**Creating a WebSphere Commerce web service module (SOI)**

Here we are creating a WebSphere Commerce web service module (SOI) with WebSphere Commerce Developer. This tutorial contains two parts: creating a **Get TutorialStore web service**, and **creating a Process TutorialStore web service.**

1. **Creating the Get TutorialStore web service**

The services that are provided by the tutorial store are Get and Process services. In this part of the tutorial, the Get service is used to retrieve store information that is based on a search expression.

Since this tutorial uses SOI web services, the pattern creates a service module that provides service-oriented integration into your existing controller commands, access beans, and EJBs.

**Learning objectives**

The tutorial demonstrates how to support the following XPath expressions.

* Find store by store ID: /Store[StoreIdentifier[UniqueID='123']]
* Find store by name: /Store[StoreIdentifier[ExternalIdentifier[NameIdentifier='StoreName']]]
* Retrieve all stores: /Store

For this tutorial, the definition of the TutorialStore contains the information that is listed in the table below. The table shows the access profile that includes this information for the preceding XPath expressions.

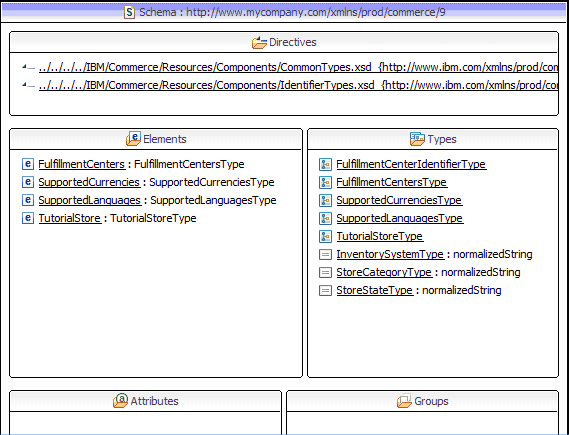
| **Data** | **Description** | **Applicable Access Profiles** |
| --- | --- | --- |
| Store identifier information | Information to uniquely identify the store. | Summary, Detail, All |
| Store description | Description information about the store. | Detail, All |
| State | Whether the store is "Open", "Closed" or "Suspended." | Detail, All |
| Store category | The category of the store, such as B2C or B2B. | Detail, All |
| Inventory system | The inventory system that is used by the store. | Detail, All |
| Store relationship | The store directory that contains the file resources that are associated with the store. | Detail, All |
| Supported languages | The languages that are supported by the store. | All |
| Support currencies | The currencies that are supported by the store. | All |
| Fulfillment center | The fulfillment center that is associated with the store. | All |

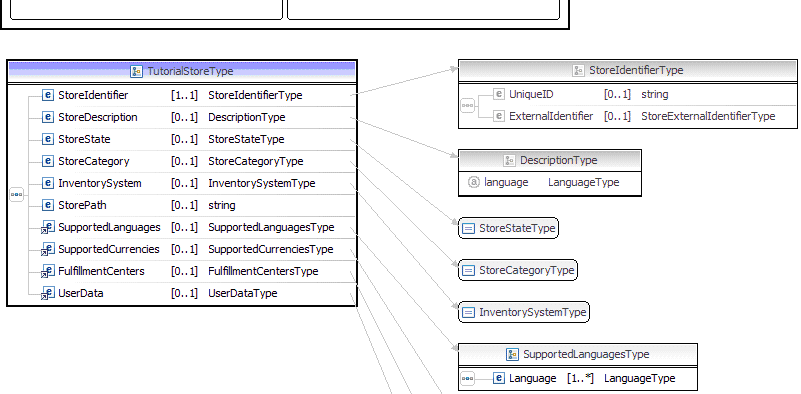
**Step 1: Defining the TutorialStore noun**

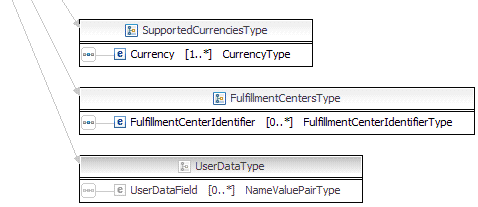
In this lesson, the TutorialStore noun is defined. This noun represents the logical data model of the new component that you are going to create. The definition of this logical model is a key development step, as this model is exposed to any client using the TutorialStore service, for example JUnit, and WebSphere Portal.

**Procedure**

1. Review the logical data model for the TutorialStore noun definition from the provided TutorialStore.xsd file:







1. After reviewing the logical data model for the TutorialStore noun definition from the provided TutorialStore.xsd file, note the following:

The diagrams illustrate the high-level structure of the TutorialStore logical data model defined in the TutorialStore.xsd file. It contains all type definitions that make up the TutorialStore noun, including:

**StoreIdentifier**

Store identification information, including the StoreID, Store Identifier, and Store owner.

**StoreDescription**

The store description.

**StoreState**

The store state, whether the store is open, closed, or suspended.

**StoreCategory**

The store category. For example: B2C, B2B, CPS.

**InventorySystem**

The inventory system. For example: ATP, Non-ATP, or none.

**StorePath**

The store directory that contains the file resources associated with the store.

**SupportedLanguages**

The languages supported by the store.

**SupportedCurrencies**

The currencies supported by the store.

**FulfillmentCenters**

The store fulfillment centers.

**UserData**

The user data area.

Step 2: **Generating the TutorialStore service module projects**

In this step, you are going to use the Java Emitter Template (JET) to generate the base code for your TutorialStore component. A significant amount of Java code is related to the processing of WebSphere Commerce nouns and can be abstracted and generated. This allows you to focus on implementing the code specific to each component, for example the component's nouns and the noun's business logic.

**Procedure**

1. [Start WebSphere Commerce Developer](http://www.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/tasks/tsrwcdev.htm?lang=en-us).
2. Open the Java EE perspective.
3. Create the application definition file to create the base code for the TutorialStore SOI service module.
   1. Create the service module input file.
      1. Right-click on the WebSphereCommerceServerExtensionsLogic project and select **New** > **Folder**.
      2. Enter ServiceModuleDefinition as the folder name. This creates a single location to store all service module input files.
      3. Right-click the ServiceModuleDefinition folder and select **New** > **File**.
      4. Enter SOITutorialStore.xml as the filename and click **Finish**.
   2. Paste the following application definition into the file:

<\_pattern:commerceComponent xmlns:\_pattern="http://www.ibm.com/xmlns/prod/commerce/foundation/pattern"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.ibm.com/xmlns/prod/commerce/foundation/pattern ../../WC/xml/config/xsd/wc-component-patten.xsd "

name="SOITutorialStore" packagenameprefix="com.mycompany.commerce" company="MyCompany"

namespace="http://www.mycompany.com/xmlns/prod/commerce/9/soitutorialstore" nlsprefix="myco" type="SOI">

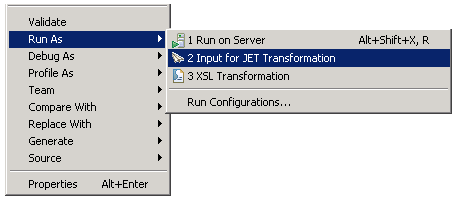
<\_pattern:noun name="TutorialStore" get="true" process="false" change="false" sync="false"/>

</\_pattern:commerceComponent>

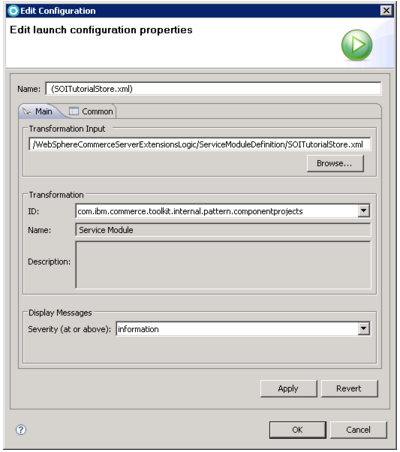
**Note:** The definition indicates the name of the noun (TutorialStore) and the verbs that are enabled for the pattern. This file determines the assets and code to be generated for you.

* 1. Save and close the file.

1. Right-click **SOITutorialStore.xml** and select **Run As** > **Input for JET Transformation**:



1. In the **Transformation** section:
2. Select the following ID:
   * com.ibm.commerce.toolkit.internal.pattern.componentprojects



1. Click Ok
2. The pattern is applied. Verify that the following projects are created, by switching to the Navigator view:

* SOITutorialStore-Client
* SOITutorialStore-DataObjects
* SOITutorialStore-Server
* SOITutorialStore-UnitTests
* SOITutorialStoreServicesHTTPInterface
* SOITutorialStoreServicesJMSInterface

1. Add the service module to the WebSphere Commerce application to update the build dependencies for the service module. Build dependencies are managed using the Java EE Module dependencies instead of the project classpath.
2. Switch to the Java EE perspective and expand the **WC** project.
3. Double-click **WebSphereCommerceServer**.
4. Select the **Design** tab.
5. Click **Add**. If **Add** cannot be selected, click **Application**, then click **Add**.
6. Select **Module** and click **OK**.
7. Select **SOITutorialStore-Server** and click **Finish**.
8. Click **Add**.
9. Select **Module** and click **OK**.
10. Select **SOITutorialStoreServicesHTTPInterface** and click **Finish**.
11. In the **Application Deployment Descriptor Editor**, under **Actions**, click **Manage Utility Jars**. If Manage Utility Jars does not appear close then reopen WebSphereCommerceServer.
12. Ensure the following values are selected
    * SOITutorialStore-Client
    * SOITutorialStore-DataObjects
13. Click **OK**.
14. Save the deployment descriptor.
15. Right-click the **SOITutorialStore-Server** project and select **Properties**.
16. Select **Java EE Module Dependencies**.
17. Ensure the following are selected:
    1. Enablement-RelationshipManagementData.jar from the WC project.
    2. Enablement-RelationshipManagementLogic.jar from the WC project.
    3. SOITutorialStore-Client from the SOITutorialStore-Client project.
    4. SOITutorialStore-DataObjects from the SOITutorialStore-DataObjects project.
18. Click **OK**.