom.xml**. At the bottom of the pom, verify the dependency exists. If this does not exist, add the dependency, as shown:

Note: You can get this dependency to copy and paste from the comments in the TitleWithSubtitle.java class:

```
<dependency>
   <groupId>com.adobe.cq</groupId>
   <artifactId>core.wcm.components.core</artifactId>
</dependency>
```

```
<!-- Dependency for TitleWithSubtitle.java -->
<dependency>
   <groupId>com.adobe.cq</groupId>
   <artifactId>core.wcm.components.core</artifactId>
</dependency>
```

10. Save your changes.
11. Open a command prompt to the location of your Maven project. For example: **C:/adobe/< myproject >**

Note: If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

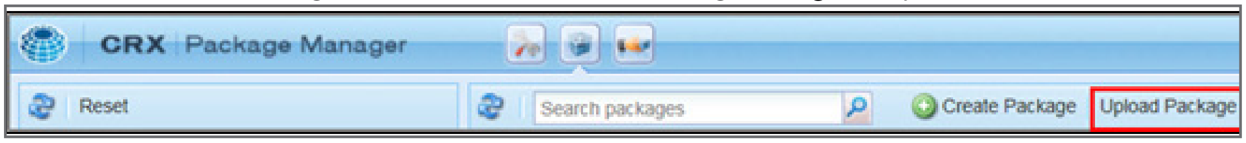
12. In the command prompt run the command:

```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

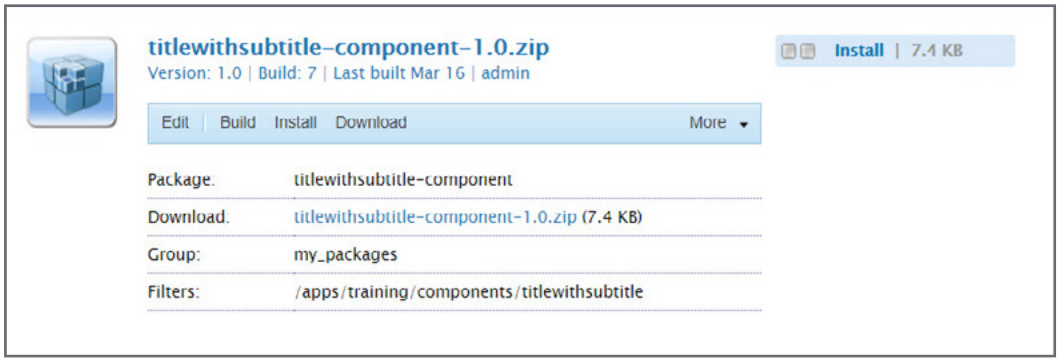
Your project has now successfully been installed into your local AEM Server.

Task 2: Install the content package

1. From **CRXDE Lite**, click the **Package** icon. The CRX Package Manager page is displayed.
2. Click **Upload Package**, as shown. The **Upload Package** dialog box opens.



3. In the **Upload Package** dialog box, click **Browse**, and select the **titlewithsubtitle-component-1.0.zip** package file from the **Exercise_Files-EC** under **/training-files/Sling-Models/**. Click **Open**.
4. Click **OK**.
5. Verify the uploaded package is now available in CRX Package Manager, as shown:


titlewithsubtitle-component-1.0.zip
Version: 1.0 | Build: 7 | Last built Mar 16 | admin
Install | 7.4 KB
Edit
Build
Install
Download
More

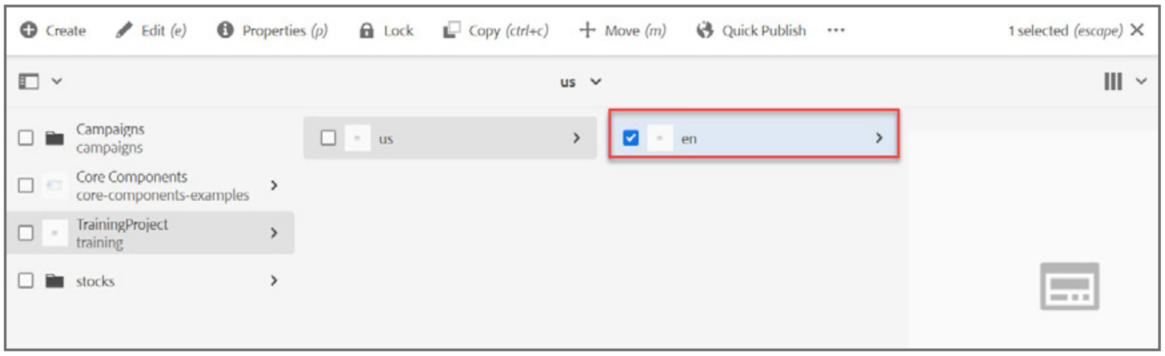
Package:	titlewithsubtitle-component
Download:	titlewithsubtitle-component-1.0.zip (7.4 KB)
Group:	my_packages
Filters:	/apps/training/components/titlewithsubtitle

Click **Install**.

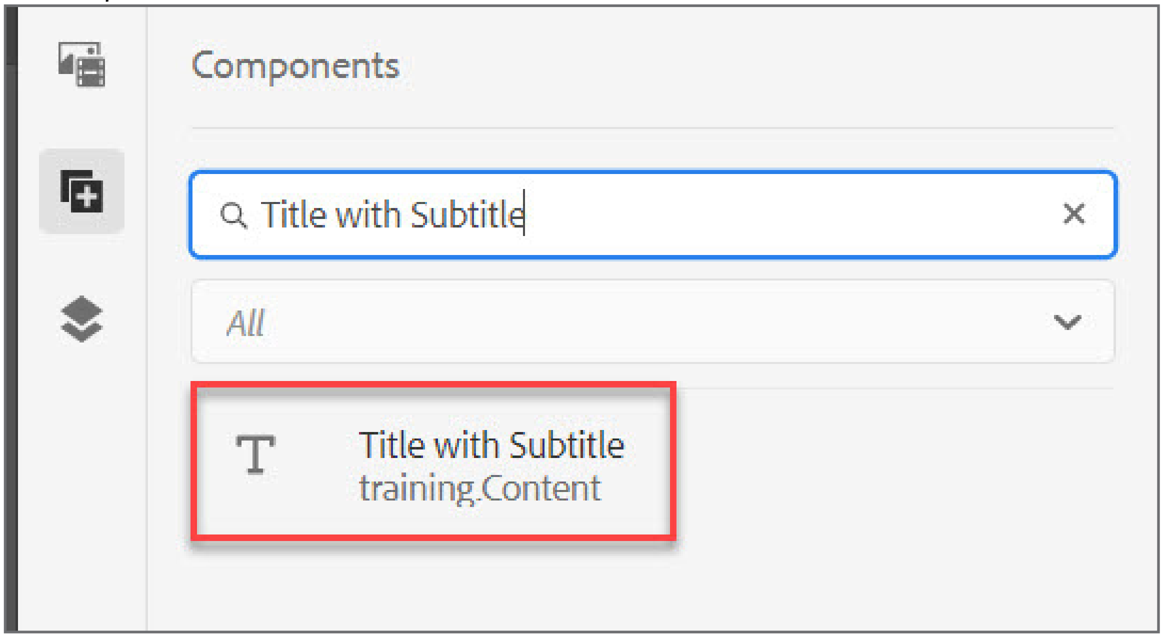
6. When the **Install Package** window appears, leave the **Advanced Settings** as-is and click **Install**. The package is installed.

Task 3: Test the component

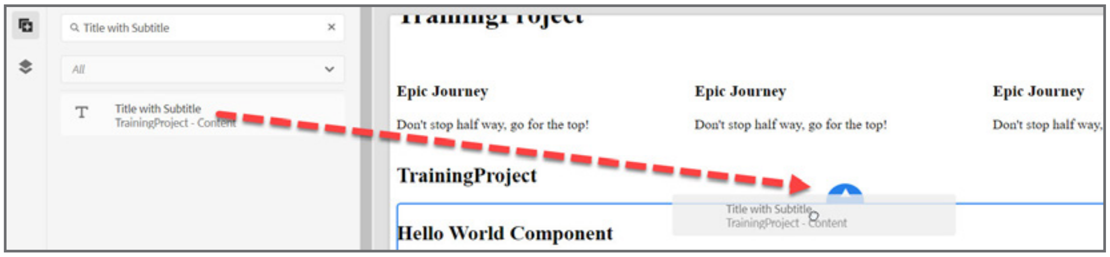
1. Go to the **Sites** Console, http://localhost:4502/sites.html/content.
2. Navigate to **Sites** > **TrainingProject** Site, and click the thumbnail on **English** and select the page, as shown:



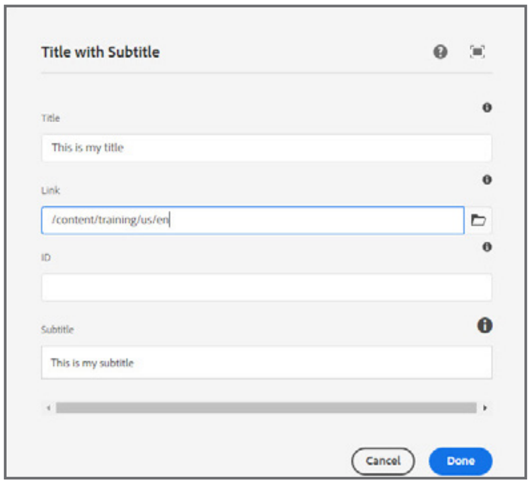
3. Click **Edit (*e*).** The page opens in a new tab for editing.
4. In the left pane, click the **Components** icon. Use the Filter option to find the **TitlewithSubtitle** component, as shown:



5. Click and drag the **Title with Subtitle** onto the Drag components here container, as shown:

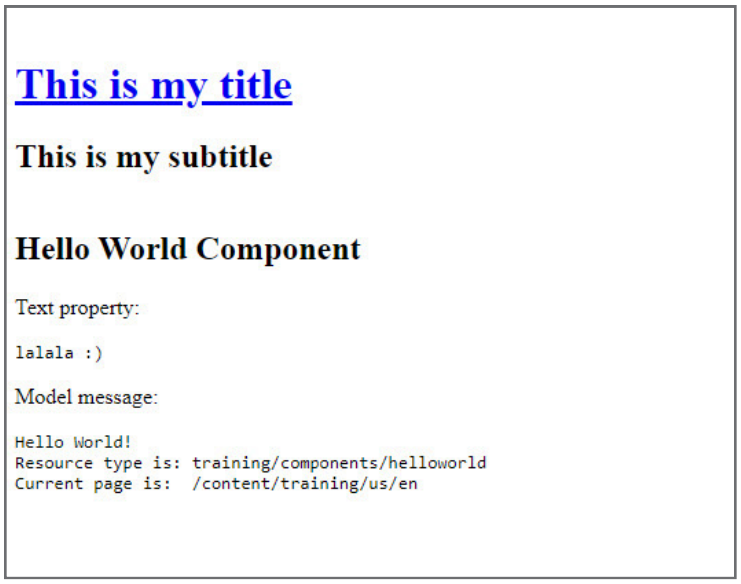


6. Double-click the **Title with Subtitle** components box to open it.
7. Add a **Title**, **Link**, and **Subtitle**, as shown:



Title: This is my title
Link: /content/training/us/en
ID:
Subtitle: This is my subtitle

8. Click **Done** to save the changes.
9. Verify the changes, as shown:



This is my title

This is my subtitle

Hello World Component

Text property:

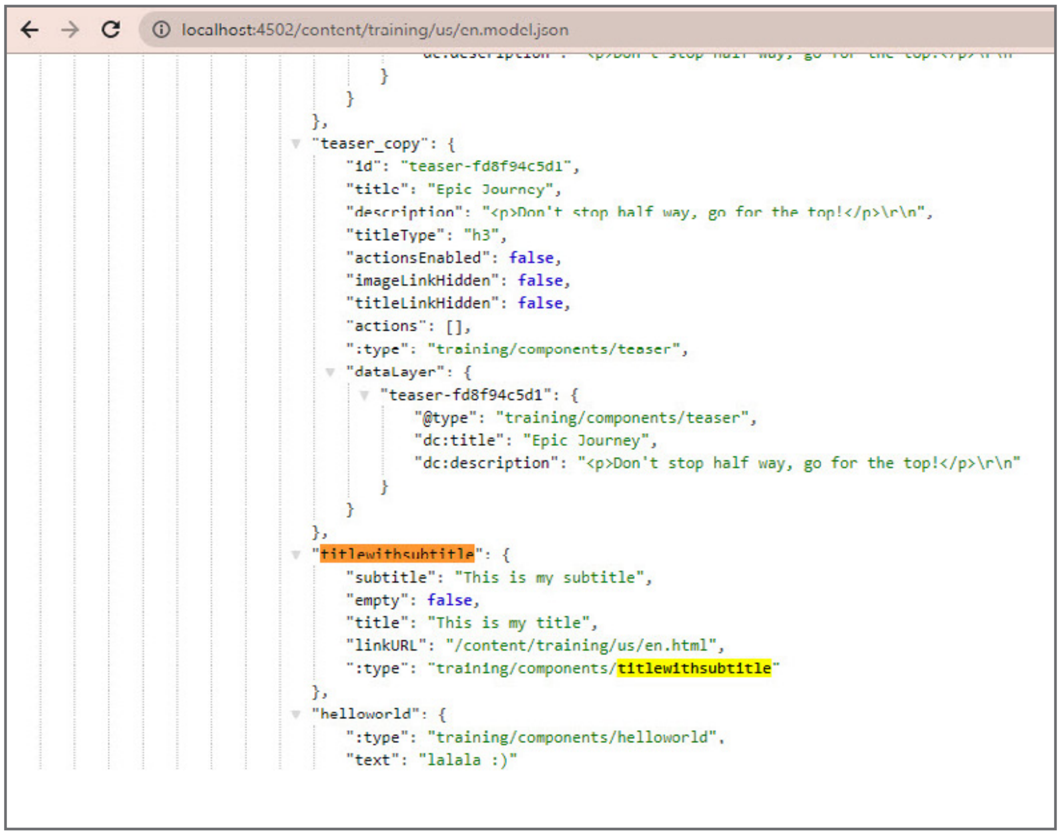
```
lalala :)
```

Model message:

```
Hello World!
Resource type is: training/components/helloworld
Current page is:  /content/training/us/en
```

10. Click the **Page information** icon > **View as Published**. The page gets published.
11. Verify the component exported as JSON by using the URL:
    http://localhost:4502/content/training/us/en.model.json

12. Search for **titlewithsubtitle** to find your component. Notice your model returned the model of the title component wrapped with the special values.



localhost:4502/content/training/us/en.model.json

```
                }
            }
        },
        "teaser_copy": {
            "id": "teaser-fd8f94c5d1",
            "title": "Epic Journey",
            "description": "<p>Don't stop half way, go for the top!</p>\r\n",
            "titleType": "h3",
            "actionsEnabled": false,
            "imageLinkHidden": false,
            "titleLinkHidden": false,
            "actions": [],
            ":type": "training/components/teaser",
            "dataLayer": {
                "teaser-fd8f94c5d1": {
                    "@type": "training/components/teaser",
                    "dc:title": "Epic Journey",
                    "dc:description": "<p>Don't stop half way, go for the top!</p>\r\n"
                }
            }
        },
        "titlewithsubtitle": {
            "subtitle": "This is my subtitle",
            "empty": false,
            "title": "This is my title",
            "linkURL": "/content/training/us/en.html",
            ":type": "training/components/titlewithsubtitle"
        },
        "helloworld": {
            ":type": "training/components/helloworld",
            "text": "lalala :)"
```

Exercise 2: Create a custom Sling Model

In this exercise, you will upload and install a stockplex component. This will be the front-end HTL code for your component. You will then write a Sling Model to support the business logic.

This exercise includes the following tasks:

1. Create a Sling Model
2. Install via command line
3. Install the stockplex component
4. Test the component

Task 1: Create a Sling Model

1. In your IDE, navigate to **core** > **src** > **main/java/com/adobe/training/core/**.
2. Right-click **models** and choose **New File**.
3. Rename the file as **Stockplex.java.**
4. Double-click **Stockplex.java** to edit the file.
5. Copy the contents from **Exercise_Files-EC** under **/core/src/main/java/com/adobe/training/core/ Stockplex.java** to **Stockplex.java** in your IDE, as shown:

```
package com.adobe.training.core.models;
import java.util.HashMap;
import java.util.Map;
import javax.annotation.PostConstruct;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ValueMap;
import org.apache.sling.models.annotations.DefaultInjectionStrategy;
import org.apache.sling.models.annotations.Exporter;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.ResourcePath;
import org.apache.sling.models.annotations.injectorspecific.ScriptVariable;
import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
import com.adobe.cq.export.json.ComponentExporter;
import com.adobe.training.core.StockDataWriterJob;
import com.day.cq.wcm.api.designer.Style;

/**
```

```
 * This model is used as the backend logic for the stockplex component. Using a Sling model allows the component
 * to be exportable via JSON for headless scenarios. Stock data that this model uses is imported into the JCR
 * via StockImportScheduler.java
 *
 * The stock data that is expected is in the form:
 * /content/stocks
 * + ADBE [sling:OrderedFolder]
 *   + lastTrade [nt:unstructured]
 *     - companyName = <value>
 *     - sector = <value>
 *     - lastTrade = <value
 *     - ..
 *     LK, updated for GITHUB data, 190710
 */

@Model(adaptables=SlingHttpServletRequest.class,
       adapters= {ComponentExporter.class},
       defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL,
       resourceType = Stockplex.RESOURCE_TYPE)
@Exporter(name="jackson", extensions = "json")
public class Stockplex implements ComponentExporter{
       protected static final String RESOURCE_TYPE = "training/components/stockplex";
       //HTL global object in the model
    //Learn more  at Helpx > HTL Global Objects
    @ScriptVariable
    private Resource resource;

    @ScriptVariable
    private Style currentStyle;

    //property on the current resource saved from the dialog of a component
    @ValueMapValue
    private String symbol;

    //property on the current resource saved from the dialog of a component
    @ValueMapValue
    private String summary;

    //property on the current resource saved from the dialog of a component
    @ValueMapValue
    private String showStockDetails;

    //content root of for stock data. /content/stocks
    @ResourcePath(path = StockDataWriterJob.STOCK_IMPORT_FOLDER)
    private Resource stocksRoot;

    private double currentPrice;
    private Map<String,Object> data;

    @PostConstruct
    public void constructDataMap() {
       ValueMap tradeValues = null;
```

```
        //Check to see if stock data has been imported into the JCR
        if(stocksRoot != null) {
            Resource stockResource = stocksRoot.getChild(symbol);
            if(stockResource != null) {
                Resource lastTradeResource = stockResource.getChild("trade");
                if(lastTradeResource != null){
                    tradeValues = lastTradeResource.getValueMap();
                }
            }
        }

        data = new HashMap<>();
        //If stock information is in the JCR, display the data
        if(tradeValues != null) {
            currentPrice = tradeValues.get(StockDataWriterJob.LASTTRADE, Double.class);
            data.put("Request Date", tradeValues.get(StockDataWriterJob.DAYOFUPDATE, String.class));
            data.put("Request Time", tradeValues.get(StockDataWriterJob.UPDATETIME, String.class));
            data.put("UpDown", tradeValues.get(StockDataWriterJob.UPDOWN, Double.class));
            data.put("Open Price", tradeValues.get(StockDataWriterJob.OPENPRICE, Double.class));
            data.put("Range High", tradeValues.get(StockDataWriterJob.RANGEHIGH, Double.class));
            data.put("Range Low", tradeValues.get(StockDataWriterJob.RANGELOW, Double.class));
            data.put("Volume",  tradeValues.get(StockDataWriterJob.VOLUME, Integer.class));
            data.put("Company", tradeValues.get(StockDataWriterJob.COMPANY, String.class));
            data.put("Sector", tradeValues.get(StockDataWriterJob.SECTOR, String.class));
            data.put("52 Week Low", tradeValues.get(StockDataWriterJob.WEEK52LOW, Double.class));
        } else {
            data.put(symbol,"No import config found. If the StockListener.
java class is apart of your project: Go to Sites console > Create Folder: stocks > Create Folder: ADBE");
        }
    }

    /**
     * All getter methods below will be a part of the output by the JSON Exporter
     */
    //Getter for dialog input
    public String getSymbol() {
        return symbol;
    }
    //Getter for dialog input
    public String getSummary() {
        return summary;
    }
    //Getter for dialog input
    public String getShowStockDetails() {
        return showStockDetails;
    }

    //Calculated current price based on imported stock info
    public Double getCurrentPrice() {
        return currentPrice;
    }
    //Calculated trade values based on imported stock info
    public Map<String,Object> getData() {
        return data;
    }
    @Override
    public String getExportedType() {
        return resource.getResourceType();
    }
}
```

6. Examine the above code. Notice it is using @Model annotation and it is also exportable as JSON.
7. Save the changes.

Task 2: Install via command line

1. Open a command prompt to the location of your Maven project. For example: C:/adobe/< myproject >

Note: If you are using an IDE with an integrated terminal such as Visual Studio Code, you can run your Maven commands there instead since it is already open to your project.

---

2. In the command prompt run the command:

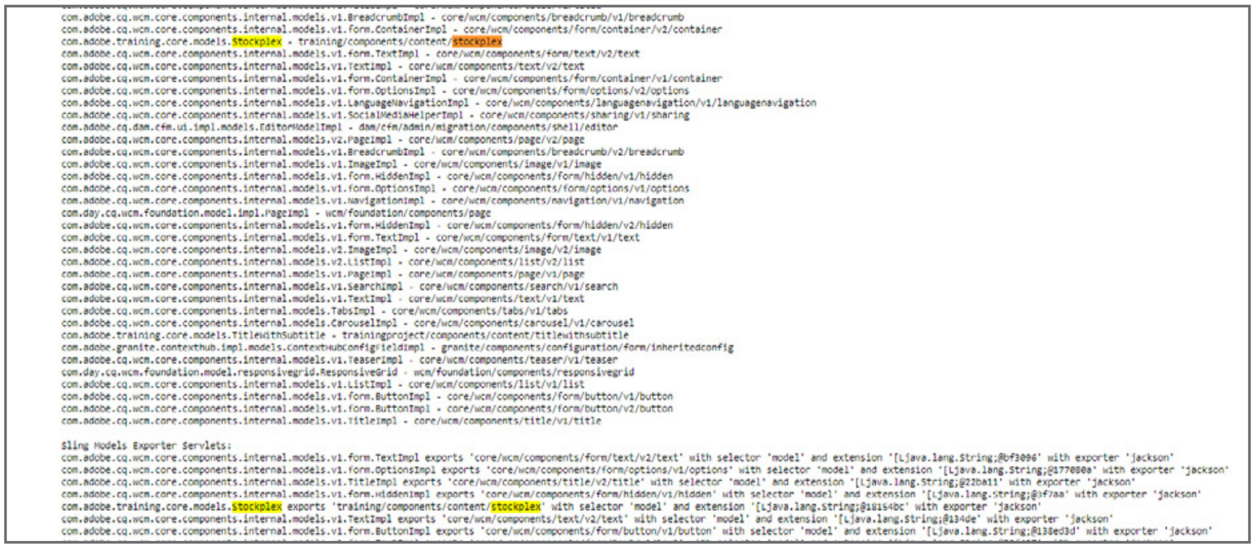
```
$ mvn clean install -Padobe-public -PautoInstallSinglePackage
```

Your project has now successfully installed into your local AEM Server.

To observe the Sling Model that you installed:

3. In a browser, navigate to AEM **Web Console**. Select **Status** > **Sling Models** and search for **stockplex**.

You should notice the **stockplex** model and also the **stockplex** component are exportable as JSON.



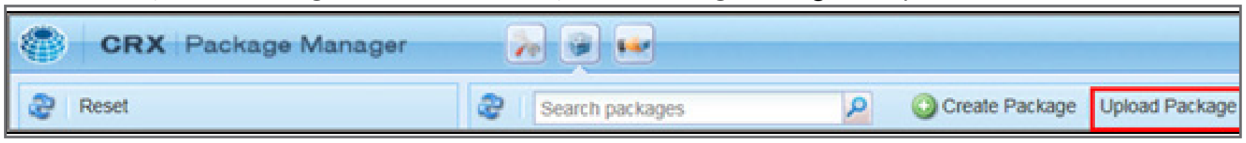
```
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl - core/wcm/components/breadcrumb/v1/breadcrumb
com.adobe.cq.wcm.core.components.internal.models.v1.form.ContainerImpl - core/wcm/components/form/container/v2/container
com.adobe.training.core.models.Stockplex - training/components/content/stockplex
com.adobe.cq.wcm.core.components.internal.models.v1.form.TextImpl - core/wcm/components/form/text/v2/text
com.adobe.cq.wcm.core.components.internal.models.v1.TextImpl - core/wcm/components/text/v2/text
com.adobe.cq.wcm.core.components.internal.models.v1.form.ContainerImpl - core/wcm/components/form/container/v1/container
com.adobe.cq.wcm.core.components.internal.models.v1.form.OptionsImpl - core/wcm/components/form/options/v2/options
com.adobe.cq.wcm.core.components.internal.models.v1.LanguageNavigationImpl - core/wcm/components/languagenavigation/v1/languagenavigation
com.adobe.cq.wcm.core.components.internal.models.v1.SocialMediaHelperImpl - core/wcm/components/sharing/v1/sharing
com.adobe.cq.dam.cfm.ui.impl.models.EditorModelImpl - dam/cfm/admin/migration/components/shell/editor
com.adobe.cq.wcm.core.components.internal.models.v2.PageImpl - core/wcm/components/page/v2/page
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl - core/wcm/components/breadcrumb/v2/breadcrumb
com.adobe.cq.wcm.core.components.internal.models.v1.ImageImpl - core/wcm/components/image/v1/image
com.adobe.cq.wcm.core.components.internal.models.v1.form.HiddenImpl - core/wcm/components/form/hidden/v1/hidden
com.adobe.cq.wcm.core.components.internal.models.v1.form.OptionsImpl - core/wcm/components/form/options/v1/options
com.adobe.cq.wcm.core.components.internal.models.v1.NavigationImpl - core/wcm/components/navigation/v1/navigation
com.day.cq.wcm.foundation.model.impl.PageImpl - wcm/foundation/components/page
com.adobe.cq.wcm.core.components.internal.models.v1.form.HiddenImpl - core/wcm/components/form/hidden/v2/hidden
com.adobe.cq.wcm.core.components.internal.models.v1.form.TextImpl - core/wcm/components/form/text/v1/text
com.adobe.cq.wcm.core.components.internal.models.v2.ImageImpl - core/wcm/components/image/v2/image
com.adobe.cq.wcm.core.components.internal.models.v2.ListImpl - core/wcm/components/list/v2/list
com.adobe.cq.wcm.core.components.internal.models.v1.PageImpl - core/wcm/components/page/v1/page
com.adobe.cq.wcm.core.components.internal.models.v1.SearchImpl - core/wcm/components/search/v1/search
com.adobe.cq.wcm.core.components.internal.models.v1.TextImpl - core/wcm/components/text/v1/text
com.adobe.cq.wcm.core.components.internal.models.TabsImpl - core/wcm/components/tabs/v1/tabs
com.adobe.cq.wcm.core.components.internal.models.CarouselImpl - core/wcm/components/carousel/v1/carousel
com.adobe.training.core.models.TitleWithSubtitle - trainingproject/components/content/titlewithsubtitle
com.adobe.granite.contexthub.impl.models.ContextHubConfigFieldImpl - granite/components/configuration/form/inheritedconfig
com.adobe.cq.wcm.core.components.internal.models.v1.TeaserImpl - core/wcm/components/teaser/v1/teaser
com.day.cq.wcm.foundation.model.responsivegrid.ResponsiveGrid - wcm/foundation/components/responsivegrid
com.adobe.cq.wcm.core.components.internal.models.v1.ListImpl - core/wcm/components/list/v1/list
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl - core/wcm/components/form/button/v1/button
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl - core/wcm/components/form/button/v2/button
com.adobe.cq.wcm.core.components.internal.models.v1.TitleImpl - core/wcm/components/title/v1/title

Sling Models Exporter Servlets:
com.adobe.cq.wcm.core.components.internal.models.v1.form.TextImpl exports 'core/wcm/components/form/text/v2/text' with selector 'model' and extension '[Ljava.lang.String;@bf3096' with exporter 'jackson'
com.adobe.cq.wcm.core.components.internal.models.v1.form.OptionsImpl exports 'core/wcm/components/form/options/v1/options' with selector 'model' and extension '[Ljava.lang.String;@177080a' with exporter 'jackson'
com.adobe.cq.wcm.core.components.internal.models.v1.TitleImpl exports 'core/wcm/components/title/v2/title' with selector 'model' and extension '[Ljava.lang.String;@22ba11' with exporter 'jackson'
com.adobe.cq.wcm.core.components.internal.models.v1.form.HiddenImpl exports 'core/wcm/components/form/hidden/v1/hidden' with selector 'model' and extension '[Ljava.lang.String;@3f7aa' with exporter 'jackson'
com.adobe.training.core.models.stockplex exports 'training/components/content/stockplex' with selector 'model' and extension '[Ljava.lang.String;@18154bc' with exporter 'jackson'
com.adobe.cq.wcm.core.components.internal.models.v1.TextImpl exports 'core/wcm/components/text/v2/text' with selector 'model' and extension '[Ljava.lang.String;@134de' with exporter 'jackson'
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl exports 'core/wcm/components/form/button/v1/button' with selector 'model' and extension '[Ljava.lang.String;@138ed3d' with exporter 'jackson'
```

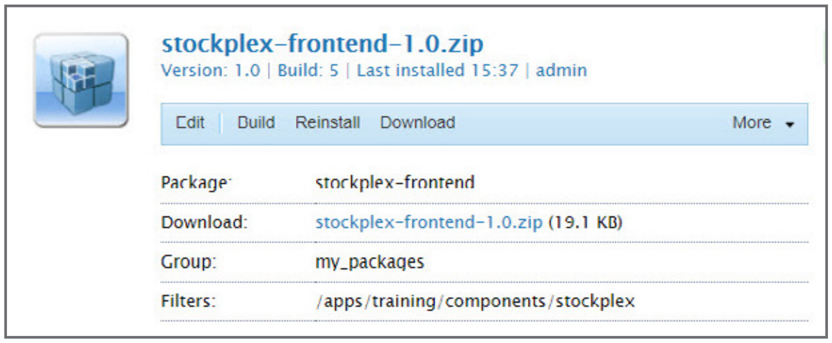
Task 3: Install the stockplex component

Now that we have the Sling model installed, we are going to install the component that uses the Sling model, the **stockplex** component. This component is built out in detail in the "Develop Websites and Components in Adobe Experience Manager" course.

1. From **CRXDE Lite**, click the **Package** icon. The CRX Package Manager page is displayed.
2. Click **Upload Package**, as shown. The **Upload Package** dialog box opens.



3. In the Upload Package dialog box, click **Browse**, and select the **stockplex-frontend.zip** package file from **Exercise_Files-EC** under **/training-files/Sling-Models**. Click **Open**.
4. Click **OK**.
5. Verify the uploaded package is now available in **CRX Package Manager**.
6. Click **Install**.
7. When the **Install Package** window opens, leave **Advanced Settings** as-is and click Install. The package is installed.
8. Click the Develop icon at the top of CRXDE Lite. Verify the package is now installed under /**apps/training/components/stockplex:**

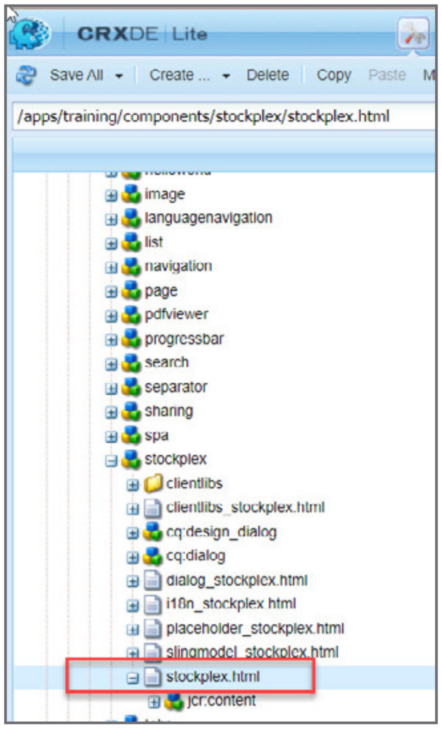


stockplex-frontend-1.0.zip
Version: 1.0 | Build: 5 | Last installed 15:37 | admin

Package:	stockplex-frontend
Download:	stockplex-frontend-1.0.zip (19.1 KB)
Group:	my_packages
Filters:	/apps/training/components/stockplex

9. In **CRXDE Lite**, navigate to **/apps/training/components/stockplex**.

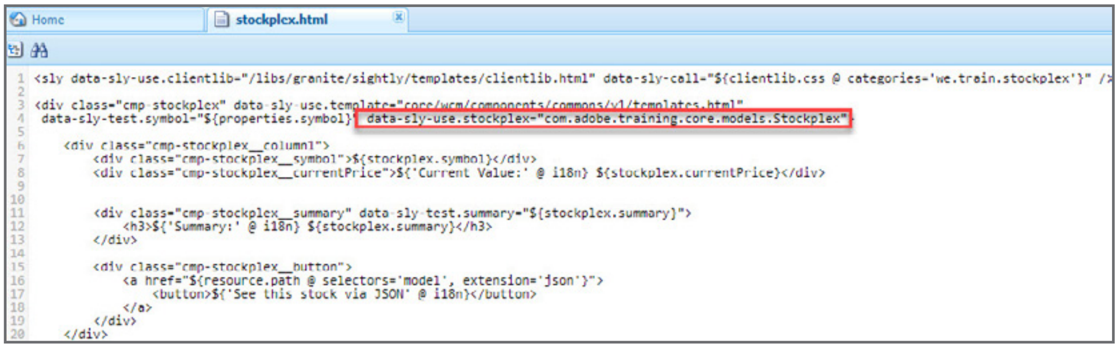
10. Click the **+** icon to expand the **stockplex** node and double-click **stockplex.html** to open it in the editor.



/apps/training/components/stockplex/stockplex.html

```
image
languagenavigation
list
navigation
page
pdfviewer
progressbar
search
separator
sharing
spa
stockplex
    clientlibs
    clientlibs_stockplex.html
    cq:design_dialog
    cq:dialog
    dialog_stockplex.html
    i18n_stockplex.html
    placeholder_stockplex.html
    slingmodel_stockplex.html
    stockplex.html
        jcr:content
```

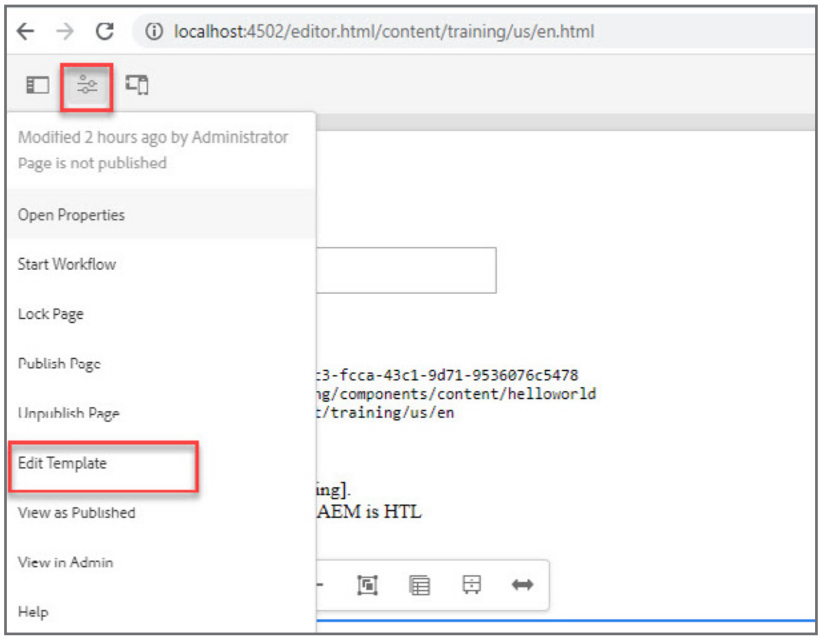
11. In the script, notice how it is requesting your Sling model stockplex on line 4, as shown:



```
1  <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-call="${clientlib.css @ categories='we.train.stockplex'}" />
2
3  <div class="cmp-stockplex" data-sly-use.template="core/wcm/components/commons/v1/templates.html"
4   data-sly-test.symbol="${properties.symbol}" data-sly-use.stockplex="com.adobe.training.core.models.Stockplex">
5
6      <div class="cmp-stockplex__column1">
7          <div class="cmp-stockplex__symbol">${stockplex.symbol}</div>
8          <div class="cmp-stockplex__currentPrice">${'Current Value:' @ i18n} ${stockplex.currentPrice}</div>
9
10
11         <div class="cmp-stockplex__summary" data-sly-test.summary="${stockplex.summary}">
12             <h3>${'Summary:' @ i18n} ${stockplex.summary}</h3>
13         </div>
14
15         <div class="cmp-stockplex__button">
16             <a href="${resource.path @ selectors='model', extension='json'}">
17                 <button>${'See this stock via JSON' @ i18n}</button>
18             </a>
19         </div>
20     </div>
```

Task 4: Test the stockplex component

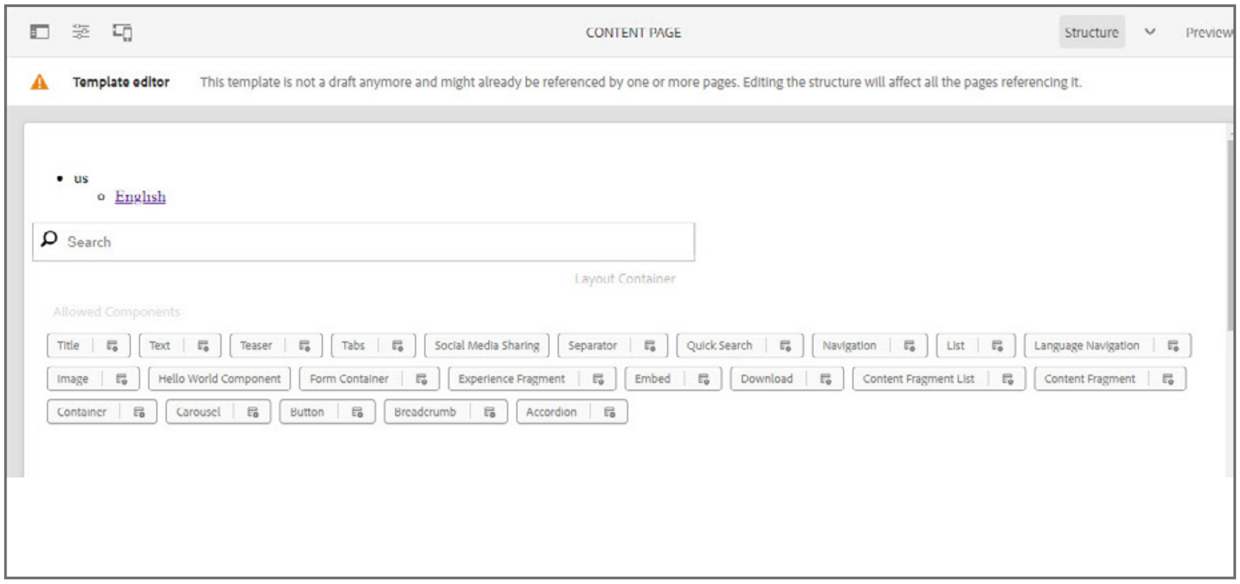
1. Go to: http://localhost:4502/editor.html/content/training/us/en.html
2. Click the **Page information** icon > **Edit Template**, as shown:



```
:3-fcca-43c1-9d71-9536076c5478
ng/components/content/helloworld
:/training/us/en
```

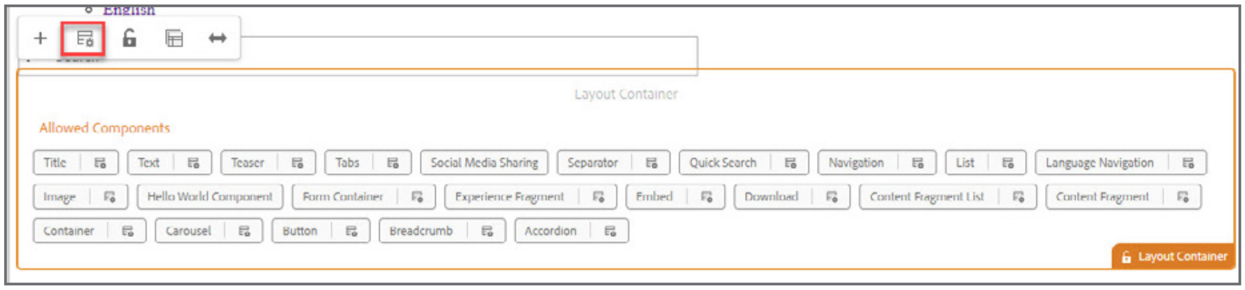
ing].
AEM is HTL

The **Template Editor** opens, as shown:



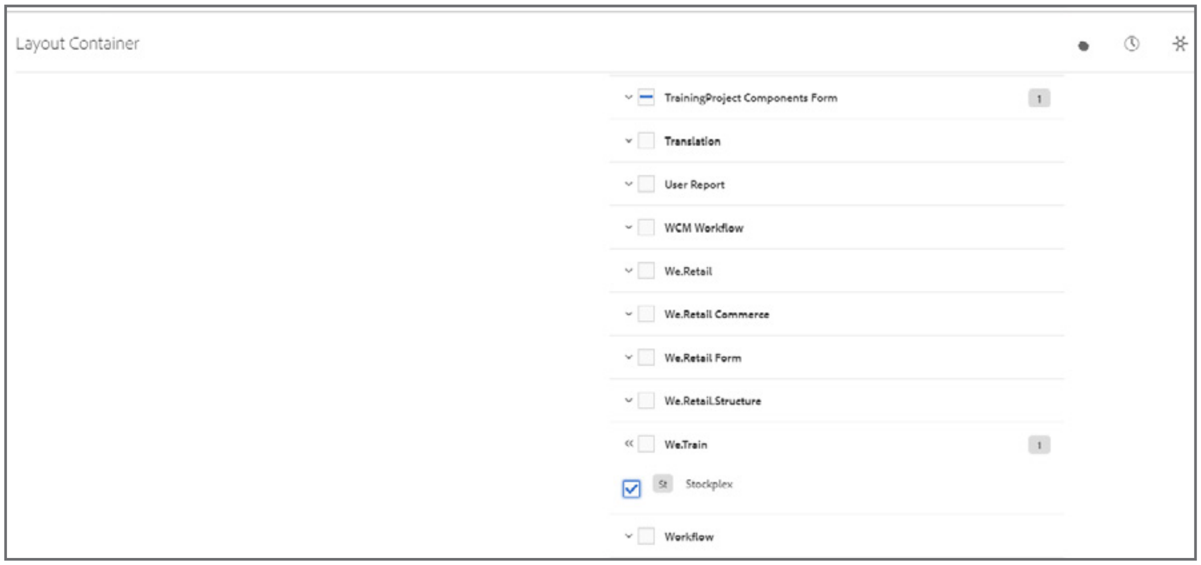
Template editor This template is not a draft anymore and might already be referenced by one or more pages. Editing the structure will affect all the pages referencing it.

3. Select the **Layout Container** and click on the **Policy** icon, as shown:



The Layout Container window opens.

4. Under the **Allowed Components** tab, select **We.train** > **Stockplex**, as shown:

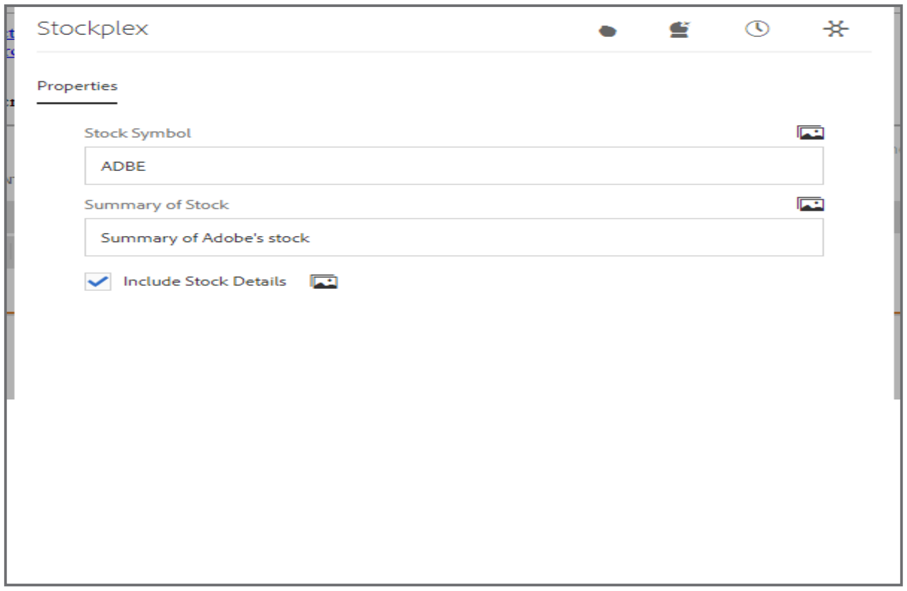


5. Click **Done** The changes are saved.
6. Navigate to http://localhost:4502/editor.html/content/training/us/en.html
7. Click **Toggle Side Panel** (if not already done).
8. In the left pane, click the **Components** icon. Use the Filter option to find **Stockplex** component. If you cannot see it, refresh the page by pressing **Ctrl+Shift+R**.
9. Drag and drop the **Stockplex** component onto your page.

10. Double-click the **Stockplex** components box to open it.

11. Configure the **Properties**, as shown:

    a. Stock Symbol: **ADBE**
    b. Summary of Stock: **Summary of Adobe's stock**
    c. Select **Include Stock Details**



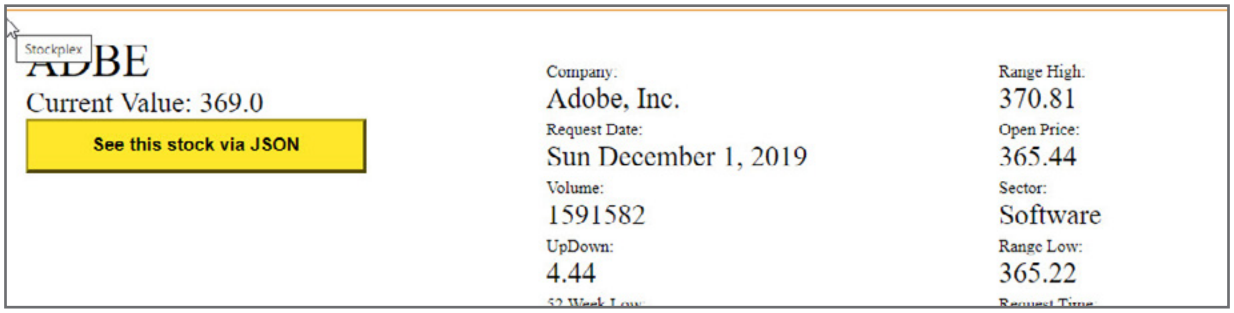
Stock Symbol: ADBE

Summary of Stock: Summary of Adobe's stock

✓ Include Stock Details

12. Click **Done** to save your changes.

13. Refresh the page. Notice the Sling Models goes into the JCR, finds the ADBE stock information that we imported earlier, and displays it onto the component, as shown:

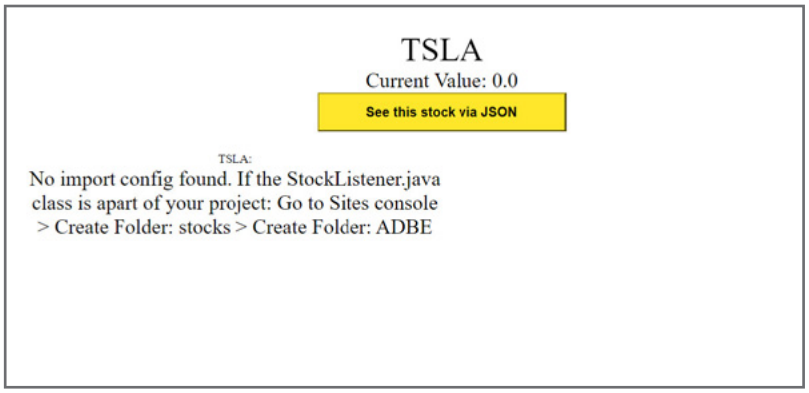


ADBE
Current Value: 369.0

Company: Adobe, Inc.
Request Date: Sun December 1, 2019
Volume: 1591582
UpDown: 4.44
52 Week Low:

Range High: 370.81
Open Price: 365.44
Sector: Software
Range Low: 365.22
Request Time:

14. Drag and drop another **Stockplex** component onto the page.
15. Configure the stockplex component to add a stock that is not in the JCR. For example, TSLA.
16. Click **Done** to save your changes.
17. Refresh the page. Notice the stock data is not imported since the stock data is not stored in the JCR, as shown:


TSLA
Current Value: 0.0

TSLA:
No import config found. If the StockListener.java class is apart of your project: Go to Sites console > Create Folder: stocks > Create Folder: ADBE

Remember! In this exercise, you uploaded the **stockplex** component via a content package, it is not a part of your Maven project. If this content should be part of your code base, it needs to be synchronized back to the Maven project. The content to import from AEM: /**apps/training/components/stockplex.**

- In **ui.apps**, update the **filter.xml** with:
  <filter root="/apps/training/components/ stockplex"/>
- In your IDE, right click **ui.apps** > **Import from the AEM Server**.