# API Wrapper Update Guide

2011/4/18, By Chintan Desai, Solution Architect, Zuora Inc.

Key Contributors: - Lukasz Weber, Solution Architect, Zuora Inc.

## Overview

The following is a description on how to deploy, use, and modify APEX Z-Commerce API Wrappers in SFDC. The APEX Wrappers are nothing more than web services stub code that normally would be generated by a WSDL parser. However, due to advanced features (complex types) being used extensively in the Zuora WSDL, the SFDC WSDL parser cannot correctly generate the proper APEX stub classes. These APEX Wrappers provide object and action definitions that map one-to-one with the Zuora Z-Commerce API. Please note the following features and limitations:

* The wrappers are provided as sample code only, as is, with no support, maintenance, or warranty provisions.
* The wrappers described in this document support v28 of the Zuora WSDL only.
* Test coverage of the wrappers is currently known to be above 75%, but please plan to re-test the code during development and correct the coverage if necessary.

Lastly, this code has been written by Zuora Professional Services, and although it has been used by other customers in production, it is only meant as a short-term workaround.

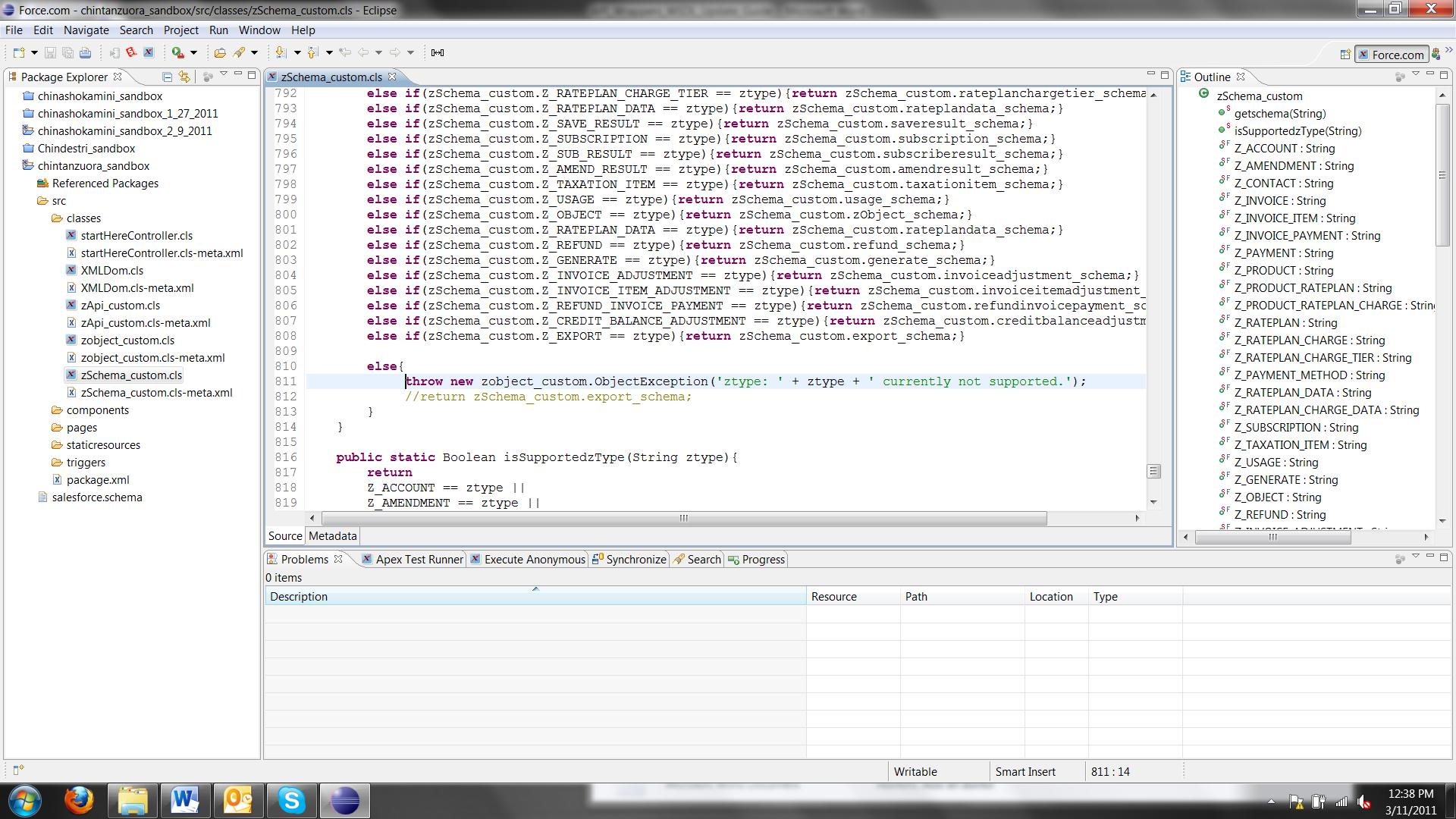
Zuora is planning to release a productized version of these wrappers, so please inquire at [support@zuora.com](mailto:support@zuora.com) about alternate integration options before using these classes.

## Deployment

### Import the wrapper classes.

The Eclipse IDE should be installed with the Force.com plug-in, before any of the wrapper classes can be imported.

1. Create a new Apex class and name it “zSchema\_custom.” Copy and paste the code from “zSchema\_custom.cls” file into this new class.
2. Create a new Apex class and name it “zobject\_custom.” Copy and paste the code from “zobject\_custom.cls” file into this new class.
3. Create a new Apex class and name it “zApi\_custom.” Copy and paste the code from “zApi\_custom.cls” file into this new class.
4. In the “zSchema\_custom” class (please see screenshot below):
   1. Comment out line 811: *throw new zobject\_custom.ObjectException(*…
   2. Uncomment line 812: *return zSchema\_custom.export\_schema;*

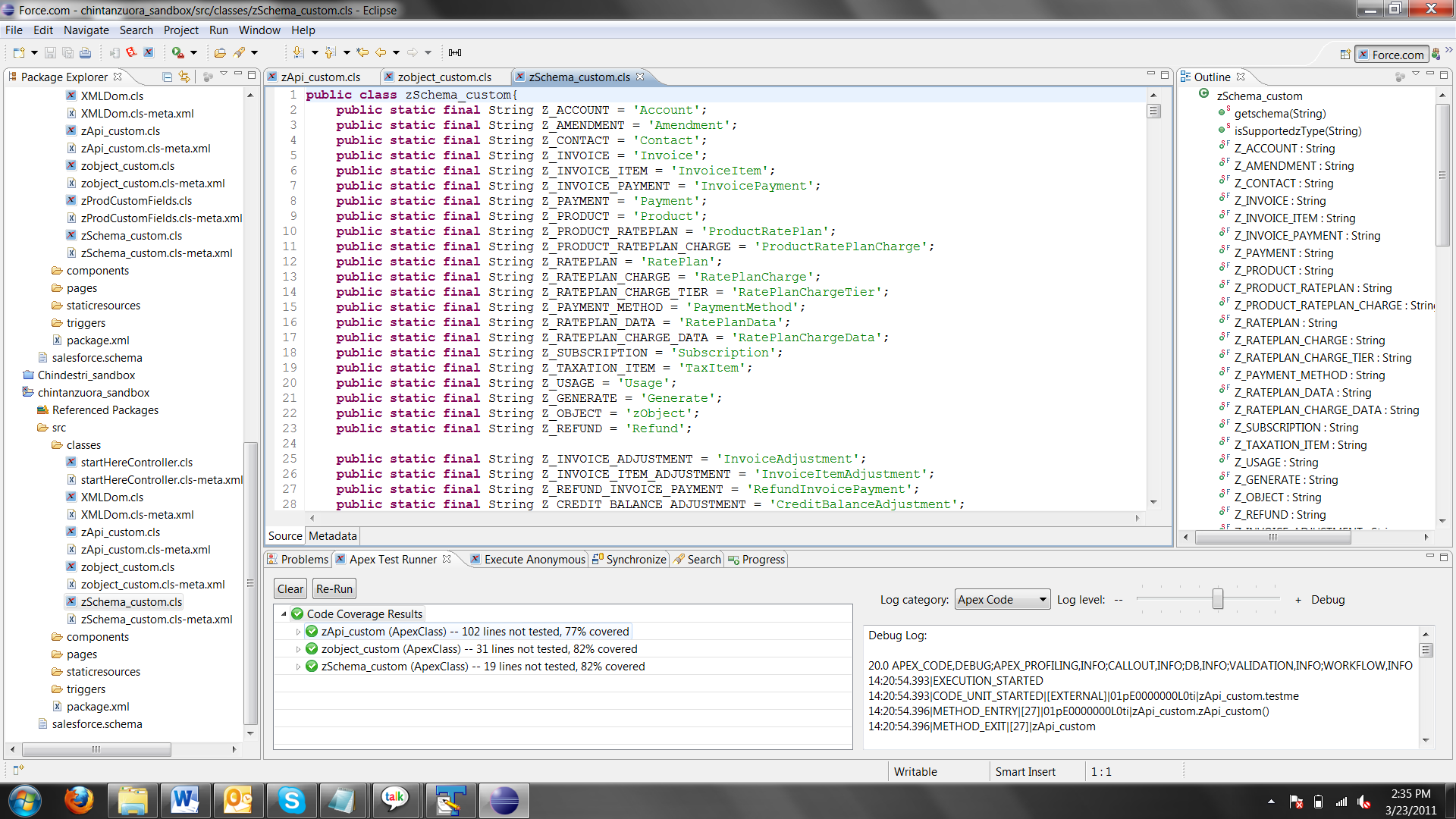


1. Compile the “zSchema\_custom” class.
2. In class “zobject\_custom” comment out the following lines
   1. Line # 59
   2. Line # 91
   3. Line #125 to Line#133
   4. Line #223 to Line#280
3. Compile the “zobject\_custom” class.
4. Compile the “zApi\_custom” class.
5. Now that “zApi\_custom” class is compiled, go ahead and reverse the actions taken in step 6, and re-build.
6. Now that “zobject\_custom” class is compiled, go ahead and reverse the actions taken in step 4, and re-build.

All three class files should now have successfully built and are ready to use.

## Test Coverage for the Apex Wrappers

Once you have successfully imported these three Apex classes; please do “Run Test” in Salesforce.com to make sure these three wrapper classes have enough test coverage to make sure they can be deployed in Salesforce.com production environment; Zuora engineering has covered these three classes to have sufficient test coverage (Equal or More than 75%) and as shown in the following screenshot; so please make sure you have verify this coverage.



## Maintenance

### Wrapper Updates

Any time there is a new Zuora WSDL version, the Apex wrappers can be updated to reflect the latest schema as well as any new operations that may have been added.

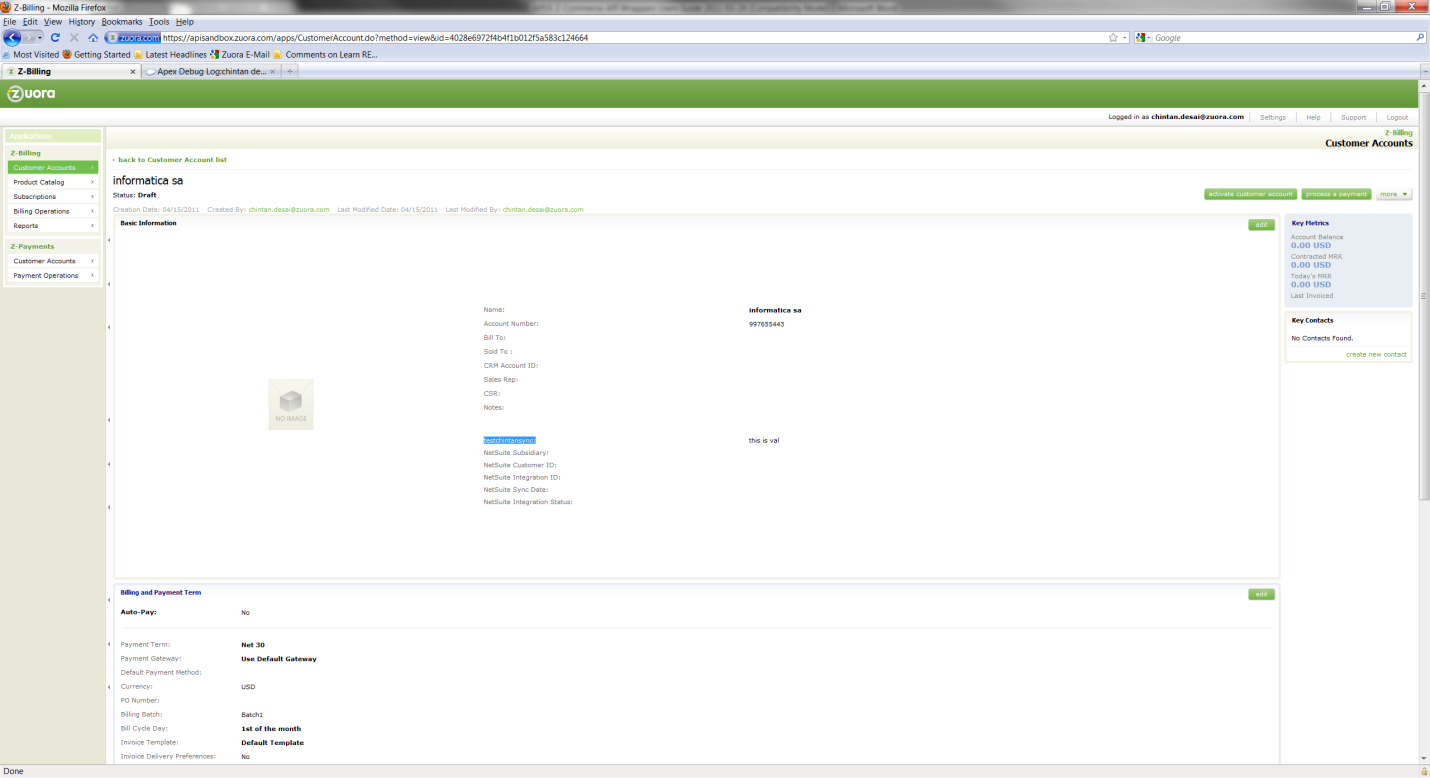
Here are the main wrapper classes that will need to be updated to reflect the changes in the WSDL:

* zSchema\_custom
* zobject\_custom
* zApi\_custom

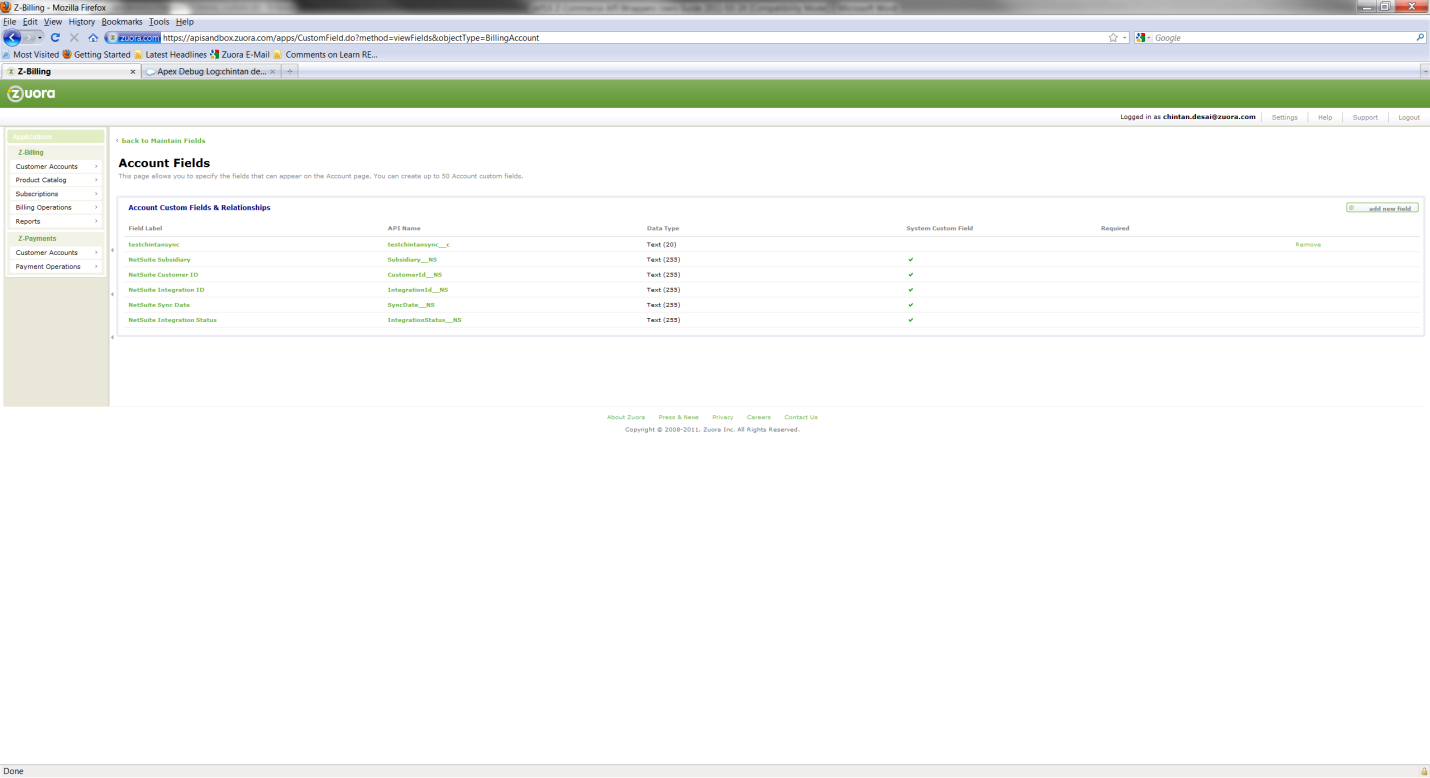
There are four types of changes that the new WSDL will incorporate, that will need require code changes to the wrappers:

1. New custom field has been added to Zuora object for the purpose of your project.

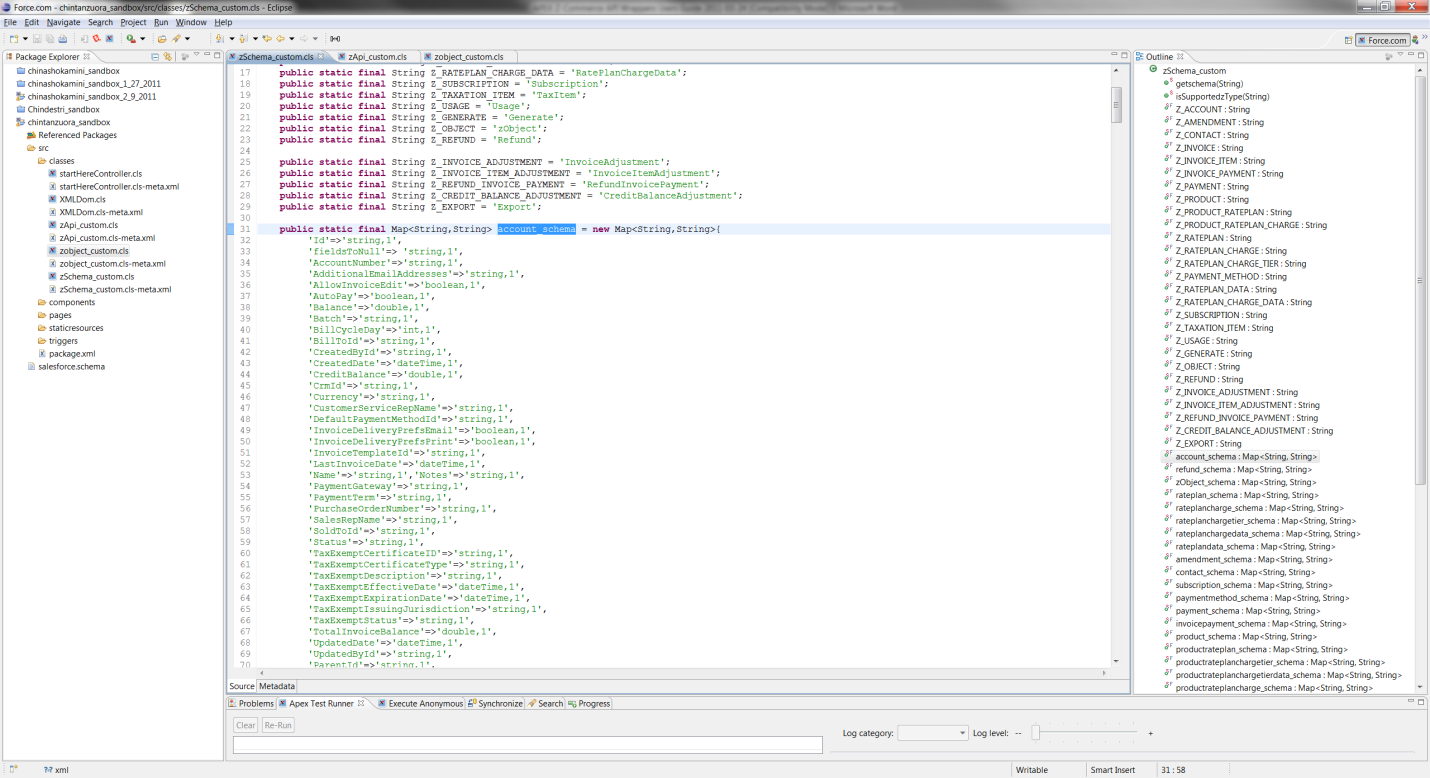
As shown below let’s assume you have added a custom field on “Account” object in Zuora with the name “testchintansync”.



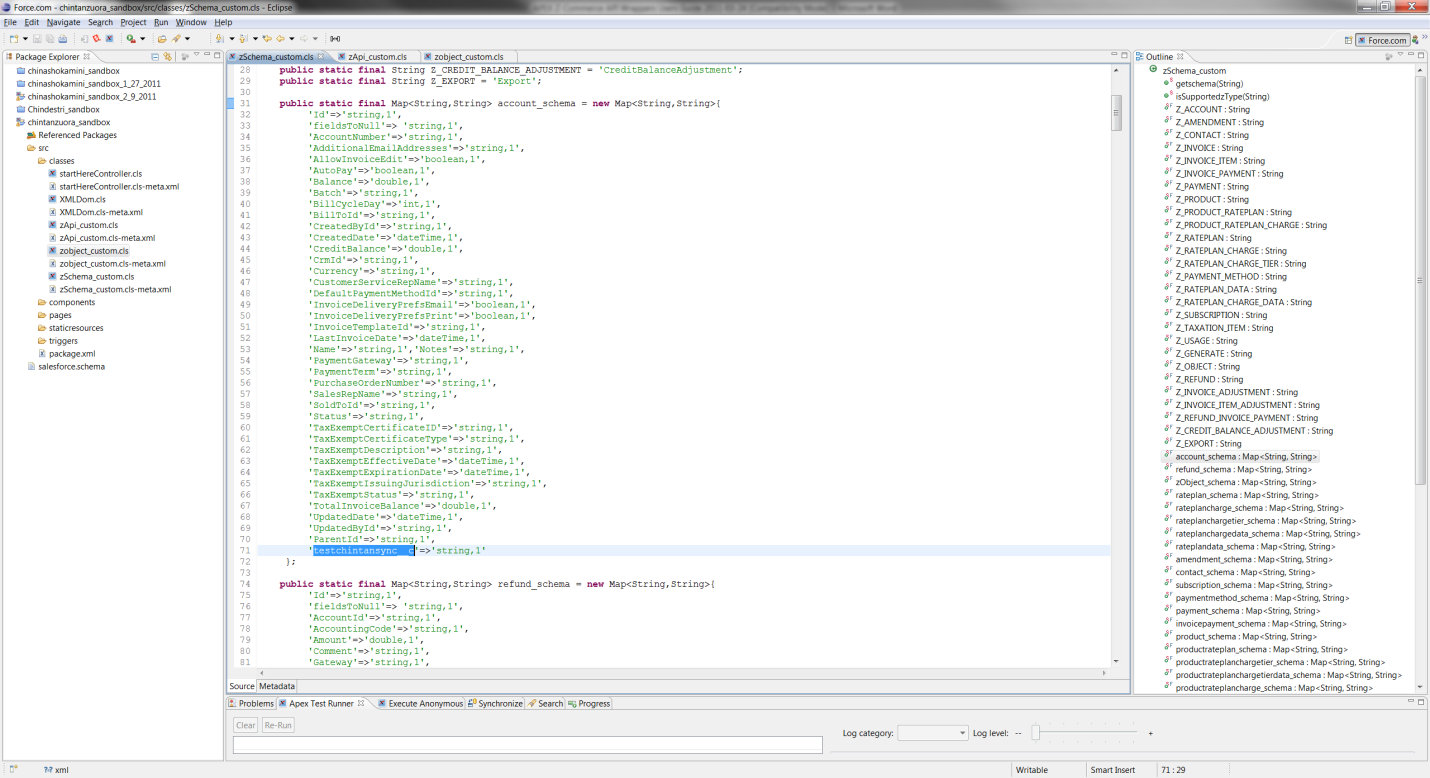
Please note down the API name for this field; as shown below the API name is “testchintansync\_\_c”.



Now go to your zSchema\_custom class and update the schema definition of “Account” object called “account\_schema” as shown below



And add the custom field you have just added in Zuora at the end of the “account\_schema” as shown below;



Now you can add this custom field in your API calls as shown below in blue highlight.

zAPi\_custom za = new zAPi\_custom();

za.setEndpoint('https://apisandbox.zuora.com/apps/services/a/27.0');

zApi\_custom.LoginResult loginResult = za.zlogin('chintan.desai@zuora.com','');

za.setSessionId(loginResult.Session);

// create account,zobject

zobject\_custom acc1 = new zobject\_custom(zSchema\_custom.Z\_ACCOUNT);

acc1.setValue('AccountNumber','123429');

acc1.setValue('AllowInvoiceEdit','false');

acc1.setValue('AutoPay','false');

acc1.setValue('Batch','Batch1');

acc1.setValue('BillCycleDay','1');

acc1.setValue('Currency','USD');

acc1.setValue('Name','chintancatt');

acc1.setValue('PaymentTerm','Net 30');

acc1.setValue('Status','Draft');

acc1.setValue(‘[testchintansync\_\_c](https://apisandbox.zuora.com/apps/CustomField.do?method=viewDetail&id=4028e6972f4b4f1b012f5a3f67063fca),’testing’);

zobject\_custom[] objs = new zobject\_custom[1];

objs[0] = acc1;

List<zApi\_custom.SaveResult> resultList = za.zcreate(objs);

for(zApi\_custom.SaveResult zobj : resultList)

{

if(zobj.Success)

{

System.debug('chintanZuoraAccount------------> Id =' + zobj.Id);

}

else

{

zobject\_custom[] errs = zobj.Errors;

for(zobject\_custom er : errs)

{

System.debug('chintan ------------>error =' + er.getValueString('error'));

}

}

}

Note here that if you do not have this custom field referenced in the “zSchema\_custom” class then you will get the following error in Salesforce.com.

11:05:09.986 (986568000)|FATAL\_ERROR|System.AssertException: Assertion Failed: Fieldname testchintansync\_\_c does not exist for zobject Account

Class.zobject\_custom.getFieldDescribe: line 340, column 9

Class.zobject\_custom.setValue\_0: line 22, column 28

Class.zobject\_custom.setValue: line 18, column 9

AnonymousBlock: line 24, column 1

11:05:09.873|CUMULATIVE\_LIMIT\_USAGE

1. An existing object schema has been changed (new elements have been added) in the WSDL.

Here is an example of an “Account” object schema from the WSDL. It has several elements; some of them may have been added recently in the new version of the WSDL and thus may not be reflected in the API wrappers.

<complexType name="Account">

<complexContent>

<extension base="ons:zObject">

<sequence>

<element minOccurs="0" name="AccountNumber" nillable="true" type="string" />

<element minOccurs="0" name="AdditionalEmailAddresses" nillable="true" type="string" />

<element minOccurs="0" name="AllowInvoiceEdit" nillable="true" type="boolean" />

<element minOccurs="0" name="AutoPay" nillable="true" type="boolean" />

<element minOccurs="0" name="Balance" nillable="true" type="decimal" />

<element minOccurs="0" name="Batch" nillable="true" type="string" />

<element minOccurs="0" name="BillCycleDay" type="int" />

<element minOccurs="0" name="BillToId" nillable="true" type="zns:ID" />

<element minOccurs="0" name="CreatedById" nillable="true" type="zns:ID" />

<element minOccurs="0" name="CreatedDate" nillable="true" type="dateTime" />

<element minOccurs="0" name="CrmId" nillable="true" type="string" />

<element minOccurs="0" name="Currency" nillable="true" type="string" />

<element minOccurs="0" name="CustomerServiceRepName" nillable="true" type="string" />

<element minOccurs="0" name="DefaultPaymentMethodId" nillable="true" type="zns:ID" />

<element minOccurs="0" name="InvoiceDeliveryPrefsEmail" nillable="true" type="boolean" />

<element minOccurs="0" name="InvoiceDeliveryPrefsPrint" nillable="true" type="boolean" />

<element minOccurs="0" name="InvoiceTemplateId" nillable="true" type="zns:ID" />

<element minOccurs="0" name="LastInvoiceDate" nillable="true" type="dateTime" />

<element minOccurs="0" name="Name" nillable="true" type="string" />

<element minOccurs="0" name="Notes" nillable="true" type="string" />

<element minOccurs="0" name="PaymentGateway" nillable="true" type="string" />

<element minOccurs="0" name="PaymentTerm" nillable="true" type="string" />

<element minOccurs="0" name="PurchaseOrderNumber" nillable="true" type="string" />

<element minOccurs="0" name="SalesRepName" nillable="true" type="string" />

<element minOccurs="0" name="SoldToId" nillable="true" type="zns:ID" />

<element minOccurs="0" name="Status" nillable="true" type="string" />

<element minOccurs="0" name="UpdatedById" nillable="true" type="zns:ID" />

<element minOccurs="0" name="UpdatedDate" nillable="true" type="dateTime" />

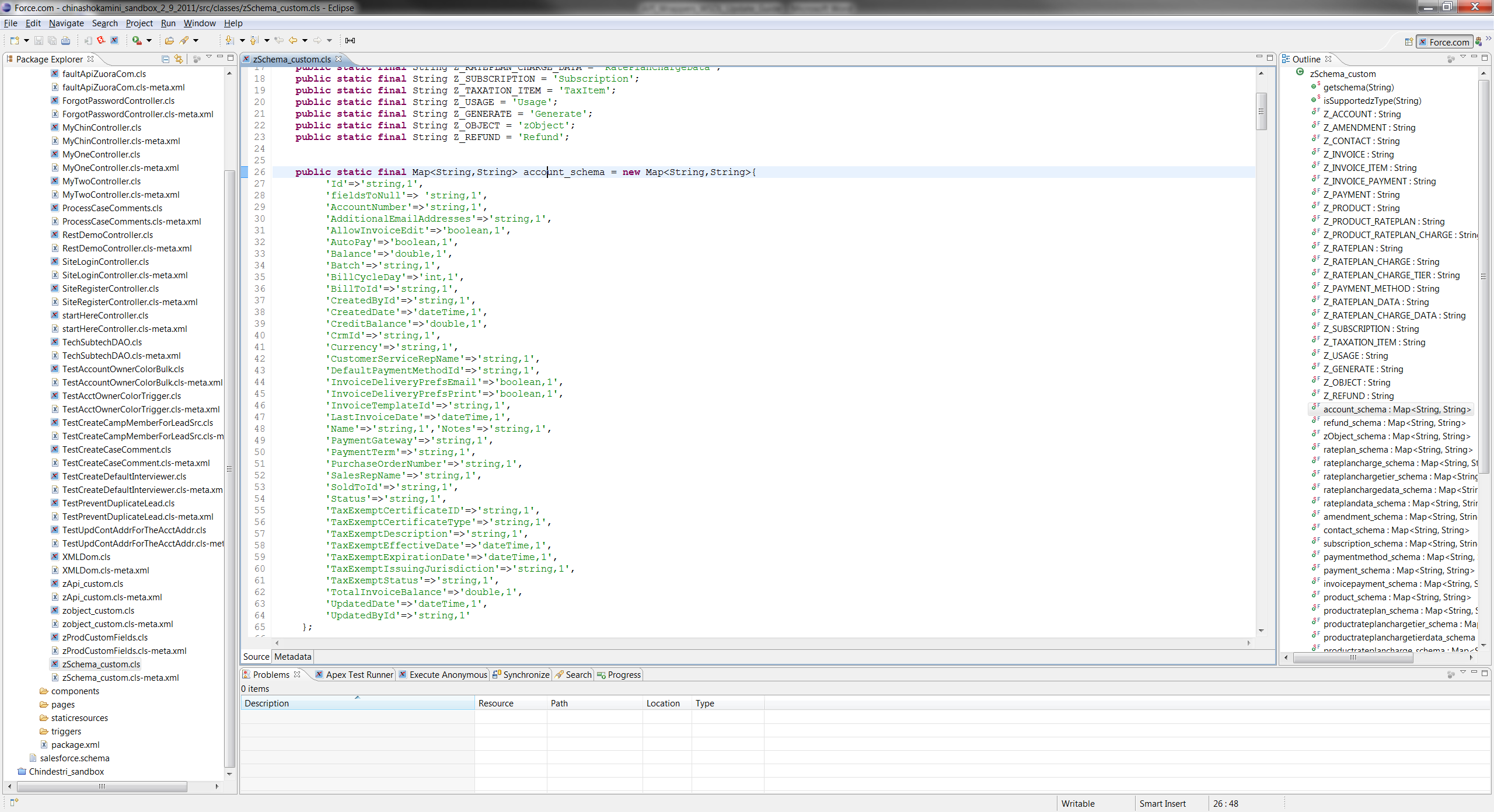
</sequence>

</extension>

</complexContent>

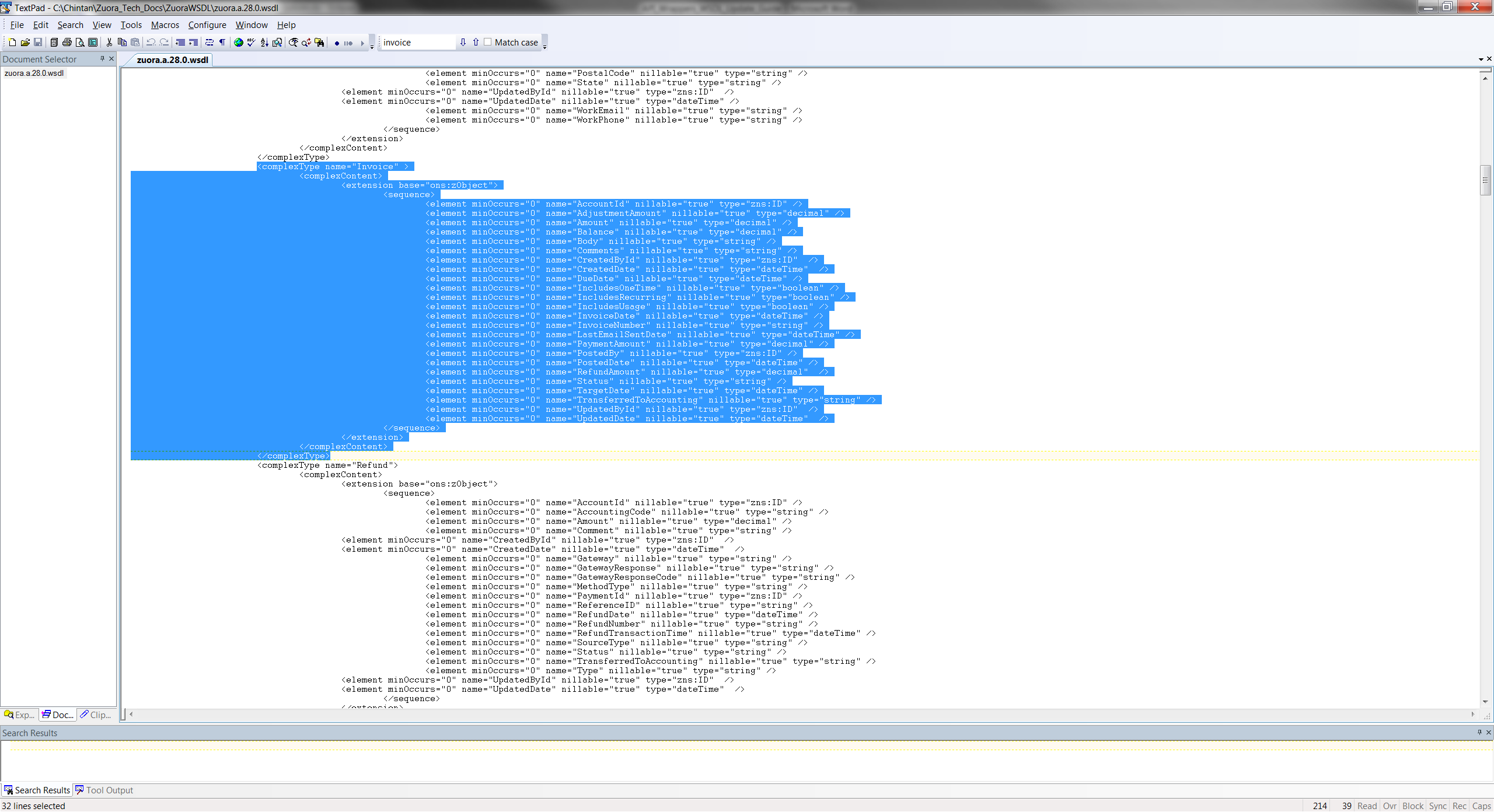
</complexType>

To make sure this definition of an “Account” object schema is updated in the API wrapper, the “zSchema\_custom” class may need to change. As shown below in the “zSchema\_custom” class, the “account\_schema” should be found and verified that all the elements from the WSDL are included with an appropriate data type.



1. WSDL has a schema definition that does not exist at all in the API wrapper; this is a brand new schema.

As shown below, the WSDL has an object schema defined with the name “Invoice”:



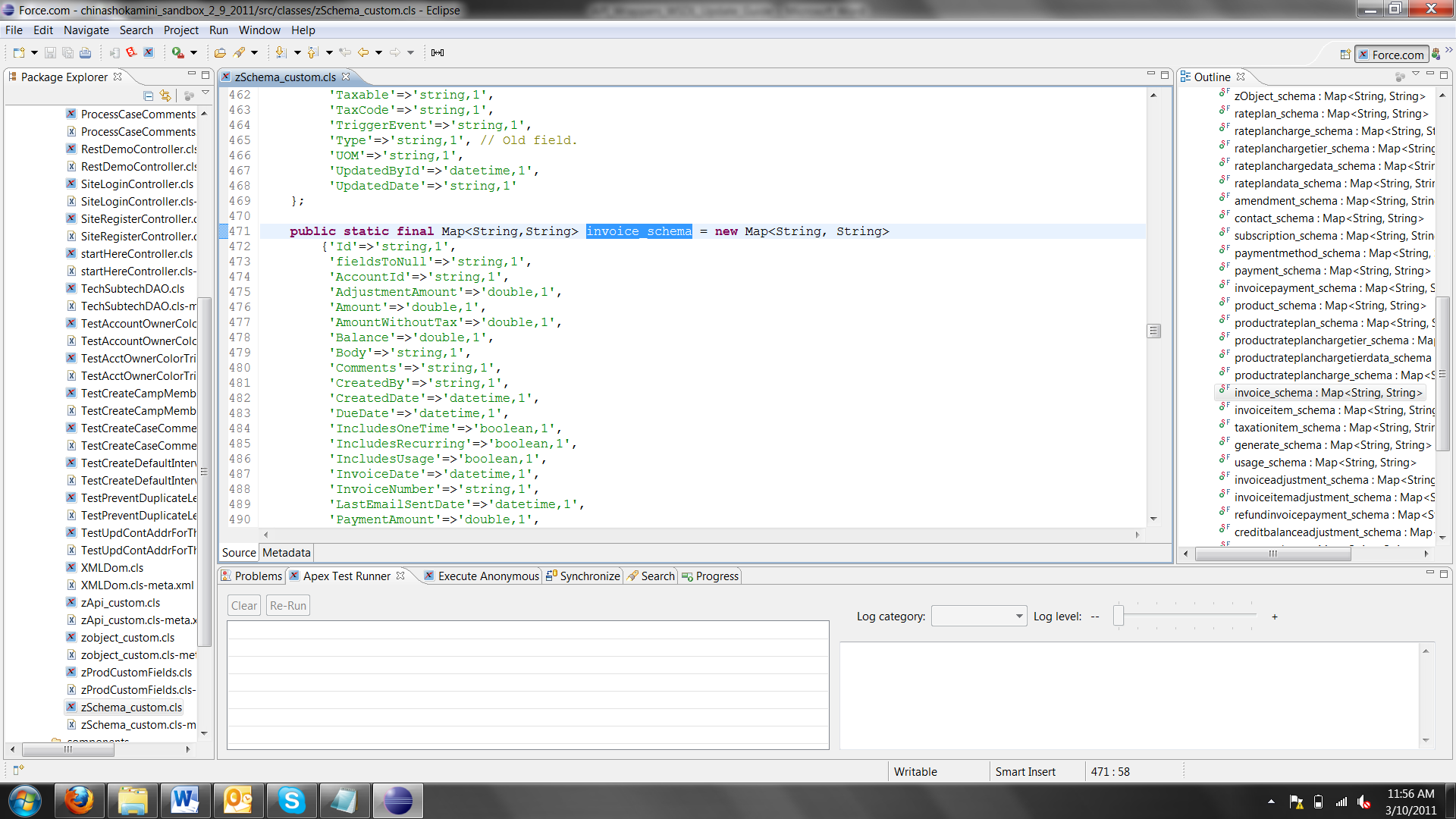
However, this object schema is not reflected in the “zSchema\_custom” API wrapper. This new object schema needs to be added in the wrapper class.

Following set of diagrams shows several touch points that needs to be added/edited to add a new object schema:

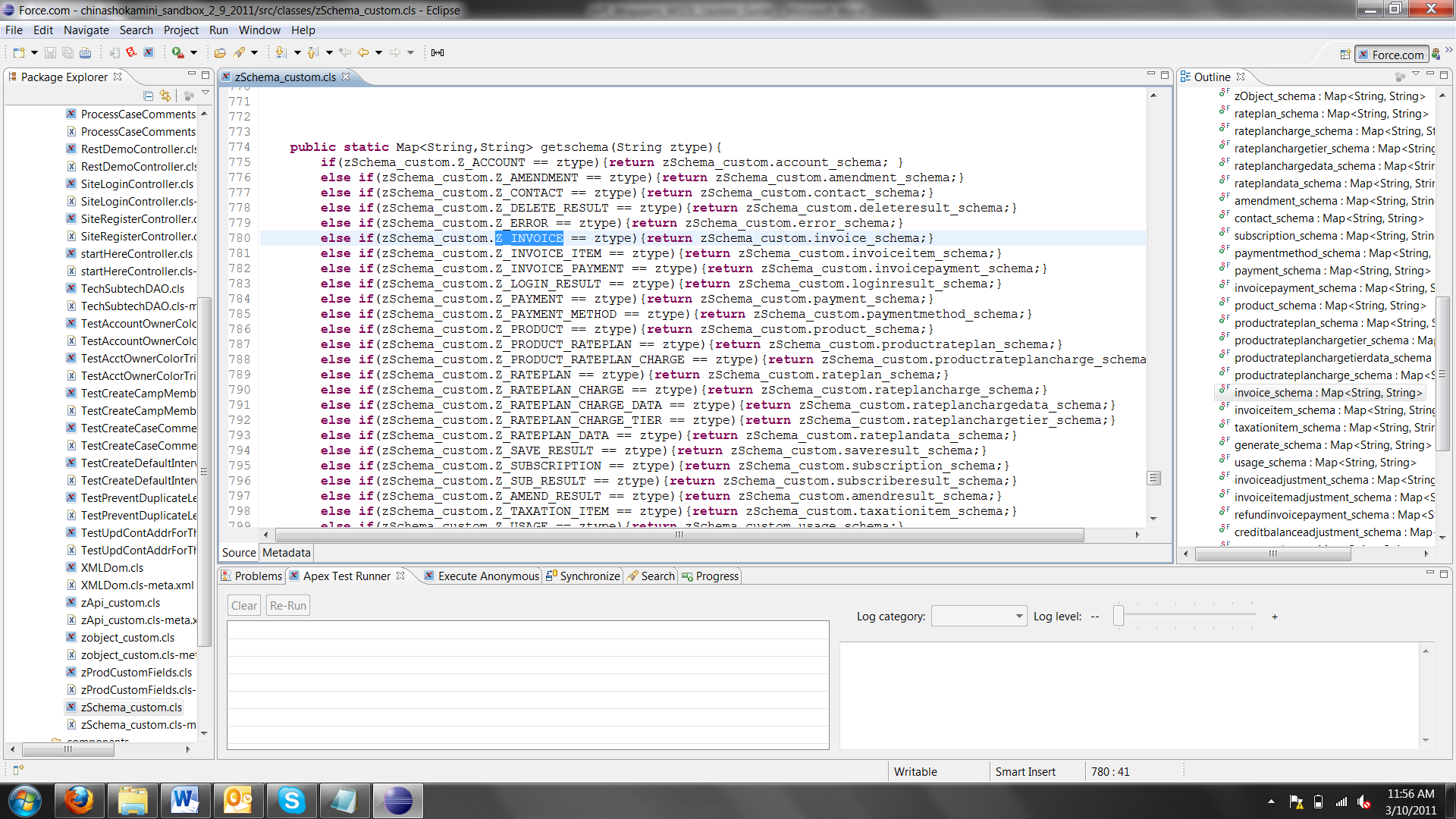
* 1. Define a new static type for the new schema.



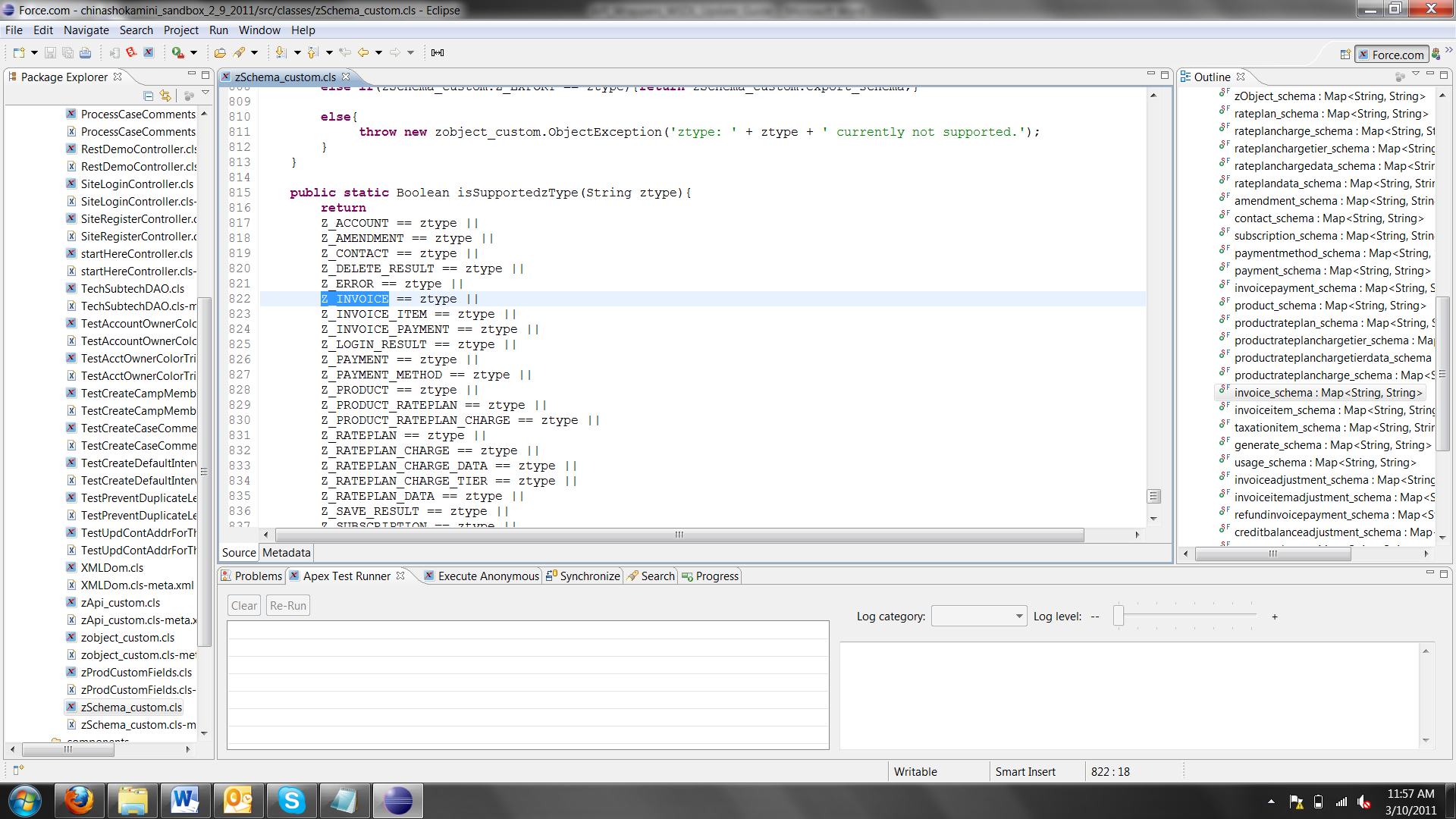
* 1. Define the schema with all the elements ;make sure the correct element types are defined.



* 1. Define the mapping between the static type that is defined and the new schema that is defined so that other code outside of this class can use it.



* 1. Add the new static type as a supported ztype.



All done! Thus any time there is a need to add a new object schema, please make sure all the four edits mentioned above are performed.

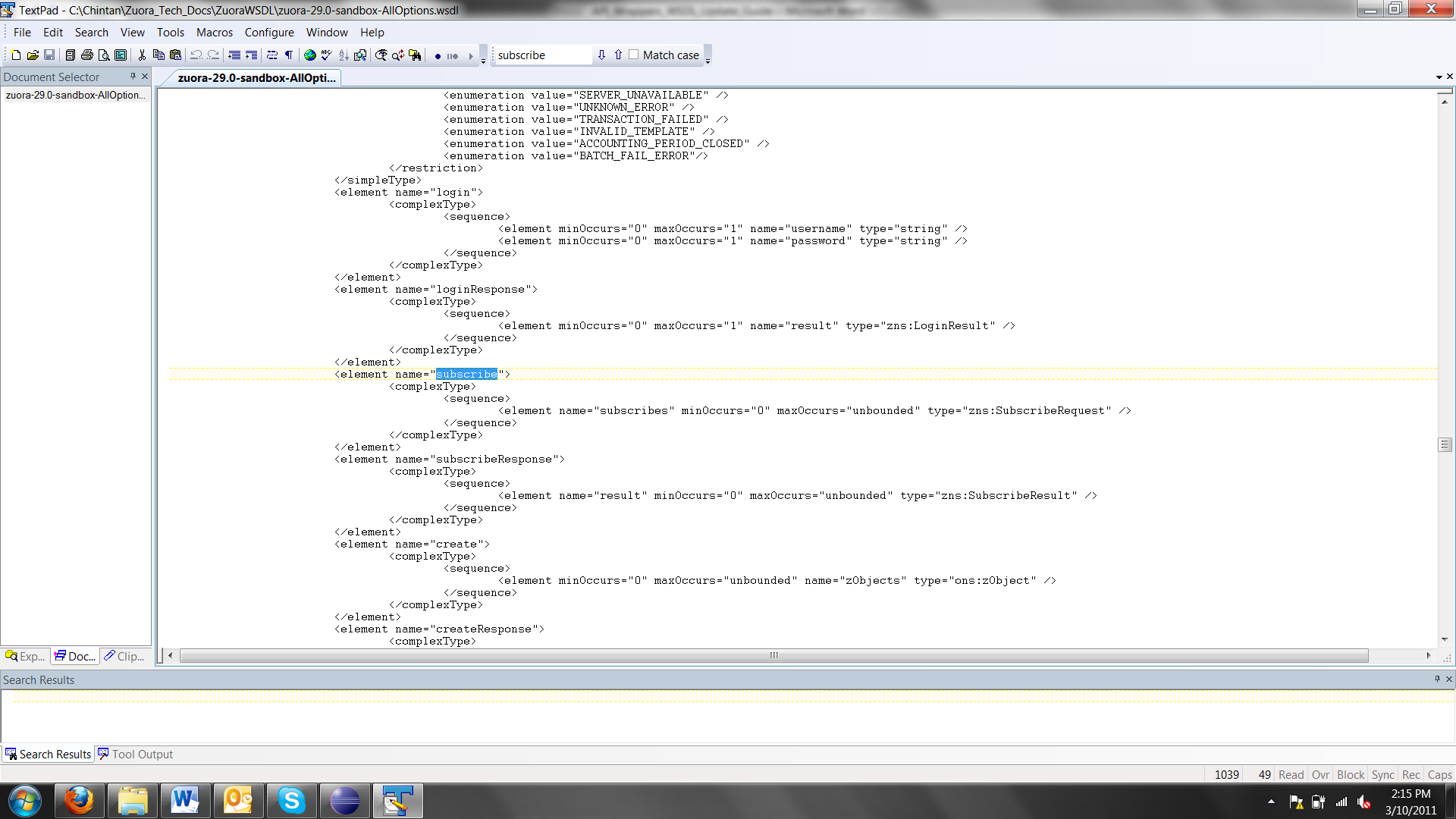
4. WSDL has a new operation.

For the purpose of this discussion, let’s define the namespace from the WSDL. Listed here are the two most important namespaces defined in the WSDL:

xmlns:zns="http://api.zuora.com/"

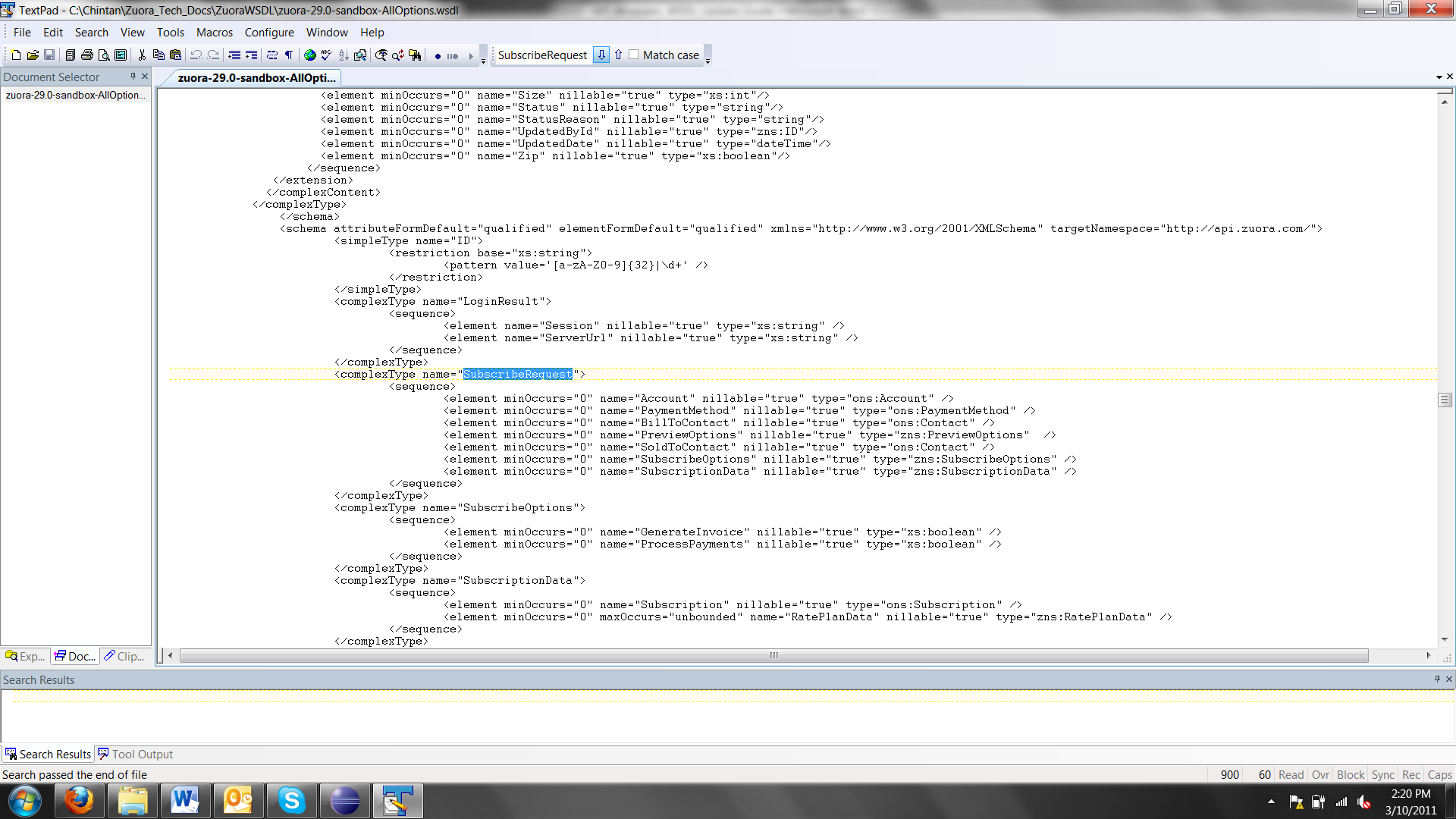
xmlns:ons="http://object.api.zuora.com/"

As shown in the screenshot below, there is a “subscribe” call that the WSDL has defined.

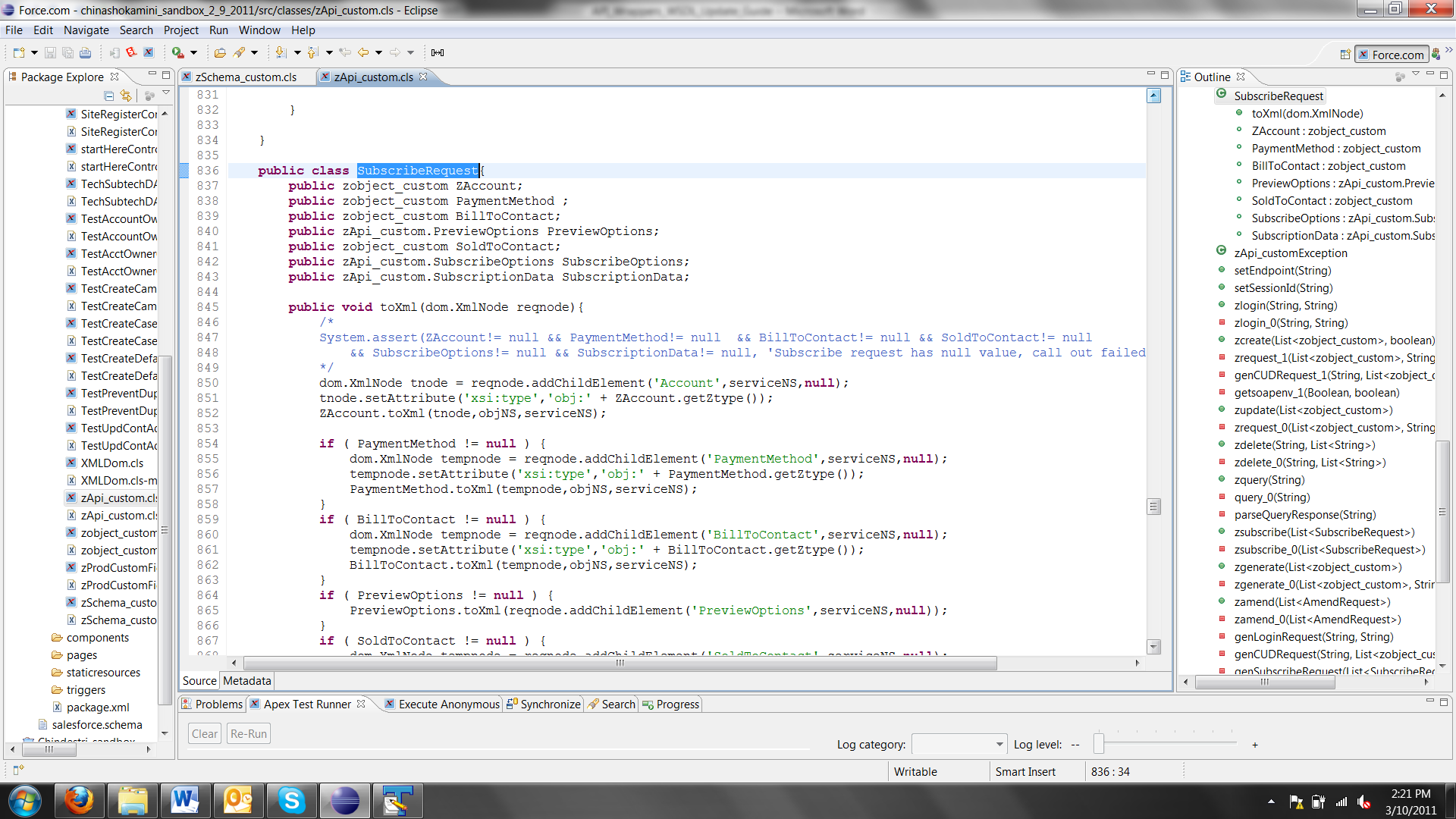


It’s made up of the complex object of type “SubscribeRequest.” This type has to be defined in the “zApi\_custom” wrapper class.

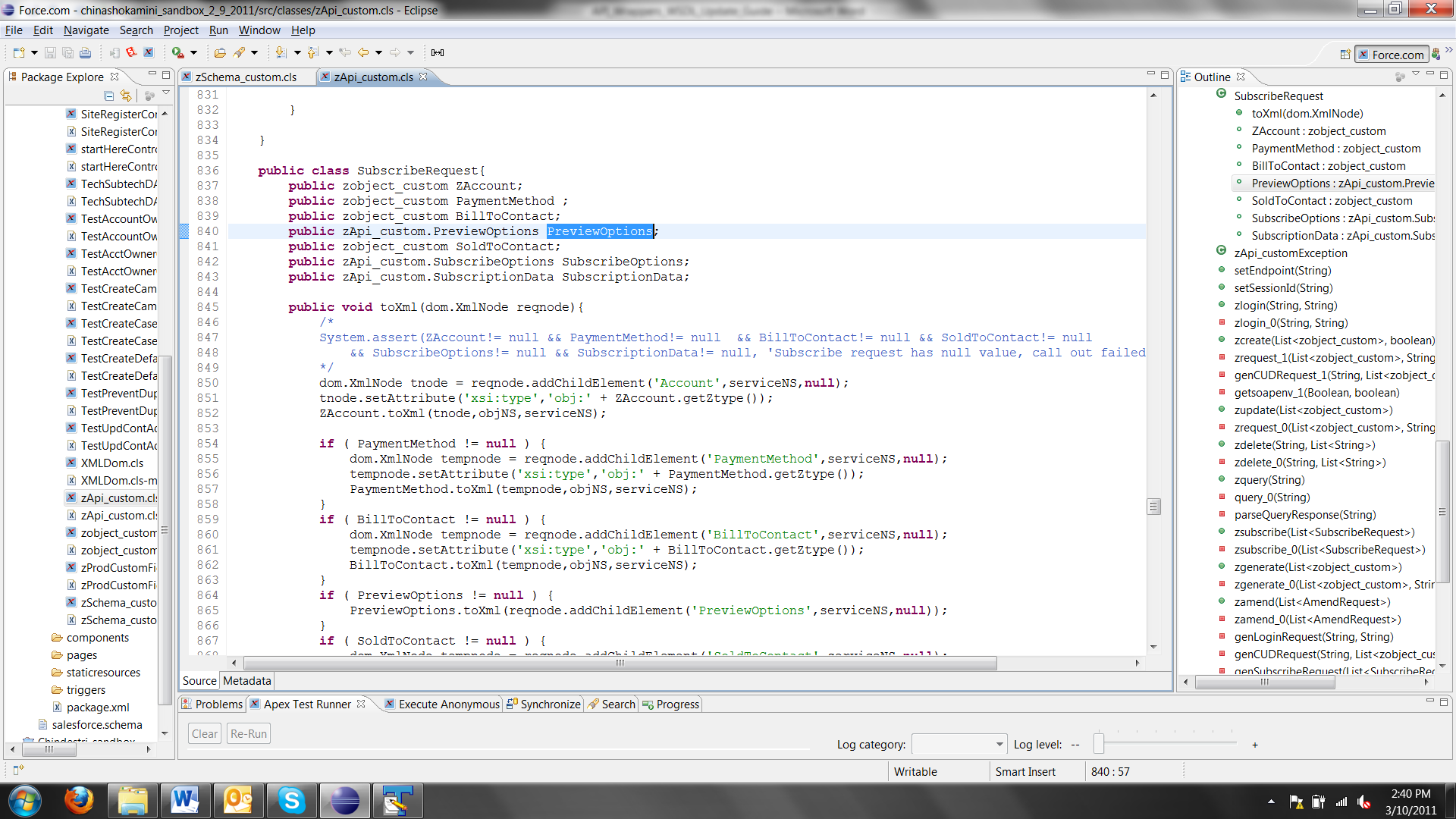
Here is how the “SubscribeRequest” complex schema has been defined.



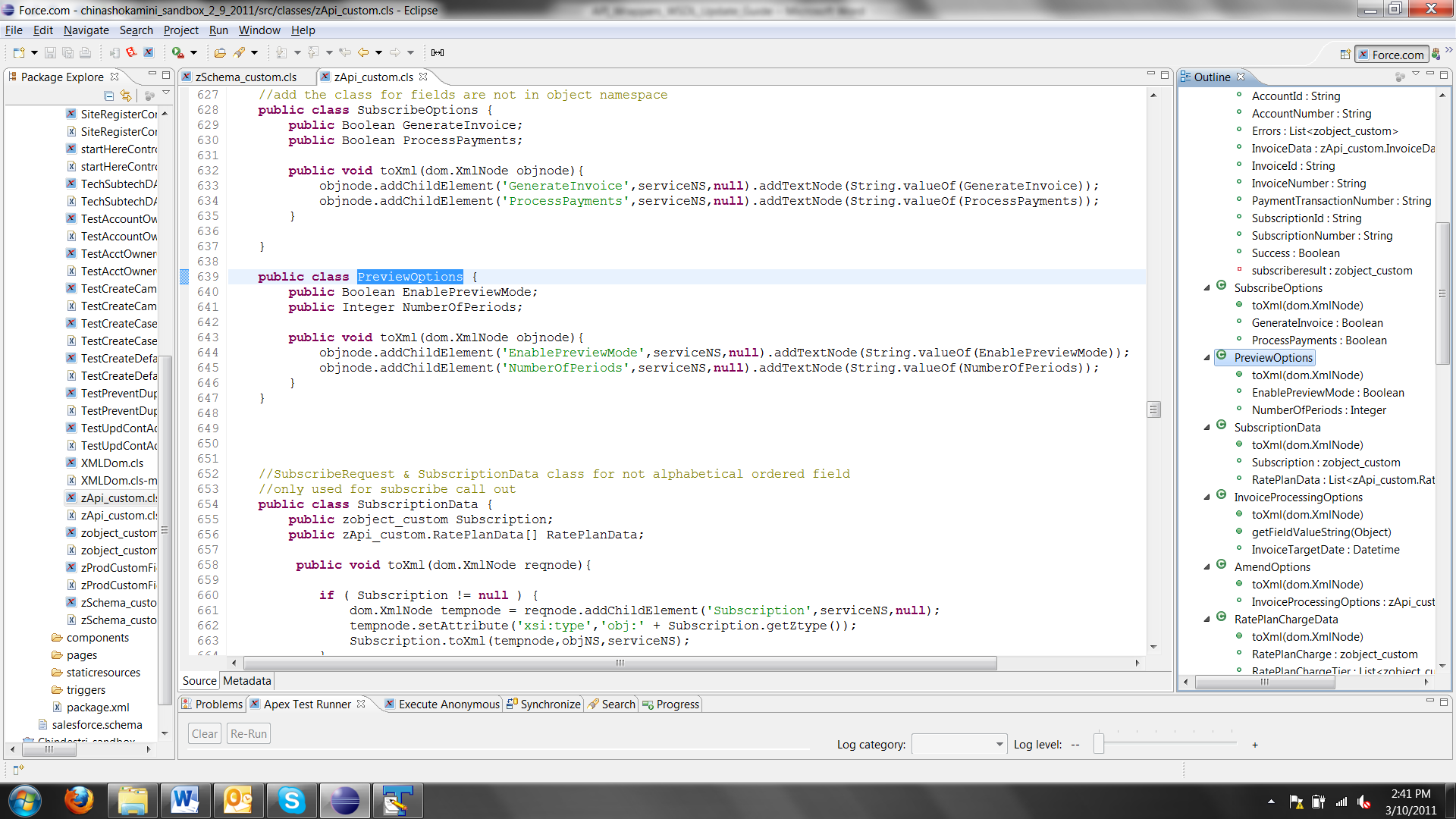
Here is how the class “SubscribeRequest” is defined in the “zApi\_custom” class to reflect the definition provided in the WSDL. Note that any sub element under “SubscribeRequest” that is under namespace “ons” is defined of type “zobject\_custom”.



Furthermore, any sub element under namespace “zns” has to be defined as a complex sub-class. For example, as shown below, “PreviewOptions” is defined as a complex sub-class.



This is how the “PreviewOptions” subclass is defined with the code to serialize the attributes.



Note here, that any time a new class is created to represent a complex type, code will have to be written to serialize the attribute of each class as shown below.

**public** **void** toXml(dom.XmlNode objnode)

{

objnode.addChildElement('EnablePreviewMode',serviceNS,**null**).addTextNode(String.valueOf(EnablePreviewMode));

objnode.addChildElement('NumberOfPeriods',serviceNS,**null**).addTextNode(String.valueOf(NumberOfPeriods));

}

This completes the API Wrapper maintenance section.