Connecting WooCommerce Shop with ElevateDB Using Delphi VCL App

Prepared By: Jamal Banna

Aug 1- Aug 22

Table Of Contents

Setting Up Woocommerce Shop

This section assumes that you already have a shop. If you don't have a Woocommerce shop created please create one before continuing.

Authenticate to WooCommerce

This section will cover basic authentication. If you want to use OAuth1.0 Authentication or standard OAuth2.0 Authentication please refer to the following manual.

Basic Authentication

To set up basic auth navigate to your shop Settings => Woocommerce => Advanced => REST API

Press on add key to create a new key.

Select a user and give them read/write permissions.

Then copy the consumer key and consumer secret and store them somewhere along with your shop's url.

Connecting to your VCL Delphi App

This section assumes you already installed the Cdata Firedac Woocommerce plugin on your environment. This section also assumes you already know how to create a VCL app.

TFDPhysWooCommerceDriverLink

Drag and drop a TFDPhysWooCommerceDriverLink component from the design palette onto the form.

TFDConnection

Drag and drop a TFDConnection component from the design palette onto the form.
Double tap the component and select CDataWooCommerce in the Driver Id menu.
Inside the params, set the AuthScheme to basic and paste the shop url, consumer key and consumer password from where you stored them into the URL, UserName and Password fields respectively.
Set connected to true. You should be able to connect.

Data Explorer

From the application navbar select View => Tool Windows => Data Explorer.
From the FireDAC dropdown menu ,right click on CData Woocommerce Data Source and press on add a new connection.
Name the connection then a modification window will open. Set the AuthScheme, ShopURL, User, and Password same as you did in the TFDConnection component.
Right click the connection and refresh.

TFDQuery

Drag and drop a TFDQuery component from the design palette onto the form.

Double tap on it and set the connection name to the TFDConnection you created before.

Then tap on SQL and write any SQL Query.

Set active to true.

It should work.

Data Source

Drag and drop a TFDQuery component from the design palette onto the form.

Double tap the data source and set the Data Set to the Query you added before.

You should be able to set the enabled property to true.

DBGrid

In order to be able to display your query results if it was a select query, Drag and drop a DBGrid component from the design palette into the form.

Double click on it and select the data source you created before from the DataSource menu.

TWooCommerceConnector Class:

To use this class you need to put the WooCommerceConnector.pas file inside your project folder. Then import the class into your program.

Initializing Connection:

In your program create an object of type TWooCommerceConnector. This will create a TConnection for your object. Then call the SetUpConnection method and pass your ShopURL, ConsumerKey, ConsumerSecret for your shop and the Runtime License Key(you can find it in the readMe txt file of the CData WooCommerce FireDAC Components folder) for your shop as strings. The following method sets up the connection variable and connects it with your shop database.

```
constructor TWooCommerceConnector.Create;
begin
  TConnection := TFDConnection.Create(nil);
end;

procedure TWooCommerceConnector.SetUpConnection(AShopURL, AUserName, APassword,  ARTK: String);
begin
  TConnection.ConnectionString := 'DriverId=CDataWooCommerce;AuthScheme=Basic;Url='+ AShopUrl +
                                  ';User=' + AUserName + ';Password='+ APassword +';RTK=' + ARTK + ';';
  try
    TConnection.Connected := True;
  finally
  end;
end;
```

Creating Items:

In order to create a new item on your WooCommerce store you can call the CreateItem method and pass the Sku, Name, Price, Desc and Category fields as strings along with Quantity as an Integer. Please note that the SKU is unique, you can't create 2 items with the same SKU. In order to check if an item exists you can call the getId and pass the SKU as a parameter. If Id returned is 0 then an item with a SKU doesn't exist and you can create it.

```
procedure TWooCommerceConnector.CreateItem(ASku, AName, APrice, ADesc, ACategory: String; AQuantity: Integer);
begin
  var
    vQuery: TFDQuery;
  begin
    vQuery := TFDQuery.Create(nil);
    try
      vQuery.Connection := TConnection;
      vQuery.SQL.Text :=
        'INSERT INTO Products (SKU,RegularPrice,ManageStock ,StockQuantity,ShortDescription,Name,CategoriesAggregate) '
        + 'VALUES (:sku,:price,True,:quantity, :desc,:name, :category)';
      vQuery.ParamByName('sku').AsString := ASku;
      vQuery.ParamByName('price').AsString := APrice;
      vQuery.ParamByName('quantity').AsInteger := AQuantity;
      vQuery.ParamByName('desc').AsString := ADesc;
      vQuery.ParamByName('name').AsString := AName;
      vQuery.ParamByName('category').AsString := GetCategoryJSON(ACategory);
      vQuery.ExecSQL;
    finally
      vQuery.Free;
    end;
  end;
end;
```

Updating Items:

In order to update an item you have 3 methods. You can only update the price by calling the method UpdatePrice and passing the Id and new price of the item or you can update the quantity by calling the method UpdateQuantity and passing the Id and new stock quantity of the item. Lastly, you can update the name, price, description, category, price and quantity of an item by passing the items mentioned before respectively. Note that we can't use the SKU to perform update statements instead you can only use the Id. To get the Id of a product, you can call the GetId method and pass the SKU as a parameter.

```
function TWooCommerceConnector.GetId(ASku: string):Integer;
var
  vQuery: TFDQuery;
  vId: Integer;
begin
    vQuery := TFDQuery.Create(nil);
    vQuery.Connection := TConnection;
    vQuery.SQL.Text := 'Select Id FROM Products WHERE Sku=:sku';
    vQuery.ParamByName('SKU').AsString := ASku;
    vQuery.Open;
    vId := vQuery.FieldByName('id').AsInteger;
    vQuery.Free;
    Result := vId;
end;
```

```
procedure TWooCommerceConnector.UpdatePrice(AId: Integer; APrice: String);
var
  vQuery: TFDQuery;
begin
  vQuery := TFDQuery.Create(nil);
  try
    vQuery.Connection := TConnection;
    vQuery.SQL.Text := 'UPDATE Products SET RegularPrice = :NewPrice WHERE Id = :id';
    vQuery.ParamByName('NewPrice').AsString := APrice;
    vQuery.ParamByName('id').AsInteger := AId;
    vQuery.ExecSQL;
  finally
    vQuery.Free;
  end;
end;

procedure TWooCommerceConnector.UpdateQuantity(AId, AQuantity: Integer);
var
  vQuery: TFDQuery;
begin
  vQuery := TFDQuery.Create(nil);
  try
    vQuery.Connection := TConnection;
    vQuery.SQL.Text := 'UPDATE Products SET StockQuantity = :newQuantity WHERE Id = :id';
    vQuery.ParamByName('newQuantity').AsInteger := AQuantity;
    vQuery.ParamByName('id').AsInteger := AId;
    vQuery.ExecSQL;
  finally
    vQuery.Free;
  end;
end;
```

```
procedure TWooCommerceConnector.UpdateItem(AName, APrice, ADesc, ACategory: String; AId, AQuantity: Integer);
begin
var
  vQuery: TFDQuery;
  begin
    vQuery := TFDQuery.Create(nil);
    try
      vQuery.Connection := TConnection;
      vQuery.SQL.Text := 'UPDATE Products SET Name = :name, RegularPrice = :price, StockQuantity = :quantity,' +
                         ' ShortDescription = :desc, CategoriesAggregate = :category WHERE Id = :id';
      vQuery.ParamByName('id').AsInteger := AId;
      vQuery.ParamByName('price').AsString := APrice;
      vQuery.ParamByName('quantity').AsInteger := AQuantity;
      vQuery.ParamByName('desc').AsString := ADesc;
      vQuery.ParamByName('name').AsString := AName;
      vQuery.ParamByName('category').AsString := GetCategoryJSON(ACategory);
      vQuery.ExecSQL;
    finally
      vQuery.Free;
    end;
  end;
end;
```

Deleting Products:

In order to delete a product you also need the Id of the product. Note that you can only delete one item at a time and you can't do a 'DELETE * FROM Products' SQL Statement. There are 2 methods inside the class for deleting. The first one is DeleteOne which takes an Id and deletes the item associated with the Id. The other method is DeleteAll which retrieves all Ids found in the Products Table, iterates over them and deletes them one at a time.

```
procedure TWooCommerceConnector.DeleteOne(AId: Integer);
var
  vQuery: TFDQuery;
begin
  vQuery := TFDQuery.Create(nil);
  try
    vQuery.Connection := TConnection;
    vQuery.SQL.Text := 'DELETE FROM Products WHERE Id= :id';
    vQuery.Params[0].AsInteger := AId;
    vQuery.ExecSQL;
  finally
    vQuery.Free;
  end;
end;

procedure TWooCommerceConnector.DeleteAll;
var
  vQuery: TFDQuery;
  vQuery2: TFDQuery;
begin
  vQuery := TFDQuery.Create(nil);
  vQuery2 := TFDQuery.Create(nil);
  try
    vQuery.Connection := TConnection;
    vQuery2.Connection := TConnection;
    vQuery.SQL.Text := 'SELECT Id FROM Products';
    vQuery.Open;
    while not VQuery.Eof do begin
      vQuery2.SQL.Text := 'DELETE FROM Products WHERE Id= :id';
      vQuery2.Params[0].AsInteger := vQuery.FieldByName('id').AsInteger;
      vQuery2.ExecSQL;
      vQuery.Next;
    end;
  finally
    vQuery.Free;
    vQuery2.Free;
  end;
end;
```

Retrieving Latest Orders:

In order to get the latest orders that haven't been updated locally, you can call the function GetLastestUpdated and pass in the date and time it was last updated in TDateTime format. The function will then iterate through all products that have been updated after this date and inside the function you can add code to perform the needed actions. You can also get the products that were in an order separately by calling the GetLatestOrderLineModified function.

```
procedure TWooCommerceConnector.GetLatestModified(DateLastUpdated: TDateTime);
begin
  var
    vQuery: TFDQuery;
  begin
    vQuery := TFDQuery.Create(nil);
    try
      vQuery.Connection := TConnection;
      vQuery.SQL.Text := 'SELECT * FROM Orders WHERE DateModified > :dateModified';
      vQuery.ParamByName('dateModified').AsDateTime := DateLastUpdated;
      vQuery.Open;
//      while not vQuery.Eof do
//      begin
//        // iterate over results one at a time and do specific code such as write to local db
//        vQuery.Next;
//      end;
    finally
      vQuery.Free;
    end;
  end;
end;
```

```
procedure TWooCommerceConnector.GetLatestOrderLineModified(DateLastUpdated: TDateTime);
begin
  var
    vQuery: TFDQuery;
  begin
    vQuery := TFDQuery.Create(nil);
    try
      vQuery.Connection := TConnection;
      vQuery.SQL.Text := 'SELECT * FROM OrderLineItems WHERE OrderId IN (SELECT Id FROM Orders WHERE DateModified > :dateModified)';
      vQuery.ParamByName('dateModified').AsDateTime := DateLastUpdated;
      vQuery.Open;
//      while not vQuery.Eof do
//      begin
//        // iterate over results one at a time and do specific code such as write to local db
//        vQuery.Next;
//      end;
    finally
      vQuery.Free;
    end;
  end;
end;
```

Understanding Order Status:

Payment Method / Order Status	Cash On Delivery	Online Payment
Completed	Paid and delivered (Delivered Invoice)	Paid and delivered (Delivered Invoice)
Processing	To be delivered and paid (Order Invoice)	Paid to be delivered (Not delivered Invoice)