Avalara Connector
End User Documentation

arvato
BERTELSMANN

New York, May 1st, 2018

Table of Contents

1 Avalara Extension Installation

1.1 Extension Installation

Pre-Requesites

1. Minimum hybris 640 platform installation.
2. ArvatoAvalaraExt extension arvatoAvalaraConnector.zip
3. An Avalara account with a valid company Id and license

1.2 Installation Steps

1. If hybris is running be sure to stop first by navigating to ${HYBRIS_HOME}/bin/platform/ and executing one of the two scripts based on whether a linux based or windows based server is running.
    a. **For Linux:** sh hybrisserver.sh stop
    b. For Windows:hybrisserver stop
2. Unzip the arvatoAvalaraConnector.zip folder into the ${HYBRIS_HOME}/bin/platform/custom directory
3. edit the localextensions.xml located in ${HYBRIS_HOME}/config and add the below entry to the <extensions node>

    Code Block 1 localextensions.xml

```
<extensions>
  <path dir='${HYBRIS_BIN_DIR}' autoload='false' />
    <path dir='${HYBRIS_BIN_DIR}/ext-integration/sap/'
autoload='false' />
  .
  .
  .
  <extension name='arvatoAvalaraConnector' />
</extensions>
```

4.
5. Navigate to {PLATFORM_HOME}/bin/platform and execute the ant all target to compile and build the project
6. Once the build is complete start the hybris server by navigating to the ${HYBRIS_HOME}/bin/platform/ directory and running one the appropriate script based on the operating system
    a. **For Linux:** sh hybrisserver.sh start
    b. **For Windows:** hybrisserver start
7. One hybris has started run the system update by opening the hybris admin console using the browser of your choice. If installing locally the url would be https://127.0.0.1:9002/platform/update.

8. Under the Project Data Settings croll down to the arvatoAvalaraConnector and check the box next to it

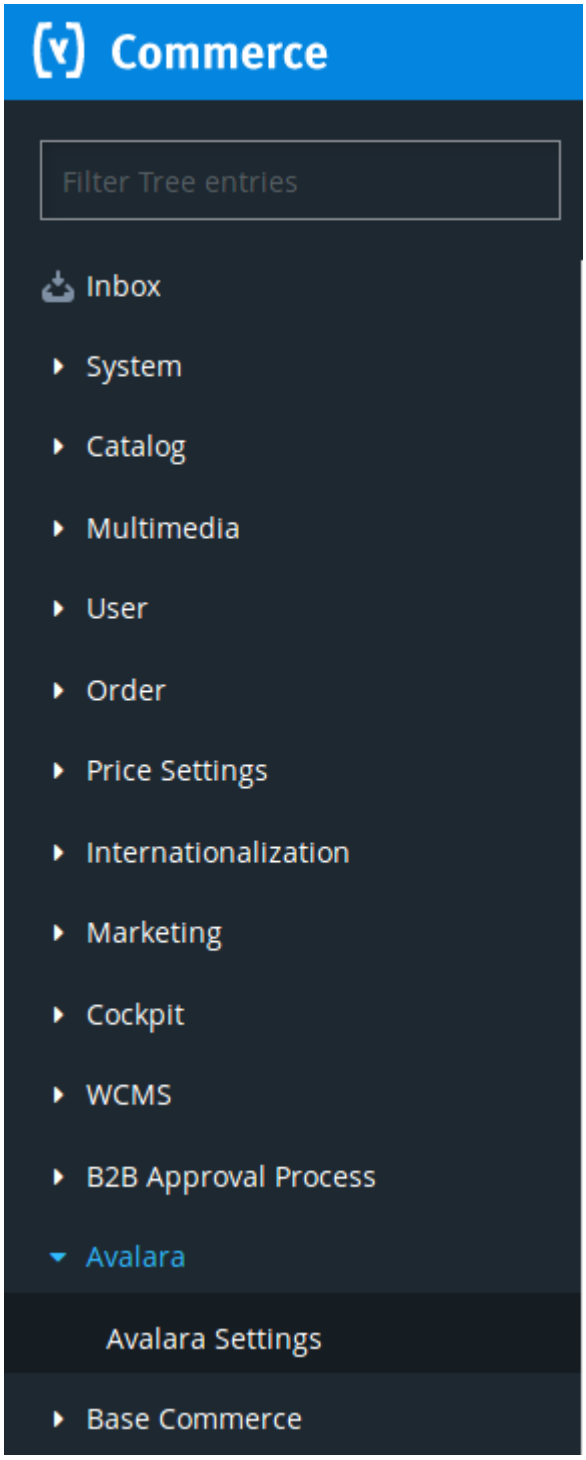
9. Finally click on the Update Button to allow the appropriate tables to be added to the hybris database

1.3 Related articles

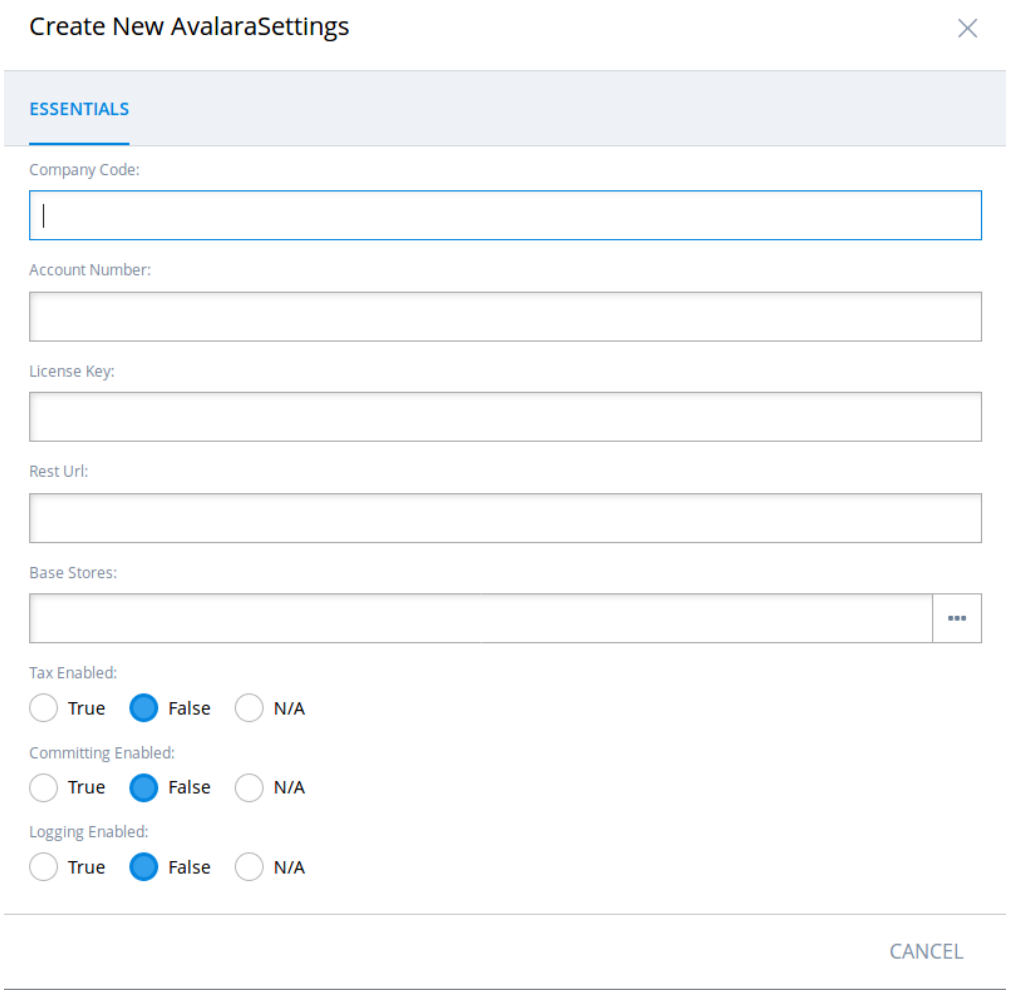
- Page: Base Store Settings
- Page: Avalara Extension Installation
- Page: Hybris Services injection and use cases
- Page: Product Tax Codes
- Page: Setting up extension for development

2 Creating new Avalara Settings

1) Login into the Backoffice, in development environments you can use *admin/nimda*.
2) In the navigation tree click on *Avalara Settings* found under *Avalara*.



3) Click on the + icon to open the Creation Wizard popup.



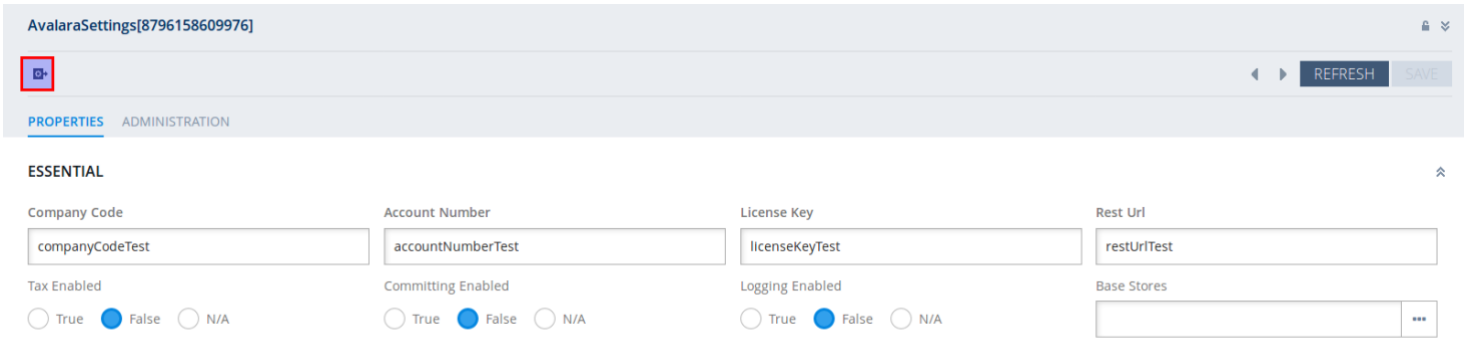
4) The mandatory fields are Company Code, Account Number, License Key, and Rest Url. Once these are provided the *Done* button will be available to be clicked on.

By default the following attributes are set to false:

- taxEnabled: If this is *false,* requests will not be sent to Avalara.
- committingEnabled: If this is *false*, transactions will not be marked as *Committed* in Avalara and therefore will not be available to be reported to a tax authority by Avalara Managed Returns.
- logginEnabled: If this is *false*, requests between Hybris and Avalara will not be logged in the console.

  By default, the Base Stores attribute will be empty. When used in a store context, the first Avalara Settings found will be used if none is specified.

5) Once created, the settings can be modified and tested in the editor view. To test a Setting click on the Ping icon on the view's action bar. You should see a Ping Success or Error message.

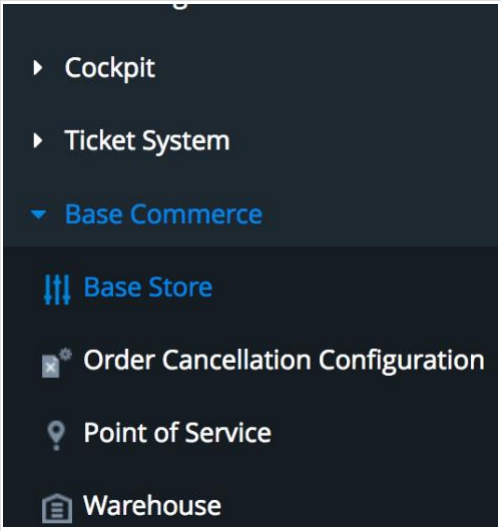
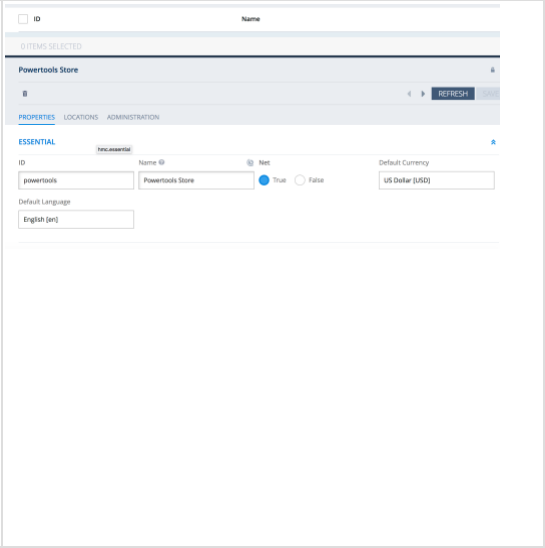

AvalaraSettings[8796158609976]
REFRESH
SAVE
PROPERTIES
ADMINISTRATION
ESSENTIAL
Company Code
companyCodeTest
Account Number
accountNumberTest
License Key
licenseKeyTest
Rest Url
restUrlTest
Tax Enabled
True
False
N/A
Committing Enabled
True
False
N/A
Logging Enabled
True
False
N/A
Base Stores

3 Base Store Settings

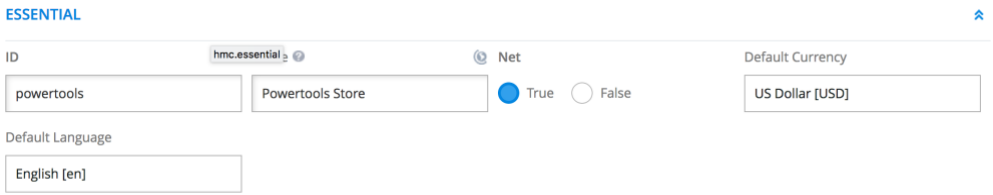
In order for Avalara to be used every store will need to be configured as below.

3.1 Base Store Configurations

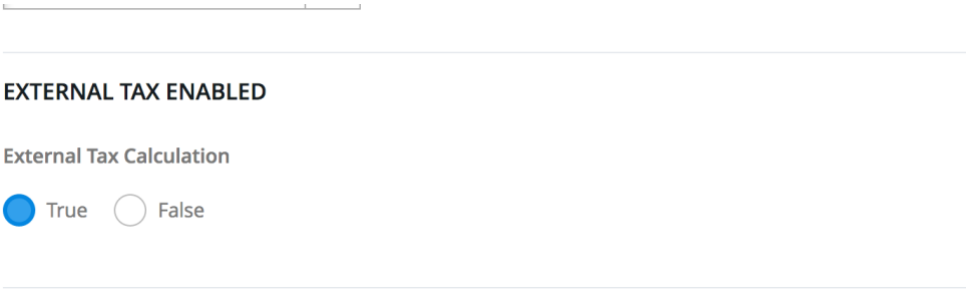
1. In the backoffice navigation tree expand the Base Commerce Node and click on Base Store to open the Base Store search window. Click the store from the result screen to populate the detail view.

Navigation Tree	Detail View
	

2. Under the Properties tab select true on the net property



3. Scroll down the window to locate the External Tax Enabled Setting and select true on that property.



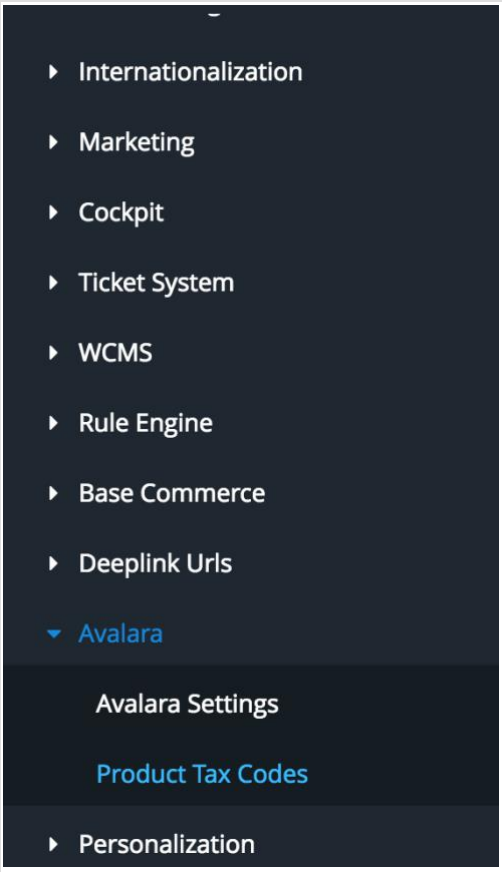
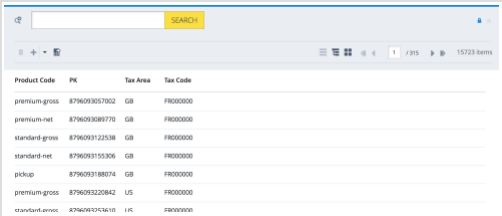
3.2 Related articles

- Page: Base Store Settings
- Page: Avalara Extension Installation
- Page: Hybris Services injection and use cases
- Page: Product Tax Codes
- Page: Setting up extension for development

4 Product Tax Codes

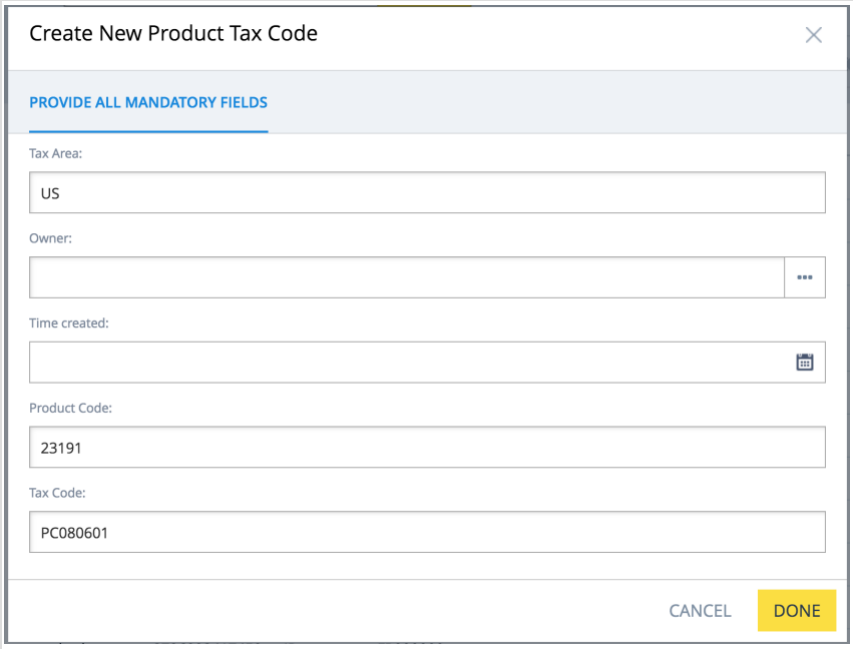
4.1 Creating a new Product Tax Code

1. log in to the hybris backoffice https://127.0.0.1:9002/backoffice/
2. In the navigation tree expand the Avalara node and double click on Product Tax Codes to open the TaxCode Listing.

Navigation Tree	Result List
	

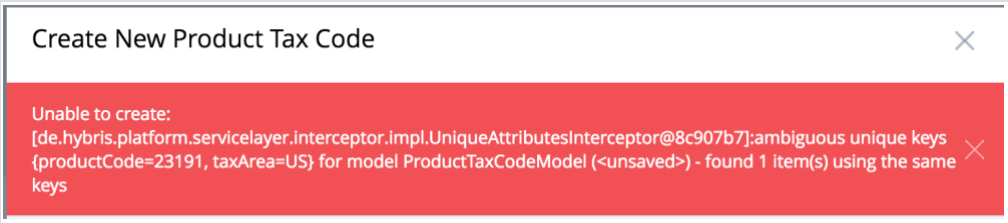
3. In the upper left click the plus sign to open the item creation form The fields to enter are as below. All fields must be unique there cannot be duplicated records to Product Code and Tax Area. For example, a product with code 2231913 cannot have more than one code in the US area.
    a. Tax Area - The two-character ISO code of the country i.e. US for United States, GB for Great Britain, JP for Japan. **This field is not editable once the entry is created**
    b. Product Code - the code of the Product in hybris**. This field is not editable once the entry is created**
    c. Tax Code - the Avalara tax code i.e. PC080601

Product Tax Code entry form



4. Once the form has been entered click done at the lower left. If there is already an entry in hybris for the Product Code An error will show at the top of the form like below. If this happens the existing entry for the product code will need to be edited.

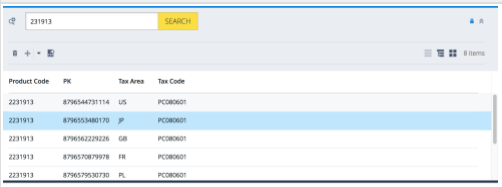
Error



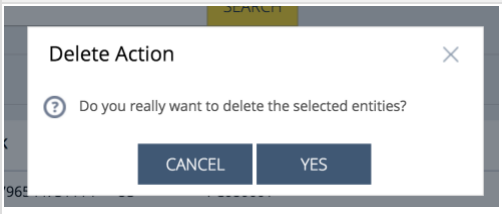
4.2 Editing an existing Product Tax Code

1. In the Backoffice explorer tree navigate to Avalara Settings -> Product Tax Codes (see screenshot above)
2. In the TaxCode listing enter either the Product Code or TaxCode and click Search to narrow down the results. Select the Entry you would like to delete and click the trash can at the upper left to remove it from the hybris database. A confirmation dialog will open and click yes to confirm the deletion

Searching for product 23193

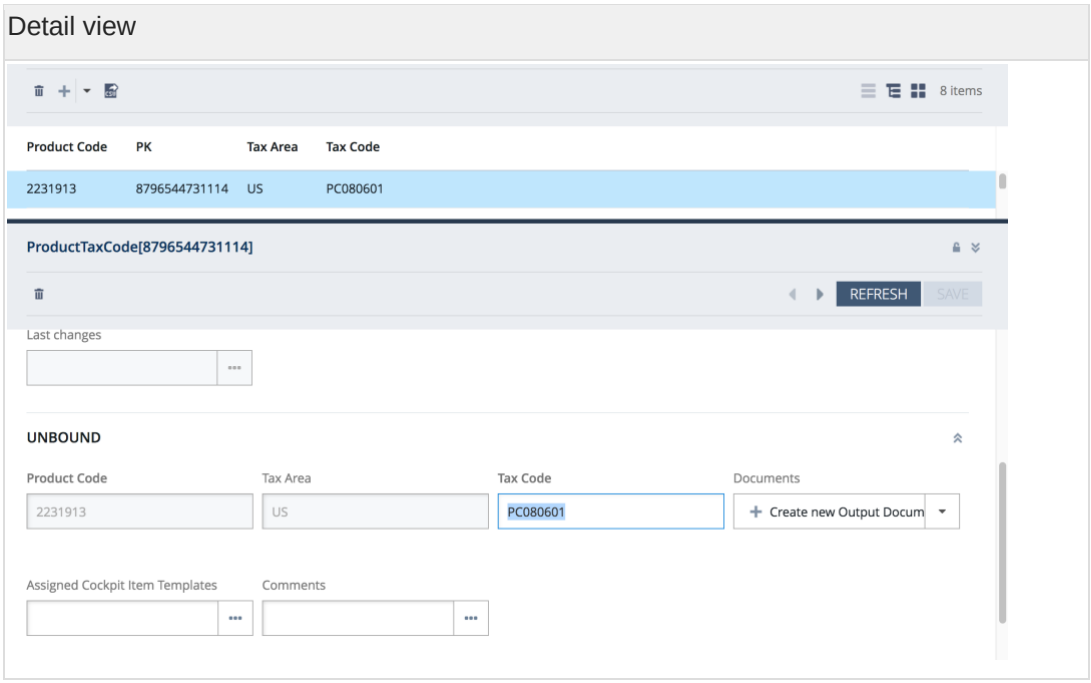


Confirmation dialog



4.3 Deleting an existing Product Tax Code

1. In the Backoffice explorer tree navigate to Avlara Settings -> Product Tax Codes (see screenshot above)
2. In the TaxCode listing enter either the Product Code or TaxCode and click Search to narrow down the results (see screen shot above).
3. Double click the entry matching the country that needs to be edited in order to populate the detail view in the center panel. The only field that may be changed is the tax code. If any other field needs to be changed the record must be deleted and then re-created.



4.4 Related articles

- Page: Base Store Settings
- Page: Avalara Extension Installation
- Page: Hybris Services injection and use cases
- Page: Product Tax Codes
- Page: Setting up extension for development

5 ArvatoAvalaraTaxConnectorService

1. The default hybris out of the box usage will only make use of one method in the CalculateExternalTaxesStrategy which is calculateExternalTaxes(AbstractOrderModel). This will do one of two actions based on the type of object that is passed in.
    a. **CartModel** will make a quote request against Avalara to get the calculated tax. This will not create any documents in the Avalara administration console
    b. **OrderModel** will create a post request against Avalara to calculate the taxes and create a document in COMMITTED status in the Avalara administration console.
2. There is also support for cancellations, invoices, checks for previously created documents, adjustments, and tests for Avalara connectivity. With the exception of ping the return value for all of these will be an Optional<ExternalTaxDocument>. If there is nothing returned by Avalara the response will be Optional.empty().
    a. **CANCEL**
        i. In order to void you will need to use the ArvatoAvalaraTaxConnectorService interface and inject it into the hybris cancellation service that is used. Consider the sample java class below that will set an order status to CANCELLED and then cancel the tax document. Also note the spring bean definition immediately after.

Code Block 2 Cancel Example

```
public class SampleCancelRequestExecutor
implements OrderCancelRequestExecutor
{

    private ModelService modelService;
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;

    public void
processCancelRequest(final
OrderCancelRequest orderCancelRequest,
            final
OrderCancelRecordEntryModel
cancelRequestRecordEntry) throws
OrderCancelException
    {
     AbstractOrderModel orderModel =
cancelRequestRecordEntry.getConsignment(
).getOrder();

orderModel.setStatus(OrderStatus.CANCELL
ED);

  arvatoAvalaraTaxConnectorService.cancel
(orderModel);
    }
```

    b. **INVOICE**
        i. This method is used to make a request to Avalara to get the tax calculations. It will also create a sales invoice in uncommitted status in the Avalara admin console. This would be used on a newly created order. Consider the below example for a CalculationService that will calculate the order total and send an invoice request to Avalara

Code Block 3 Invoice Example

```
public class OrderCalculationService
implements CalculationService{
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;
  public void
calculate(AbstractOrderModel order){
        double subtotal = 0.0;
        for (final
AbstractOrderEntryModel e :
order.getEntries())
        {

someMethodThatCalculatesTheOrderEntryTot
al(e);
             subtotal +=
e.getTotalPrice().doubleValue();
        }

  order.setTotalPrice(Double.valueOf(subt
otal));
     if(order instanceof OrderModel){

  arvatoAvalaraTaxConnectorService.invoic
e(order);
     }
  }
}
```

c. **EXISTS**

i. this method can be used to see it there is already a tax documented created for an existing order in Avalara. Building on the above example we can add a check to first see there was already a tax request and for the order and if there is it will not make an invoice request.

Code Block 4 Exists example

```
public class OrderCalculationService
implements CalculationService{
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;
  public void
calculate(AbstractOrderModel order){
        double subtotal = 0.0;
        for (final
AbstractOrderEntryModel e :
order.getEntries())
        {

someMethodThatCalculatesTheOrderEntryTot
al(e);
             subtotal +=
e.getTotalPrice().doubleValue();
        }

  order.setTotalPrice(Double.valueOf(subt
otal));
     if(!order instanceof OrderModel){
        return;
     }

  if(arvatoAvalaraTaxConnectorService.exi
sts(order).empty()){

  arvatoAvalaraTaxConnectorService.invoic
e(order);
     }
  }
}
```

d. **ADJUST**

i. This can be used to re-calculate taxes on an order whose amount has changed. Consider the below piece of code that calculates an existing OrderEntry and once done will send a request to Avalara to adjust the existing tax document. Building yet again on the above example if there was already a tax document instead of just making no call to Avalara an adjustment call can be made in this case.

Code Block 5 Adjust Example

```
public class OrderCalculationService
implements CalculationService{
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;
  public void
calculate(AbstractOrderModel order){
        double subtotal = 0.0;
        for (final
AbstractOrderEntryModel e :
order.getEntries())
        {

someMethodThatCalculatesTheOrderEntryTot
al(e);
              subtotal +=
e.getTotalPrice().doubleValue();
        }

  order.setTotalPrice(Double.valueOf(subt
otal));
     if(!order instanceof OrderModel){
        return;
     }

  if(arvatoAvalaraTaxConnectorService.exi
sts(order).empty()){

  arvatoAvalaraTaxConnectorService.invoic
e(order);
     }
     else{

  arvatoAvalaraTaxConnectorService.adust(
order);
     }
}
```

e. **PING**

i. This method can be used to ensure there is an active connection to Avalara. It will fail if there is a networking issue or if the user credentials are misconfigured in hybris. Building on the Calculation example from above

Code Block 6 Ping Example

```
public class OrderCalculationService
implements CalculationService{
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;
  public void
calculate(AbstractOrderModel order){
        double subtotal = 0.0;
        for (final
AbstractOrderEntryModel e :
order.getEntries())
        {

someMethodThatCalculatesTheOrderEntryTot
al(e);
            subtotal +=
e.getTotalPrice().doubleValue();
        }

  order.setTotalPrice(Double.valueOf(subt
otal));
    Optional<Boolean> pingResponse =
arvatoAvalaraTaxConnectorService.ping();
    if(!order instanceof OrderModel ||
pingResponse.empty() ||
Boolean.False.equals(pingResponse.get())
){
       return;
    }

  if(arvatoAvalaraTaxConnectorService.exi
sts(order).empty()){

  arvatoAvalaraTaxConnectorService.invoic
e(order);
    }
    else{

  arvatoAvalaraTaxConnectorService.adust(
order);
    }
}
```

f. **POST**

i. This method can be used to mark a previously invoiced tax document as committed. This may be used after an order has shipped to commit the tax document in Avalara. Consider the below implementation of a Warehouse2ProcessAdapter that will set the consignmentStatus to shipped and then check to see if all consignments have been shipped. If all consignments are shipped the post method will be invoked. The default implementation of the arvatoAvalaraTaxConnectorService will always send the document as committed when first created so it should not be necessary to manually commit documents.

Code Block 7 Post example

```
public class ShippingAdapter implements
Warehouse2ProcessAdapter{
  private ModelService modelService;
  private
ArvatoAvalaraTaxConnectorService
arvatoAvalaraTaxConnectorService;
  public void
receiveConsignmentStatus(final
ConsignmentModel consignment, final
WarehouseConsignmentStatus status){

  consignment.setStatus(ConsignmentStatus
.SHIPPED);
     modelService.save(consignment);
     boolean allShipped = true;
     for(ConsignmentModel consignment2 :
consignment.getOrder().getConsignments()
){
        allShipped = allShipped &&
(ConsignmentStatus.SHIPPED.equals(consig
nment2.getStatus()));
     }
     if(allShipped){

  arvatoAvalaraTaxConnectorService.post(c
onsignment.getOrder());
     }
  }
}
```

ArvatoAddressVerificationService

1. The ArvatoAddressVerificationService Does not have any new functionality. It is an implementation of the hybris out of the box AddressVerificationService. To inject the bean add the below into your spring.xml of your project.

Code Block 8 AvsInjection

```
    <alias name="arvatoAddressVerificationService"
alias="addressVerificationService"/>
    <bean id="arvatoAddressVerificationService"

class="com.arvatosystems.avalara.service.hybris.ArvatoAddressVerifi
cationService"/>
```

AvalaraDecideExternalTaxesStrategy

1. We have implemented the DecideExternalTaxesStrategy interface to customize the default shouldCalculateExternalTaxes method. This returns a boolean that defines if tax calls will be made to Avalara. To modify this logic you will need to change the following alias in the spring.xml of your project.

Code Block 9 decideExternalTaxesStrategy

```
<alias name="avalaraDecideExternalTaxesStrategy"
alias="decideExternalTaxesStrategy"/>
```

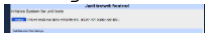
- Page: Base Store Settings
- Page: Avalara Extension Installation
- Page: Hybris Services injection and use cases
- Page: Product Tax Codes
- Page: Setting up extension for development

6 Setting up extension for development

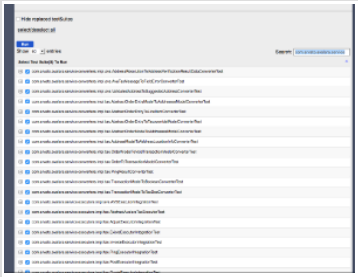
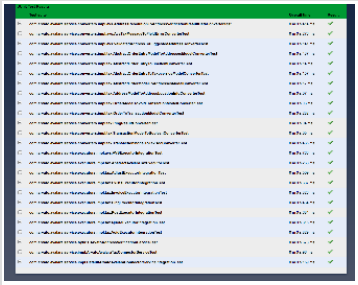
6.1 Step-by-step guide

Pre-requisites

1. This guide is for setup with mysql 5.6 or higher
2. The hybris version that is used is 6.4.0
3. java 1.8 is minimum version that should be set
4. Instructions are for macintosh machines.

1. Request access to the arvatoAvalaraExt on github by emailing itsupport@arvatosystems.com
2. Checkout arvatoAvalaraExt from github.
    a. `git clone git@github.com:arvatoSystemsNA/arvatoAvalaraExt.git`
3. Setup mysql database schema and users
    a. `mysql -u root -p`
    b. `CREATE SCHEMA `avalara` DEFAULT CHARACTER SET utf8;`
    c. `CREATE USER 'avalara'@'localhost' IDENTIFIED BY 'avalara';`
    d. `GRANT ALL ON avalara.* TO 'avalara'@'localhost';`
4. From the project root run the below commands
    a. `git submodule init`
    b. `git submodule update`
    c. `ant install`
    d. `ant clean all`
5. cd to hybris/bin/platform and start the hybris server using **sh hybrisserver.sh**
6. initialize the project by going to https://127.0.0.1:9002/platform/init in the web browser of your choice (user: admin, password: nimda)
7. Once the system is initialized as a final smoke test go to https://127.0.0.1:9002/test/
    a. Initialize the junit tenant by clicking the initialize button at the top 
    b. in the package filter enter **com.arvato.avalara.service**
    c. Click select all
    d. Click run and verify all of the unit tests pass

e.

Filter	Verify
	

Eclipse setup

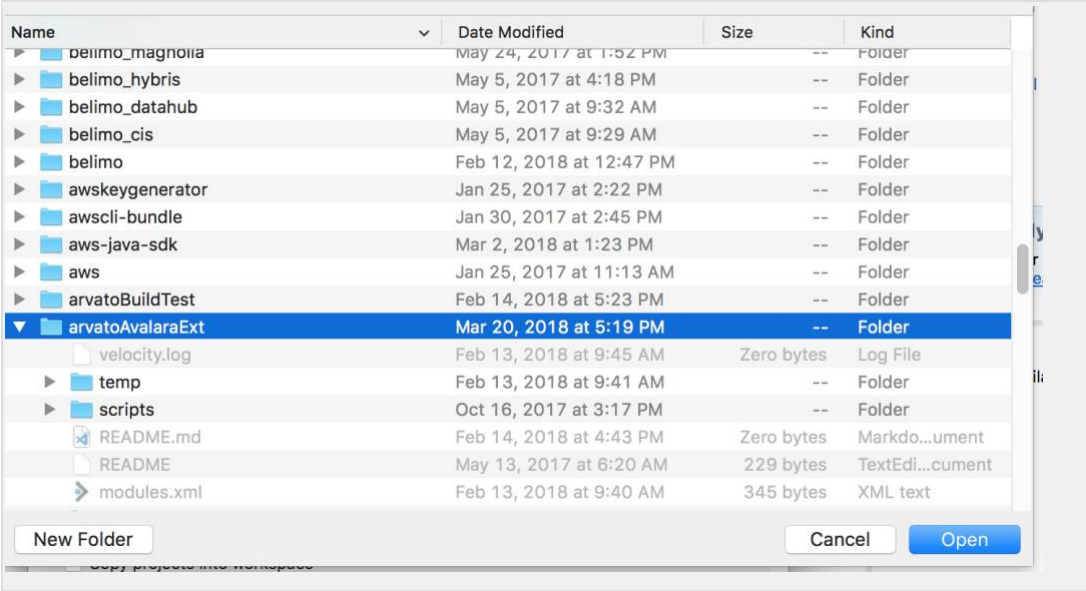
Before creating the eclipse workspace for this project make sure to first go through the itec pre-requisites for eclipse and hybris setup: Eclipse Hybris project setup

Also this jar will be required in order to allow communication with the mysql database: mysql-connector-java-5.1.34-bin.jar

6.2 Import hybris projects and arvatoAvalaraExt

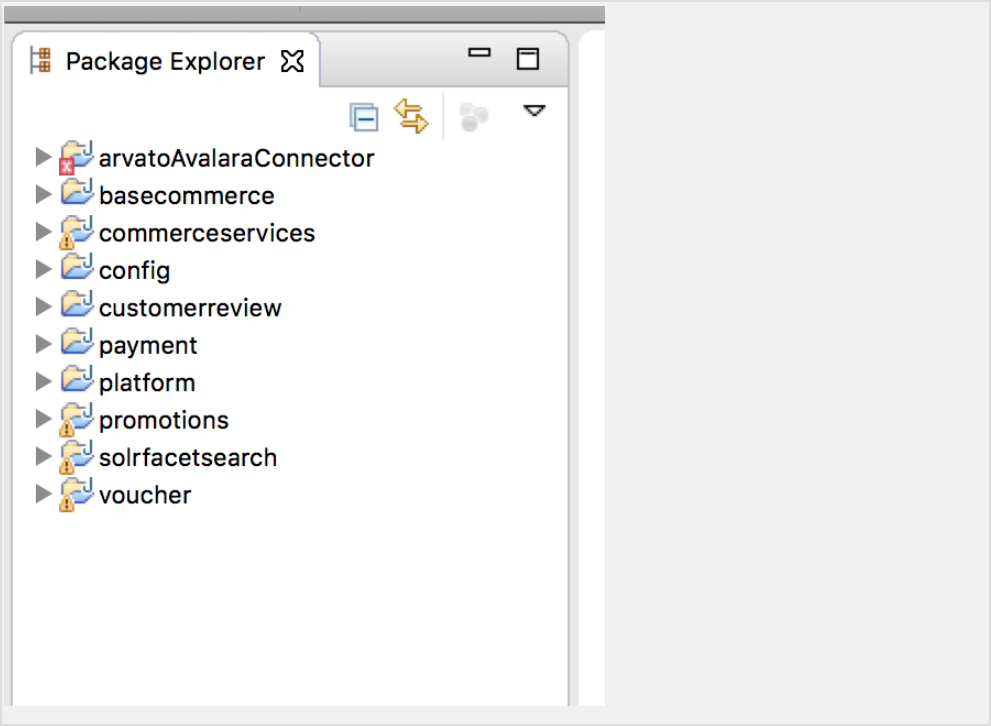
1. In eclipse click on file import
    a. expand General
    b. Highlight Existing Projects into Workspace
    c. click next
2. In the import Projects dialogue click Browse next to **Select root directory** this will open a file chooser
3. navigate to the directory on where the project was checked out to and click open

4.



5. This will list all of the available projects for import with every single one selected. Click on de-select all and check the box next to the below projects and click finish. Note that the main extensions this connector depends on is commerceservices and basecommerce however all of the below are required in order to allow eclipse to build successfully.
    a. arvatoAvalaraConnector
    b. basecommerce
    c. commerceservices
    d. config
    e. customerreview
    f. payment
    g. platform
    h. promotions
    i. solarfacetsearch
    j. voucher
6. the screenshot below is what your eclipse workspace should look like if imported correctly. Some errors may show in the arvatoAvalaraConnector project due to some generated code that does not adhere to the itec coding standard profile. Those can be ignored

a.



7. Right click on the arvatoAvalaraConnector mouse over build path and choose **Configure Build Path** from the context menu that opens.
    a. In the Java build path dialogue click the Libraries tab then click on **Add External Jars. . .**
    b. navigate to the mysql-connector-java-5.1.34-bin.jar and click on OK
    c. Click OK back on the **Configure Build Path** to complete the import of the jar
8. In the eclipse explorer navigate to arvatoAvalaraConnector/testsrc/com/arvato/avalara/service and right click on each of the below packages choose run as -> junit test and verify all tests pass
    a. converters.impl.avs
    b. converters.impl.tax
    c. executors.impl.avs
    d. executors.impl.tax
    e. hybris
    f. impl ** DefaultArvatoAvalaraConnectorServiceIntegrationTest may fail, this test can be ignored ***