eCommerce Innovation. Not Infrastructure.

Blueshift Integration User Guide 16.1.0

Purpose of this document

The purpose of this document is to outline a general concept which describes the workflow, configuration and logging view of the Blueshift schedule jobs and event setup.

Version History

Version	Date
Version 16.1.0	01/08/2016

Table of contents

Cartridge Installation

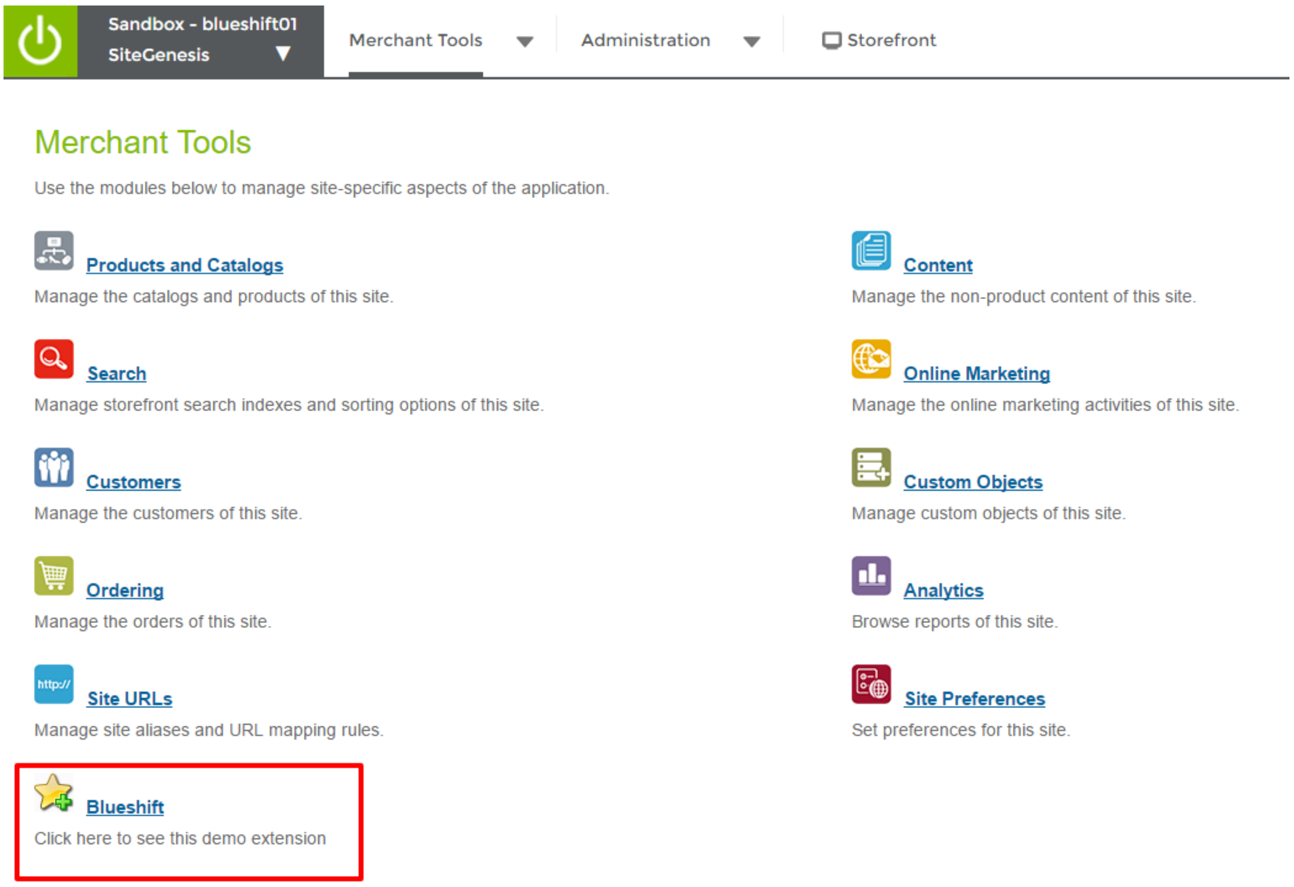
To install the Blueshift integration for the first time on your instance, you may follow the following steps:

1. Download blushiftintigration.zip from Demandware XChange
2. Extract archive to your local file system – e.g. the cartridge folder of your project
3. Import cartridges into your workspace and link them to the Server Connection
4. Import site_template.zip into your instance (Meta Data andCustom Objects and Schedules)
5. Assign the int_blueshift cartridge to the organization and all sites.
6. Assign the bm_blueshiftand int_blueshift cartridges to the business manager organization – e.g. bm_blueshift:bm_custom_plugin
7. Assign Business Manager Modules ‘Blueshift‘ and its sub modules like **‘Global Configurations’**, **‘Customers’**, **‘Catalog’**, **‘Orders’** to respective roles
8. You can now make use of the cartridge

Note : bc_library needs to be included while setting up the blueshift cartridge.

Configuration

Under Merchant Tools -> Blueshift you can configure global configuration settings , customers, order and catalog export configuration:


Sandbox - blueshift01
SiteGenesis
Merchant Tools
Administration
Storefront

Merchant Tools

Use the modules below to manage site-specific aspects of the application.

Products and Catalogs
Manage the catalogs and products of this site.

Content
Manage the non-product content of this site.

Search
Manage storefront search indexes and sorting options of this site.

Online Marketing
Manage the online marketing activities of this site.

Customers
Manage the customers of this site.

Custom Objects
Manage custom objects of this site.

Ordering
Manage the orders of this site.

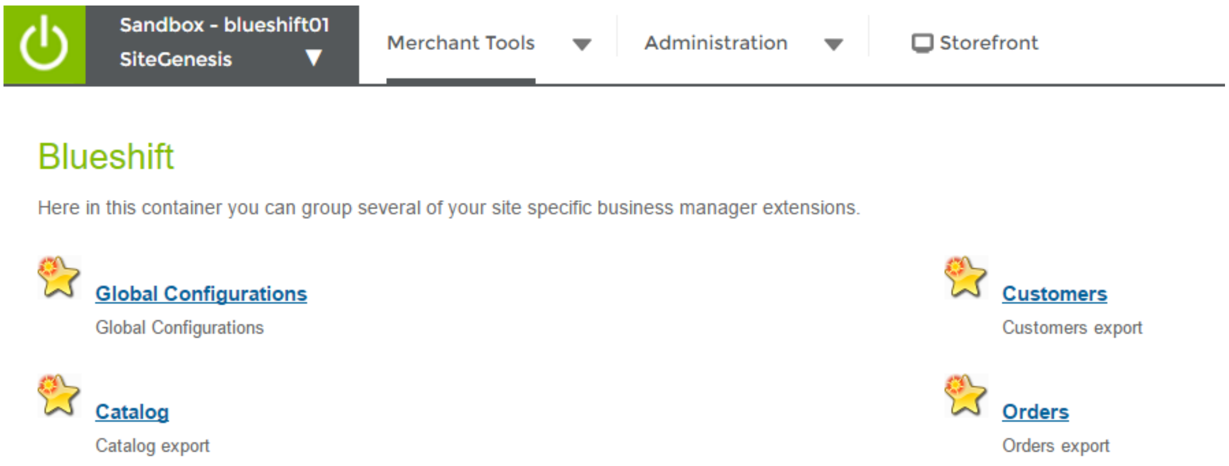
Analytics
Browse reports of this site.

Site URLs
Manage site aliases and URL mapping rules.

Site Preferences
Set preferences for this site.

Blueshift
Click here to see this demo extension

Under the Blushift menu you can see other options like Global Configuration, Customers ,Catalog , Orders.


Sandbox - blueshift01
SiteGenesis
Merchant Tools
Administration
Storefront

Blueshift

Here in this container you can group several of your site specific business manager extensions.

Global Configurations
Global Configurations

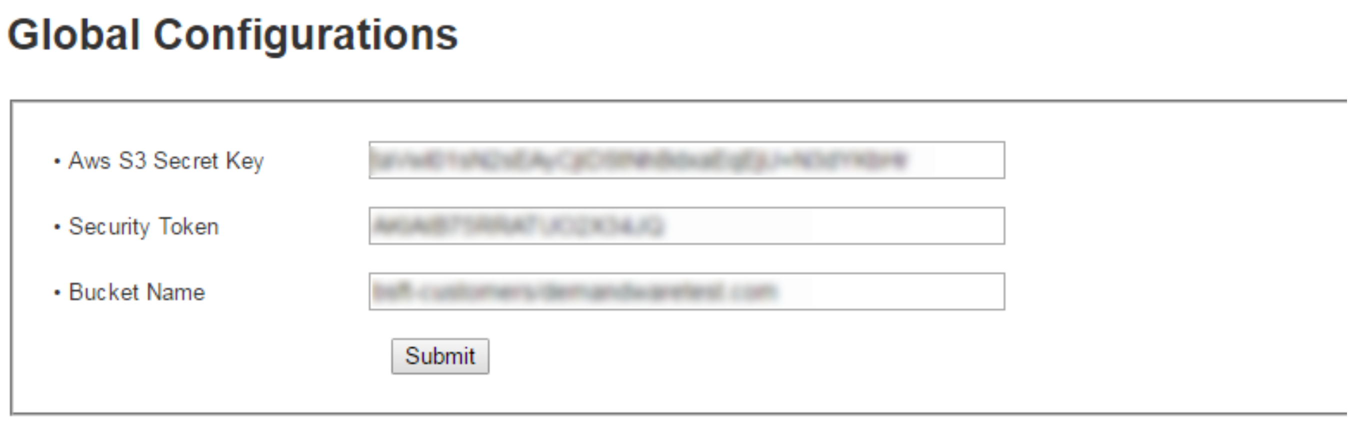
Customers
Customers export

Catalog
Catalog export

Orders
Orders export

Global Configuration

Here is the global configuration which is needed to connect with AWS S3 bucket or Blueshift API. From here you can configure API configurations.



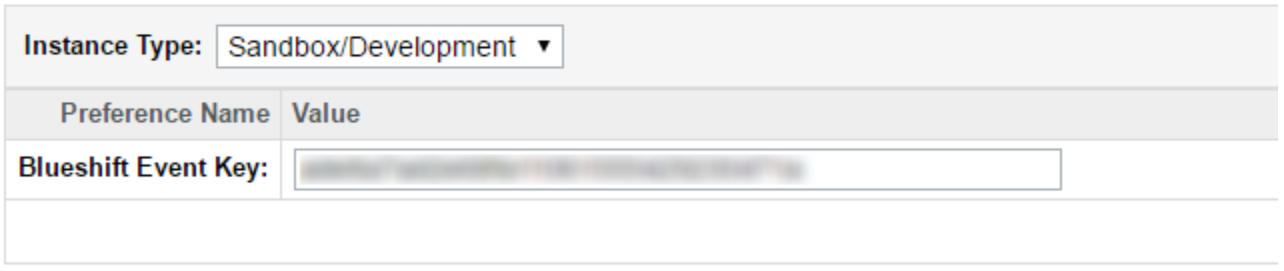
Global Configurations

• Aws S3 Secret Key	[illegible]
• Security Token	[illegible]
• Bucket Name	[illegible]

Blueshift Event Key Configuration

For configuring blueshift event key, go to the Merchant Tools -> custom Preferances select "BlueshiftEventConfigs" and add the Blueshift Event key over there.

Bellow is the screenshot for reference :



Instance Type: Sandbox/Development

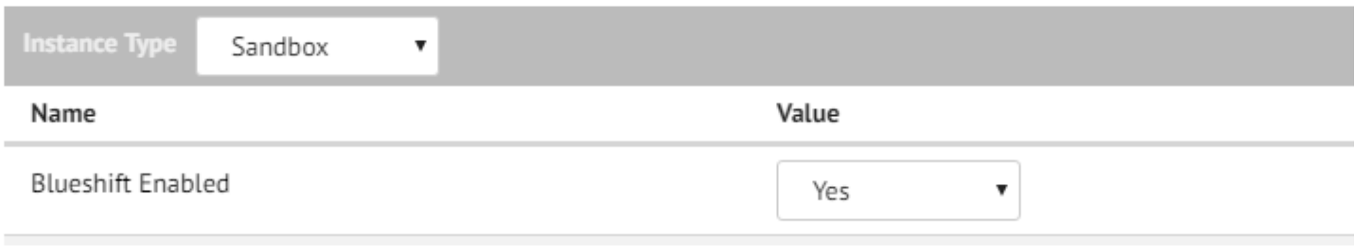
Preference Name	Value
Blueshift Event Key:	[illegible]

Blueshift Enable/Disable module :

this functionality has feature to enable or disable blueshift event hooks and scadule jobs to execute.
user can enable or disable blueshift module functionality from here.
if the blueshift is disabled, all the frontend hooks like viewproduct, identify user etc. will be disabled no data is passed to blueshift and schedule jobs are also disabled means if any job is run while disable mode, no csv file generated and nothing is uploaded to AWS S3 bucket.

For configuring blueshift enable/disable mode, go to the Merchant Tools -> custom Preferences select "BlueshiftEventConfigs" and you can see Blueshift Enabled you can change the value to Yes or No .

Bellow is the screenshot for reference.

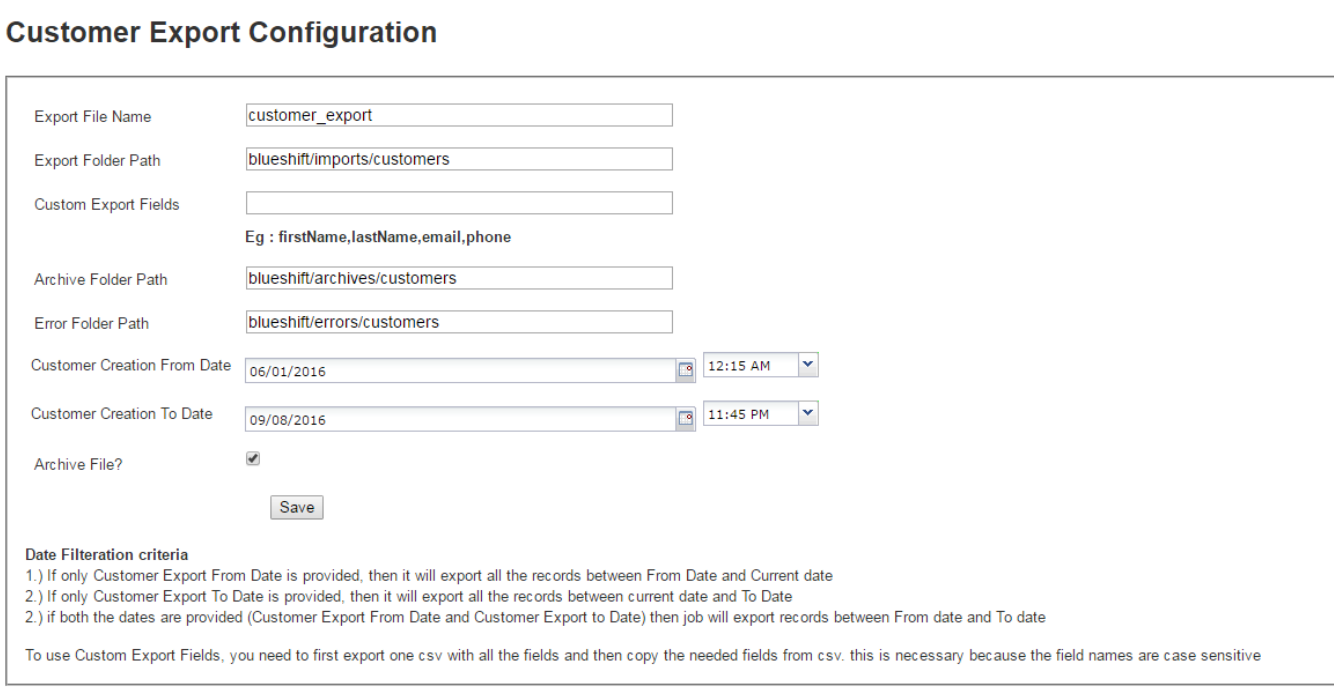


Instance Type: Sandbox

Name	Value
Blueshift Enabled	Yes

Customers Configuration

You can configure customer export configuration from here. There are option for configurations.



Customer Export Configuration

Export File Name	customer_export
Export Folder Path	blueshift/imports/customers
Custom Export Fields	
	Eg : firstName,lastName,email,phone
Archive Folder Path	blueshift/archives/customers
Error Folder Path	blueshift/errors/customers
Customer Creation From Date	06/01/2016 12:15 AM
Customer Creation To Date	09/08/2016 11:45 PM
Archive File?	✓

Date Filteration criteria
1.) If only Customer Export From Date is provided, then it will export all the records between From Date and Current date
2.) If only Customer Export To Date is provided, then it will export all the records between current date and To Date
2.) if both the dates are provided (Customer Export From Date and Customer Export to Date) then job will export records between From date and To date

To use Custom Export Fields, you need to first export one csv with all the fields and then copy the needed fields from csv. this is necessary because the field names are case sensitive

Export File Name : name of the file tobe exported
Export Folder Path : here you can set the path of export folder it will be created under IMPEX folder.

Custom Export Fields : you can specify the specific comma separated fields. To export for csv. If you want all the possible fields to be exported, then leave this field as blank.
For e.g. : firstName,lastName,lastLoginTime,phoneBusiness

Archive Folder Path :here you can set the path of archive folder. it will be created under IMPEX directory.

Error Folder Path: here you can set the path of error folder. it will be created under IMPEX folder. If any error uccares meanwhile the uploading or creation of csv file then that file will be moved to error folder for future referance.

Customer Creation From Date :this field will allow you to filter the customer records by creation date. If you enter value to this field then it will bring all the records between specified date and current date.

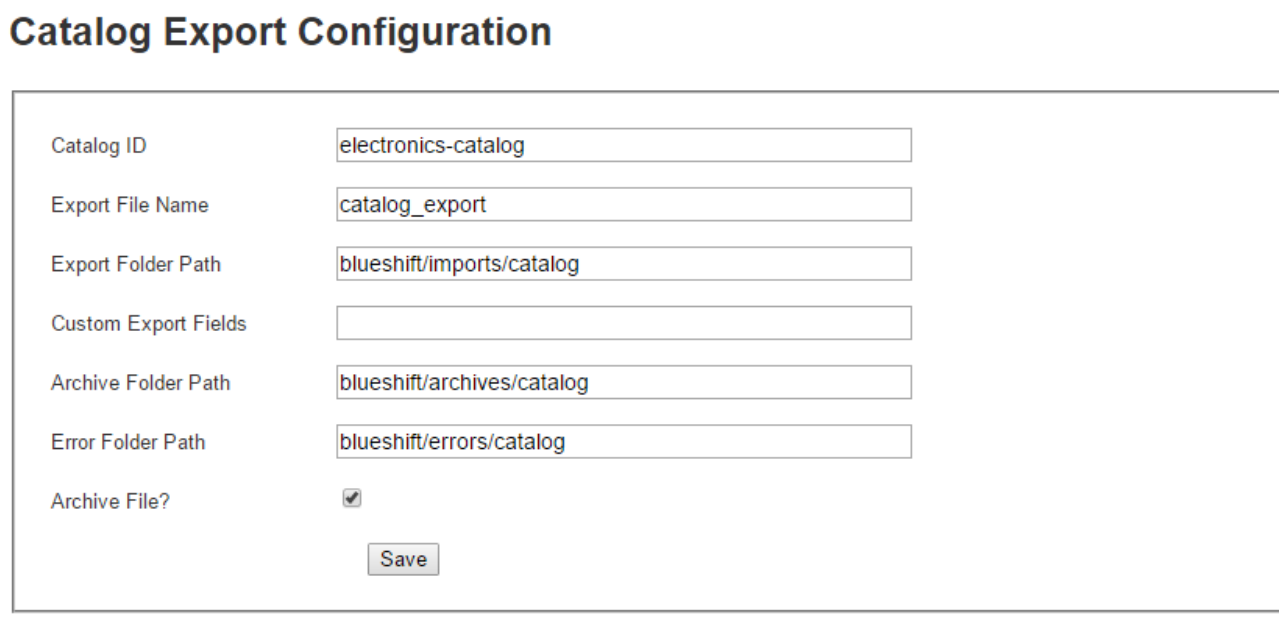
Customer Creation To Date :this field will allow you to filter the customer records by creation date. If you enter value to this field then it will bring all the records between current date and Creation To Date.

NOTE : if you provide both the dates like CreationFromDate and CreationToDate then it will fetch all the records between both the given dates.

Archive File : if this function is set to checked then the exported file will be moved to archive folder after uploading to S3. And if its unchecked then the exported file will be deleted permanently.

Catalog Configuration :

Here is the configuration for catalog exports. You can find this configuration under Blushift menu.


Catalog Export Configuration

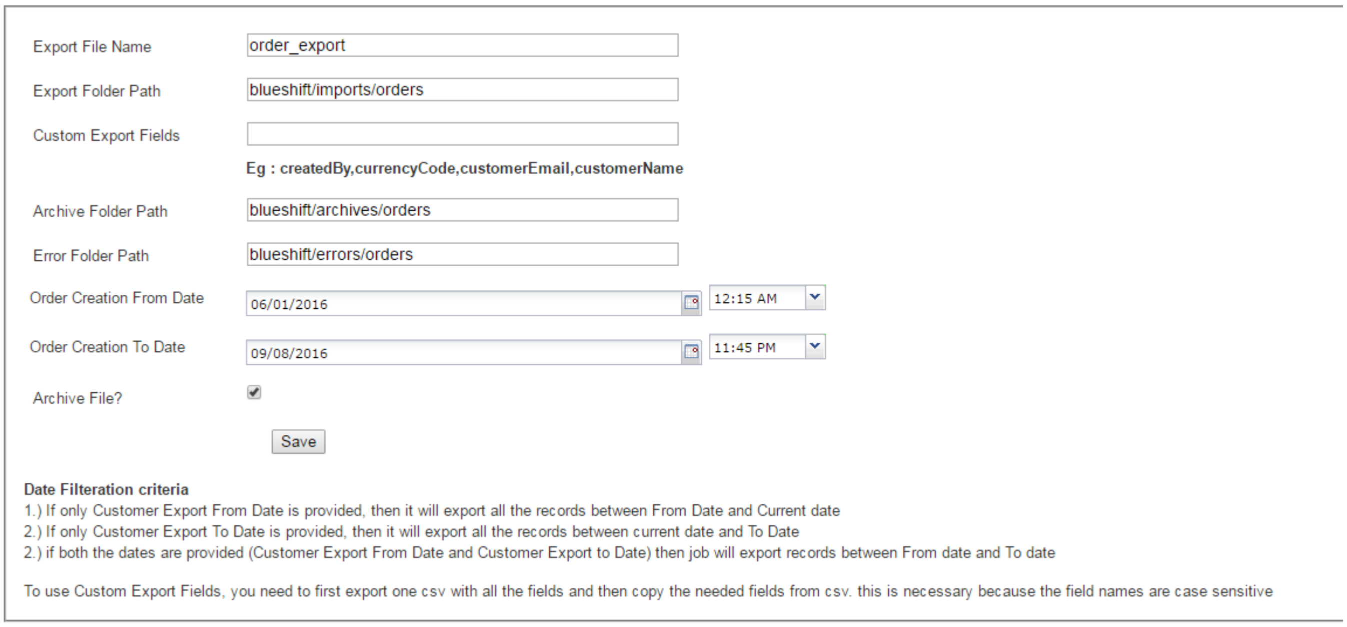
Catalog ID	electronics-catalog
Export File Name	catalog_export
Export Folder Path	blueshift/imports/catalog
Custom Export Fields	
Archive Folder Path	blueshift/archives/catalog
Error Folder Path	blueshift/errors/catalog
Archive File?	✓

Catalog ID : here you need to set catalog id, from which you want to export products. The other fields will work same as explained above.

Order Configuration :

Here is the configuration for Order exports. You can find this configuration under Blushift menu.

Order Export Configuration



Export File Name	order_export
Export Folder Path	blueshift/imports/orders
Custom Export Fields	
	Eg : createdBy,currencyCode,customerEmail,customerName
Archive Folder Path	blueshift/archives/orders
Error Folder Path	blueshift/errors/orders
Order Creation From Date	06/01/2016 12:15 AM
Order Creation To Date	09/08/2016 11:45 PM
Archive File?	✓

Date Filteration criteria
1.) If only Customer Export From Date is provided, then it will export all the records between From Date and Current date
2.) If only Customer Export To Date is provided, then it will export all the records between current date and To Date
2.) if both the dates are provided (Customer Export From Date and Customer Export to Date) then job will export records between From date and To date

To use Custom Export Fields, you need to first export one csv with all the fields and then copy the needed fields from csv. this is necessary because the field names are case sensitive

The configuration fields will work same as explained above in customer configuration area.

Schedule job Configuration :

You can schedule the jobs for exporting Customers, catalog and orders from Administrator -> job schedules.

Steps :

1.) Click on new button in job schedules.
2.) Add the appropriate field values like name , description etc.
3.) Set Execution Scope to “Sites” and assign the Site from Sites tab.
4.) Set Starting node and pipeline for particular job.
Here bellow is the pipeline and start node configuration for all three jobs

Export type	**Pipeline**	**Start Node**
Customer Export	Blueshift	InitCustomerExport
Catalog Export	Blueshift	InitCatalogExport
Order Export	Blueshift	InitOrdersExport

5.) You can manually run the job by clicking on Run button and you can scadule the job for execution on specific day or time.

Data model and related infrastructure

BlueshiftAwsS3Credentials	
Attribute/Method	Description
ID (PK)	A unique ID which represents the schedule
secretKey	Input text for AWS secret key value
accessKey	Input text for AWS access key value
bucketName	Input text for AWS S3 bucket name
executionTime	The time this workflow shall be executed
blueshiftEventKey	Input field for blueshift api event key

BlueshiftCatalogExport	
Attribute/Method	Description
ID (PK)	A unique ID which represents the schedule
catalogID	Input field for Catalog ID
exportFileName	Sets the export file name for csv file
foldername	Sets folder path for exported csv
fieldnames	Input for custom specific fields to export
archivefolder	Sets archive folder path for uploaded csv files
errorfolder	Sets error folder path if csv upload fails in any reason then it will be shifted to error folder path.
isArchive	Boolean check for archive csv files or not.

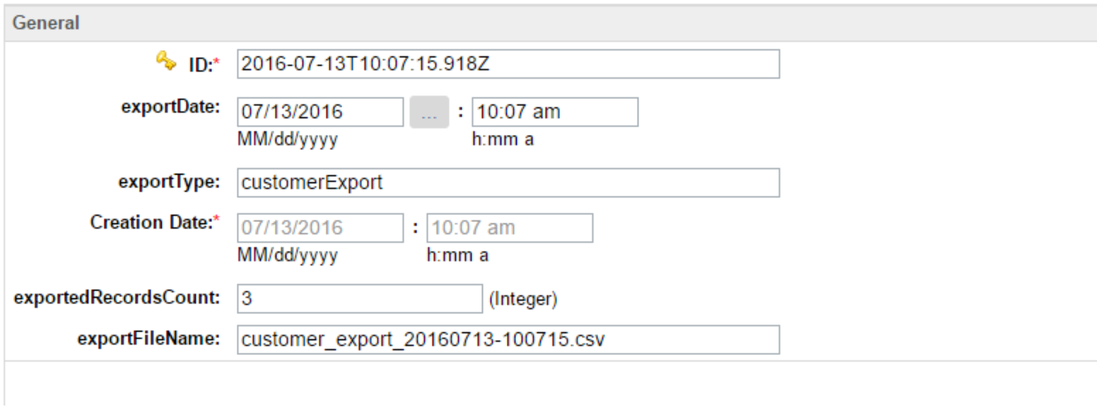
BlueshiftCsvExportLogs	
Attribute/Method	Description
id (PK)	A unique ID which represents the schedule
exportDate	It will hold the last exported csv 's export date
exportType	It will hold the export type like catalogExport, orderExport or CustomerExport
creationDate	Auto generated date for new entry created for this custom object.
exportedRecordsCount	Holds the number of exported record count from csv.
exportFileName	Saves the filename which is generated in last export execution.

BlueshiftCustomerCsvExport	
Attribute/Method	Description
ID (PK)	A unique ID which represents the schedule
exportFileName	Sets the export file name for csv file
foldername	Sets folder path for exported csv
fieldnames	Input for custom specific fields to export
archivefolder	Sets archive folder path for uploaded csv files
errorfolder	Sets error folder path if csv upload fails in any reason then it will be shifted to error folder path.
isArchive	Boolean check for archive csv files or not.
customerCreationToDate	Date time field for customer creation to date
customerCreationFromDate	Date time field for customer creation from date

BlueshiftOrderExport	
Attribute/Method	Description
ID (PK)	A unique ID which represents the schedule
exportFileName	Sets the export file name for csv file
foldername	Sets folder path for exported csv
fieldnames	Input for custom specific fields to export
archivefolder	Sets archive folder path for uploaded csv files
errorfolder	Sets error folder path if csv upload fails in any reason then it will be shifted to error folder path.
isArchive	Boolean check for archive csv files or not.
OrderCreationToDate	Date time field for Order creation to date
OrderCreationFromDate	Date time field for Order creation from date

Export Logs

You can see the exported file and its number of records logs in custom object editor section.
Go to merchant tools -.> custom object editor   select BlueshiftCsvExportLogs.
Here you can see all the exports.
There are three type of exports saved in it . CustomerExports, CatalogExport, OrderExport.



General

ID:*	2016-07-13T10:07:15.918Z
exportDate:	07/13/2016 : 10:07 am (MM/dd/yyyy h:mm a)
exportType:	customerExport
Creation Date:*	07/13/2016 : 10:07 am (MM/dd/yyyy h:mm a)
exportedRecordsCount:	3 (Integer)
exportFileName:	customer_export_20160713-100715.csv

eCommerce Innovation. Not Infrastructure.

Blueshift Storefront Integration User Guide

Storefront Integration

Blueshift provides event stream API to receive real-time data from website like user details, product, orders etc. for this blueshift is providing JavaScript API to handle all these event calls.

To use the frontend events we need “event-stream API key” you can find this API key at following location. Login to your blueshift account, go to Account Profile and you can find API Key tab.

Events

Blueshift provides various events like init, viewProduct, addToCart etc. blueshift is also allowing your custom events also, you can learn it from blueshift’s official document.

Here is the list of events we are using.

1.) pageload
2.) identifyUser
3.) viewProduct
4.) addToCart
5.) removeFromCart
6.) checkoutEvent
7.) purchase
8.) search

you can find all these event hooks in “blueshift.eventhooks.js” file.

JavaScript Configurations :

Before start implementing events on the site. First of all place the necessary javascript hooks in "app.js" file. you can find the code for app.js file in 'int_blueshift/cartridge/static/default/js/app.js'

Example code :

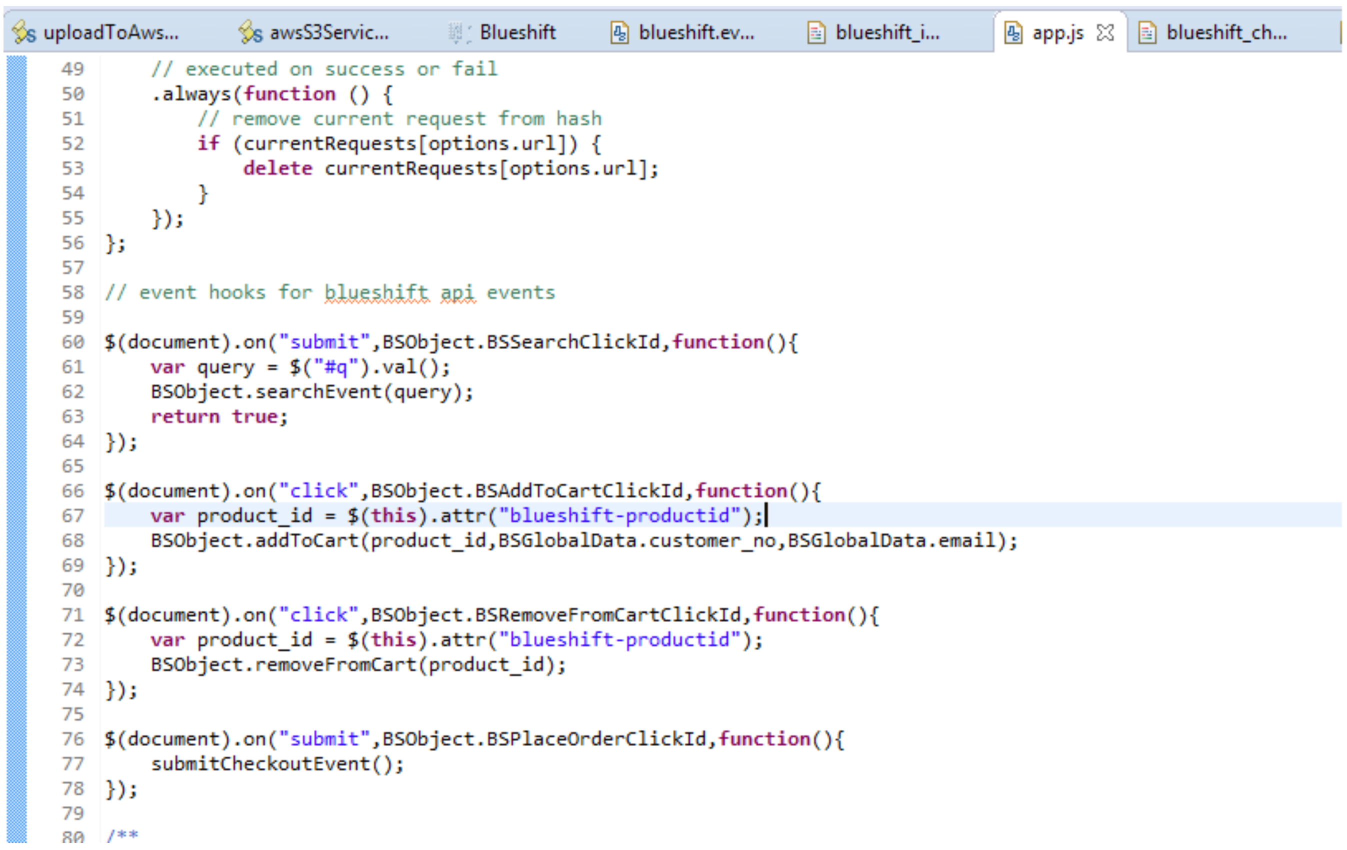
```
$(document).on("submit",BSObject.BSSearchClickId,function(){
      var query = $("#q").val();
      BSObject.searchEvent(query);
      return true;
});

$(document).on("click",BSObject.BSAddToCartClickId,function(){
      var product_id = $(this).attr("blueshift-productid");
      alert(product_id);
      BSObject.addToCart(product_id,BSGlobalData.customer_no,BSGlobalData.ema
il);
});

$(document).on("click",BSObject.BSRemoveFromCartClickId,function(){
      var product_id = $(this).attr("blueshift-productid");
      BSObject.removeFromCart(product_id);
});

$(document).on("submit",BSObject.BSPlaceOrderClickId,function(){
      submitCheckoutEvent();
});
```

After adding this code to app.js, Open the **"blueshift.eventhooks.js"**file. here we need to set the following variables.

BSSearchClickId	: "#searchEvent",
BSAddToCartClickId	: ".add-to-cart",
BSRemoveFromCartClickId	: ".remove-item",
BSPlaceOrderClickId	: ".submit-order",

These are the event identifier like for example BSAddToCartClickId which represents the class name of add to cart button. So whenever the button holding this class name is clicked, our add to cart event will be fired.

Screenshot :



```
    // executed on success or fail
    .always(function () {
        // remove current request from hash
        if (currentRequests[options.url]) {
            delete currentRequests[options.url];
        }
    });
};

// event hooks for blueshift api events

$(document).on("submit",BSObject.BSSearchClickId,function(){
    var query = $("#q").val();
    BSObject.searchEvent(query);
    return true;
});

$(document).on("click",BSObject.BSAddToCartClickId,function(){
    var product_id = $(this).attr("blueshift-productid");
    BSObject.addToCart(product_id,BSGlobalData.customer_no,BSGlobalData.email);
});

$(document).on("click",BSObject.BSRemoveFromCartClickId,function(){
    var product_id = $(this).attr("blueshift-productid");
    BSObject.removeFromCart(product_id);
});

$(document).on("submit",BSObject.BSPlaceOrderClickId,function(){
    submitCheckoutEvent();
});

/**
```

Note : the above code is only for reference example. You can set your own classes or ids for event identifier. If you are using your own classes or ids then make sure the ids or classes are also added to the related buttons or links. But the identifiers must be set, blank values will raise unexpected errors.

NOTE 1: if you want to add your custom events or add more events which blueshift is providing, then you can just modify this file and add new event functions in it. And that event can be used in frontend using “BSObject.your event function”.

NOTE 2 : all the hooks which we are used for different events, needs to be placed between <isdecorate></isdecorate> tags. As our core javascript library is loaded in header so we need to first load that library and then our hooks will work.

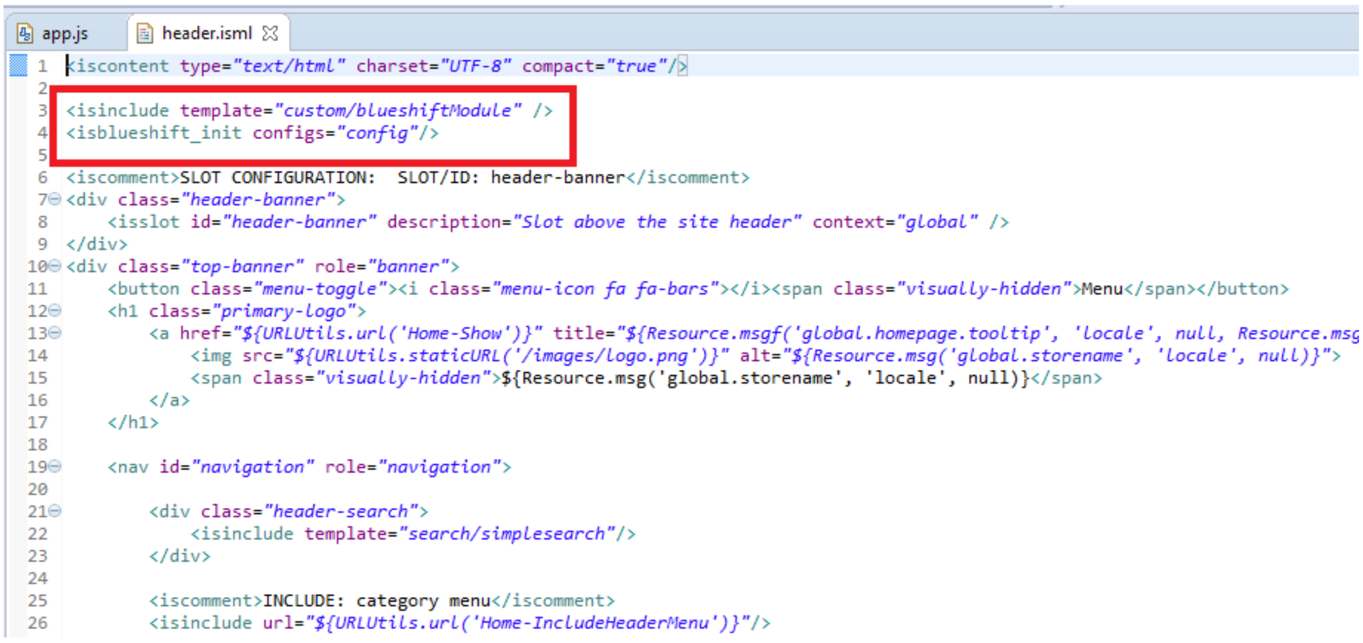
Pageload event :

Fire this event on every page of your site. Code for this event is placed in header.isml so that this event is available in all page load. The bellow is the code snipet which needs to be placed in header.isml file.

Code Snipet :

```
<isinclude template="custom/blueshiftModule" />
<isblueshift_init configs="config"/>
```

Screenshot :



```
<iscontent type="text/html" charset="UTF-8" compact="true"/>

<isinclude template="custom/blueshiftModule" />
<isblueshift_init configs="config"/>

<iscomment>SLOT CONFIGURATION:  SLOT/ID: header-banner</iscomment>
<div class="header-banner">
    <isslot id="header-banner" description="Slot above the site header" context="global" />
</div>
<div class="top-banner" role="banner">
    <button class="menu-toggle"><i class="menu-icon fa fa-bars"></i><span class="visually-hidden">Menu</span></button>
    <h1 class="primary-logo">
        <a href="${URLUtils.url('Home-Show')}" title="${Resource.msgf('global.homepage.tooltip', 'locale', null, Resource.msg
            <img src="${URLUtils.staticURL('/images/logo.png')}" alt="${Resource.msg('global.storename', 'locale', null)}">
            <span class="visually-hidden">${Resource.msg('global.storename', 'locale', null)}</span>
        </a>
    </h1>

    <nav id="navigation" role="navigation">

        <div class="header-search">
            <isinclude template="search/simplesearch"/>
        </div>

        <iscomment>INCLUDE: category menu</iscomment>
        <isinclude url="${URLUtils.url('Home-IncludeHeaderMenu')}"/>
```

This will load the blueshift.eventhooks.js file and call the “init()” event.
This event loads the core blueshift.js file and sends the pageload event to blueshift server.

Identify User Event :

Fire this event when you can identify the user that has logged in, or when you need to pass a user attribute for a known user. You must include a unique identifying parameter like customer_id or email.

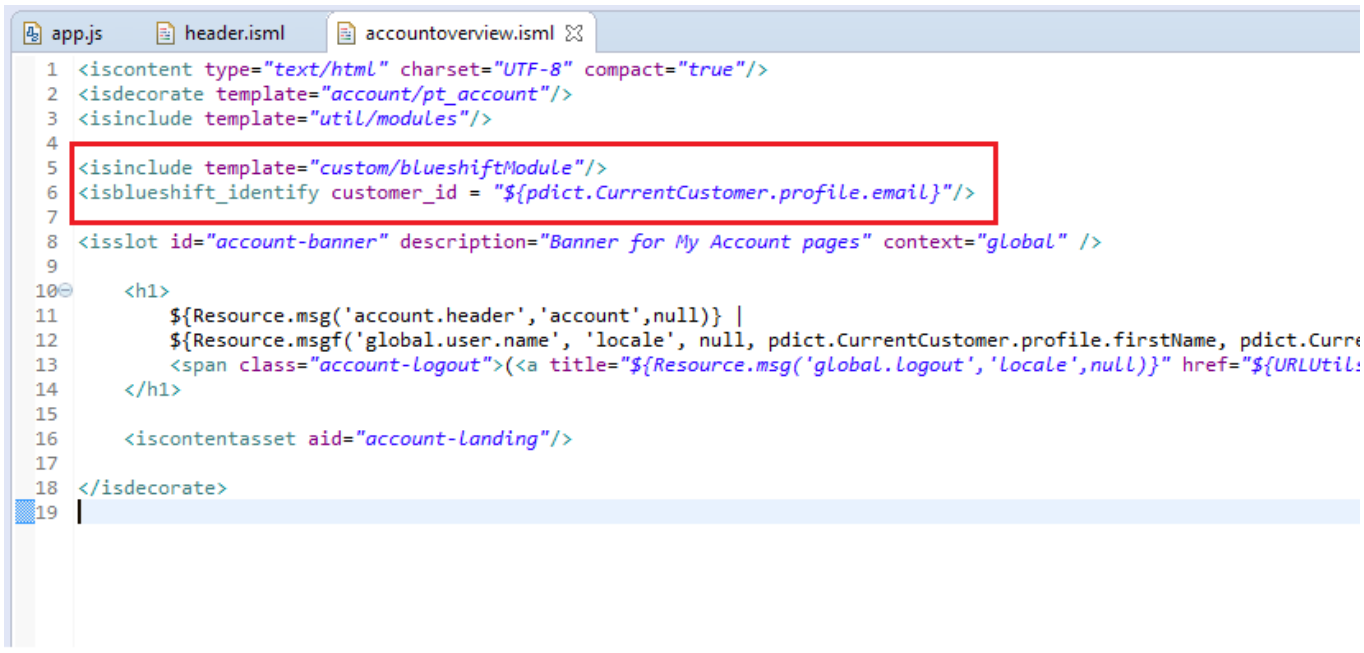
Generally we put this event call in profile page or in checkout summary page because we can find all user details in these pages.

To call this event you need to add following code in related files. For e.g. summary.isml in checkout. or you can put it on account view page(accountoverview.isml), which will appear after registration or login page like shown in screenshot.

Code Snipet :

```
<isinclude template="custom/blueshiftModule"/>
<isblueshift_identify customer_id = "${pdict.Basket.getCustomerEmail()}"/>
```

Screenshot :



```
app.js    header.isml    accountoverview.isml

<iscontent type="text/html" charset="UTF-8" compact="true"/>
<isdecorate template="account/pt_account"/>
<isinclude template="util/modules"/>

<isinclude template="custom/blueshiftModule"/>
<isblueshift_identify customer_id = "${pdict.CurrentCustomer.profile.email}"/>

<isslot id="account-banner" description="Banner for My Account pages" context="global" />

    <h1>
        ${Resource.msg('account.header','account',null)} |
        ${Resource.msgf('global.user.name', 'locale', null, pdict.CurrentCustomer.profile.firstName, pdict.Curre
        <span class="account-logout">(<a title="${Resource.msg('global.logout','locale',null)}" href="${URLUtil
    </h1>

    <iscontentasset aid="account-landing"/>

</isdecorate>
```

Here we are passing unique Id as customer_id. The above code will add one isml file.
This file will call the javascript event hook “BSObject.identifyUser(params)”.
Params are the data we are supplying to the event.

Javascript Call for developer purpose to Identify User :

```
BSObject.identifyUser({
     customer_id:'${profile.customerNo}',
     email: '${profile.email}',
     joined_at: '${profile.creationDate.toISOString()}',
     firstname: '${profile.firstName}',
     lastname: '${profile.lastName}'
 });
```

You can view full code in “template/tag_events/identiry.isml” for better reference.

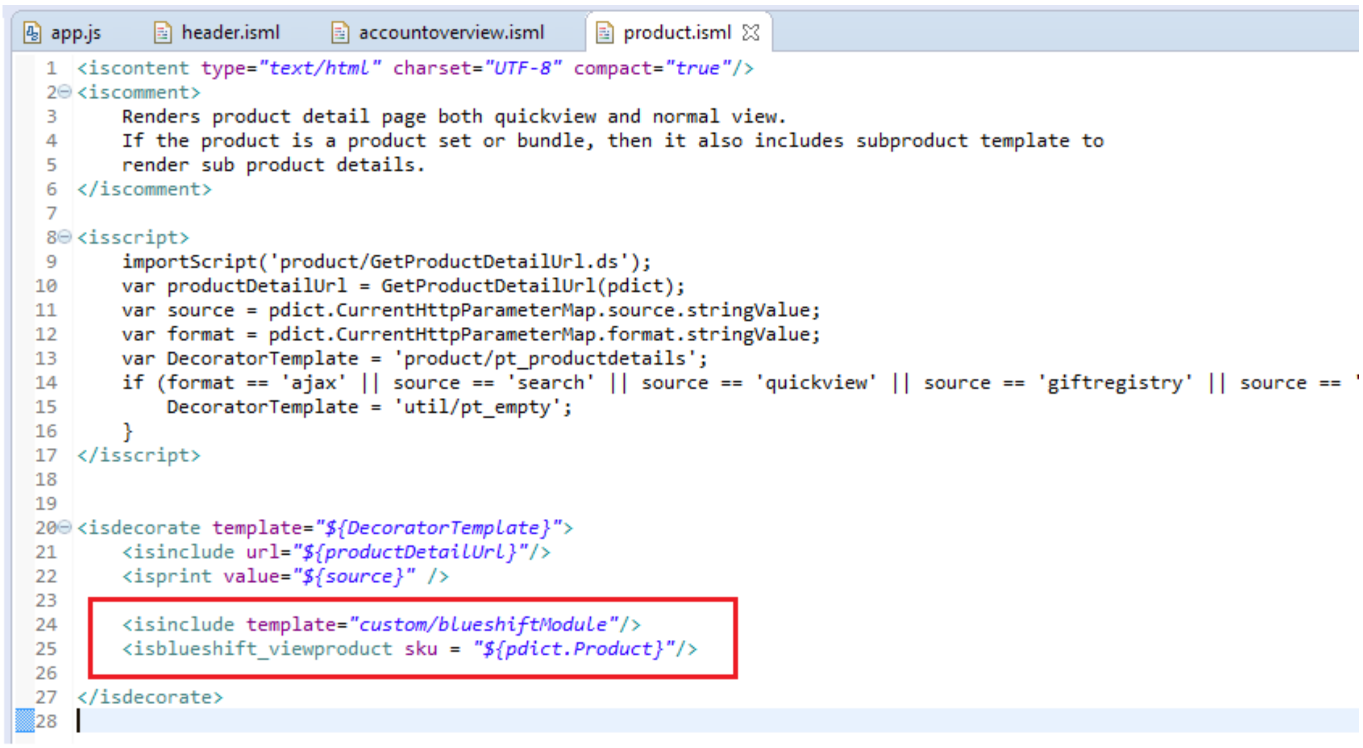
View Product Event :

This event will fire when the user views a product page on your site.
The hook for this event is placed in product.isml file. This file is the details view of the product.
To add this event bellow code snipet needs to be placed in product.isml file.

Code Snipet :

```
<isinclude template="custom/blueshiftModule"/>
<isblueshift_viewproduct sku = "${pdict.Product}"/>
```

Screenshot :



```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<iscomment>
    Renders product detail page both quickview and normal view.
    If the product is a product set or bundle, then it also includes subproduct template to
    render sub product details.
</iscomment>

<isscript>
    importScript('product/GetProductDetailUrl.ds');
    var productDetailUrl = GetProductDetailUrl(pdict);
    var source = pdict.CurrentHttpParameterMap.source.stringValue;
    var format = pdict.CurrentHttpParameterMap.format.stringValue;
    var DecoratorTemplate = 'product/pt_productdetails';
    if (format == 'ajax' || source == 'search' || source == 'quickview' || source == 'giftregistry' || source == '
        DecoratorTemplate = 'util/pt_empty';
    }
</isscript>


<isdecorate template="${DecoratorTemplate}">
    <isinclude url="${productDetailUrl}"/>
    <isprint value="${source}" />

    <isinclude template="custom/blueshiftModule"/>
    <isblueshift_viewproduct sku = "${pdict.Product}"/>

</isdecorate>
```

This hook will add one isml file and from that isml it will call the following event.
(bellow reference code is for developer purpose only)

```
BSObject.viewProduct({sku:'${pdict.Product.getID()}'},'${customer}','${email}
');
```

You can view full code in “template/tag_events/viewproduct.isml” for better reference.

Here we are passing productID , Customer Details and Email address.If the customer is not logged in then it will pass only productID.

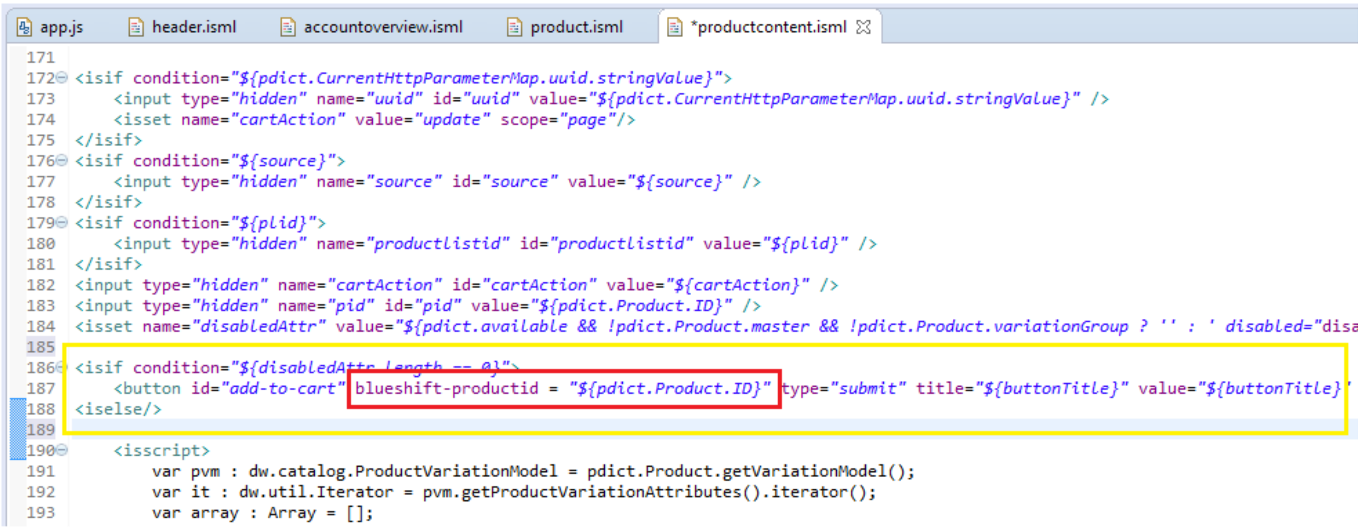
Add To Cart Event :

you can fire this event when the user adds an item to shopping cart. To add this event we need to add one click event in add to cart button.
You can find the add to cart button in “productcontent.isml”. but yes there will be more add to cart buttons in different files in a site, so just find the add to cart buttons and add the following attribute to button code.

blueshift-productid = "${pdict.Product.getID()"

for Example :
<button id="add-to-cart" **blueshift-productid = "${pdict.Product.getID()"** type="submit" title="${buttonTitle}" value="${buttonTitle}" class="button-fancy-large add-to-cart" >${buttonTitle}</button>

Screenshot :



```
<isif condition="${pdict.CurrentHttpParameterMap.uuid.stringValue}">
    <input type="hidden" name="uuid" id="uuid" value="${pdict.CurrentHttpParameterMap.uuid.stringValue}" />
    <isset name="cartAction" value="update" scope="page"/>
</isif>
<isif condition="${source}">
    <input type="hidden" name="source" id="source" value="${source}" />
</isif>
<isif condition="${plid}">
    <input type="hidden" name="productlistid" id="productlistid" value="${plid}" />
</isif>
<input type="hidden" name="cartAction" id="cartAction" value="${cartAction}" />
<input type="hidden" name="pid" id="pid" value="${pdict.Product.ID}" />
<isset name="disabledAttr" value="${pdict.available && !pdict.Product.master && !pdict.Product.variationGroup ? '' : ' disabled="disa

<isif condition="${disabledAttr.length == 0}">
    <button id="add-to-cart" blueshift-productid = "${pdict.Product.ID}" type="submit" title="${buttonTitle}" value="${buttonTitle}"
<iselse/>

    <isscript>
        var pvm : dw.catalog.ProductVariationModel = pdict.Product.getVariationModel();
        var it : dw.util.Iterator = pvm.getProductVariationAttributes().iterator();
        var array : Array = [];
```

after adding one extra attribute, this add to cart button will fire our add-to-cart click event, which we have configured in app.js file earlier.

following is the click event code for Developer reference only.

```
var product_id = $(this).attr("blueshift-productid");
BSObject.addToCart(product_id,BSGlobalData.customer_no,BSGlobalData.email);
```

You can view full code in “app.js” for better reference.

Here we are supplying product ID , customer_ID, and Email. But here also if user is not logged in then it will only supply product ID to blueshift.

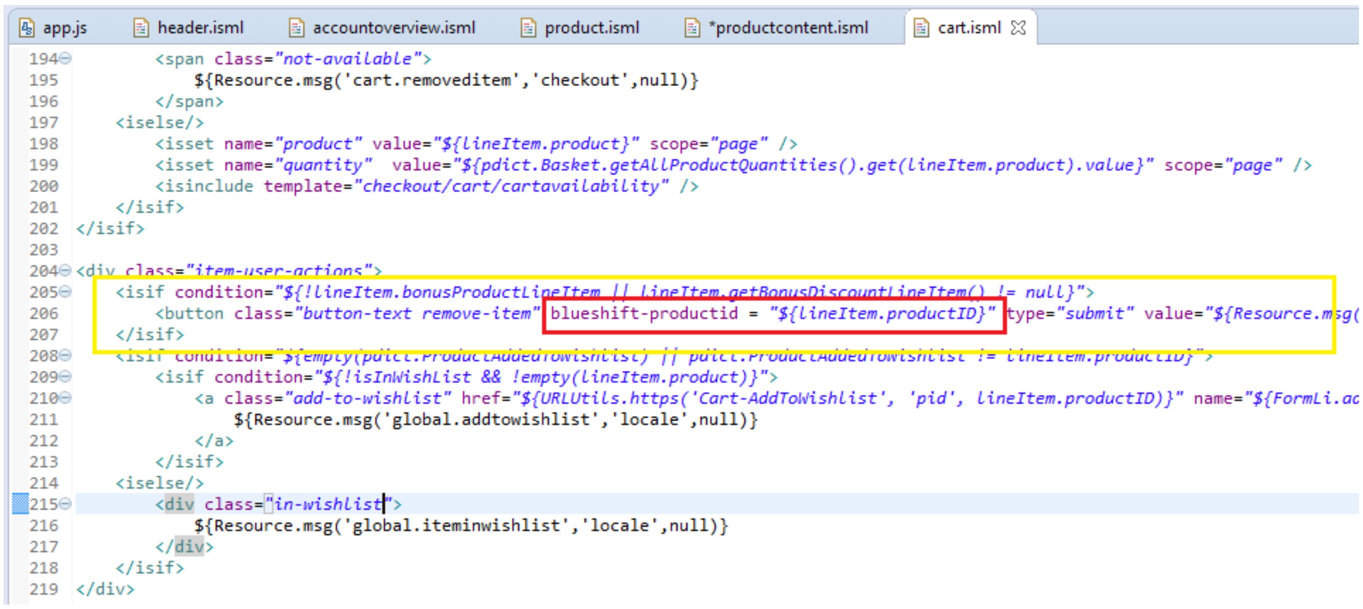
Remove From Cart Event :

Similar to Add To Cart event above, just add the same attribute to the remove cart button or link And whenever the link or button will be clicked our remove cart event will fire. You can find the remove item link in “cart.isml” file.

Following is code snipet:

blueshift-productid = "${pdict.Product.getID()"

Screenshot :



```
        <span class="not-available">
            ${Resource.msg('cart.removeditem','checkout',null)}
        </span>
    <iselse/>
        <isset name="product" value="${lineItem.product}" scope="page" />
        <isset name="quantity"  value="${pdict.Basket.getAllProductQuantities().get(lineItem.product).value}" scope="page" />
        <isinclude template="checkout/cart/cartavailability" />
    </isif>
</isif>

<div class="item-user-actions">
    <isif condition="${!lineItem.bonusProductLineItem || lineItem.getBonusDiscountLineItem() != null}">
        <button class="button-text remove-item" blueshift-productid = "${lineItem.productID}" type="submit" value="${Resource.msg(
    </isif>
    <isif condition="${empty(pdict.ProductAddedToWishList) || pdict.ProductAddedToWishList != lineItem.productID}">
        <isif condition="${!isInWishList && !empty(lineItem.product)}">
            <a class="add-to-wishlist" href="${URLUtils.https('Cart-AddToWishlist', 'pid', lineItem.productID)}" name="${FormLi.ac
                ${Resource.msg('global.addtowishlist','locale',null)}
            </a>
        </isif>
    <iselse/>
        <div class="in-wishlist">
            ${Resource.msg('global.iteminwishlist','locale',null)}
        </div>
    </isif>
</div>
```

here we are passing product id which is removed from cart.

Note : you need to configure the remove button's class name or id in blueshift.eventhooks.js file for "BSRemoveFromCartClickId" this variable.

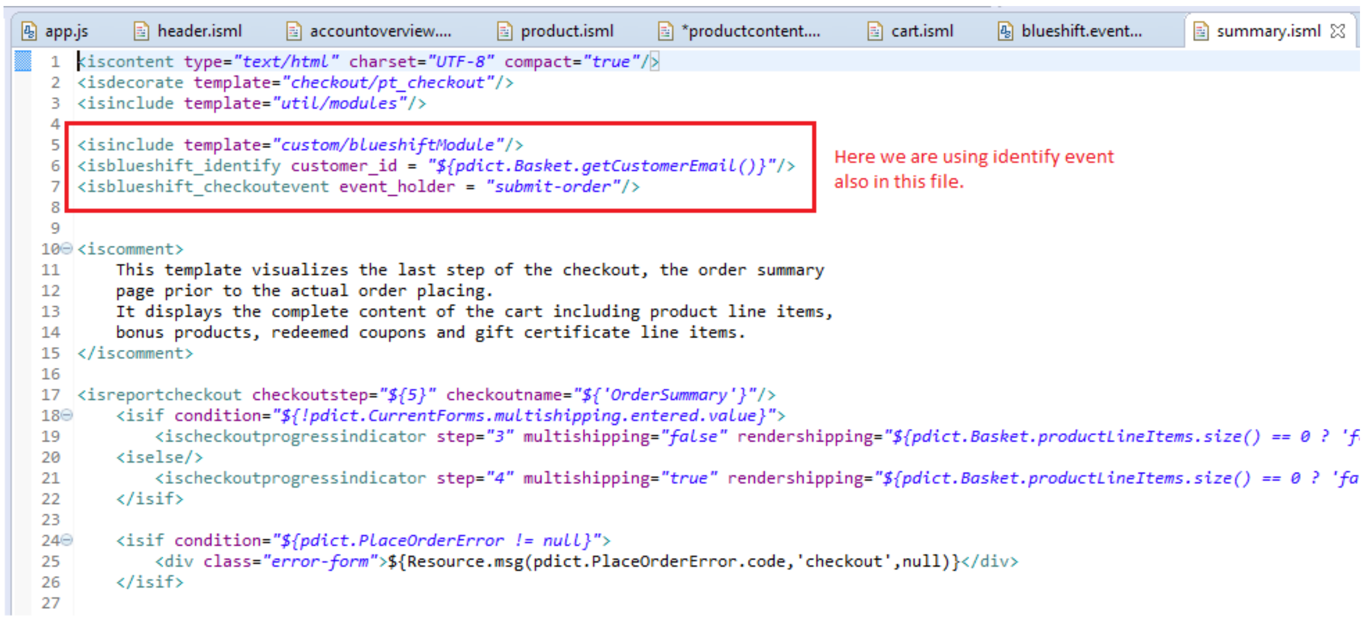
Checkout Event :

This event will be fired when the user is on checkout page with all the order details before final purchase.
To add this event, append the bellow given code to checkout summary page, that is summary.isml file.

Add the following code to this file.

```
<isblueshift_checkoutevent event_holder = "submit-order"/>
```

Screenshot :



```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<isdecorate template="checkout/pt_checkout"/>
<isinclude template="util/modules"/>

<isinclude template="custom/blueshiftModule"/>
<isblueshift_identify customer_id = "${pdict.Basket.getCustomerEmail()}"/>
<isblueshift_checkoutevent event_holder = "submit-order"/>


<iscomment>
    This template visualizes the last step of the checkout, the order summary
    page prior to the actual order placing.
    It displays the complete content of the cart including product line items,
    bonus products, redeemed coupons and gift certificate line items.
</iscomment>

<isreportcheckout checkoutstep="${5}" checkoutname="${'OrderSummary'}"/>
    <isif condition="${!pdict.CurrentForms.multishipping.entered.value}">
        <ischeckoutprogressindicator step="3" multishipping="false" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'f
    <iselse/>
        <ischeckoutprogressindicator step="4" multishipping="true" rendershipping="${pdict.Basket.productLineItems.size() == 0 ? 'fa
    </isif>

    <isif condition="${pdict.PlaceOrderError != null}">
        <div class="error-form">${Resource.msg(pdict.PlaceOrderError.code,'checkout',null)}</div>
    </isif>

```

Here we are using identify event also in this file.

You need to put this line after <isdecorate> tag as the core js file is loaded in header so we need to wait for first loading header and then we can put our calls. So its compulsory to put our hook after <isdecorate> tag.
The above code will include one isml file which contains the following function.
(the bellow code is only for developer purpose)

```
function submitCheckoutEvent(){
      var output =
      BSObject.checkoutEvent('${pdict.Basket.getCustomerEmail()}','${pdict.Ba
      sket.getShippingTotalPrice()}','<isprint value="${tl}"
      encoding="off">','${pdict.CurrentCustomer.getProfile().customerNo}');
      return true;
 }
```

Purchase Event :

you can fire this event when the user completes a purchase. When order is placed and user returns to the order summary page at that time we can put this event.

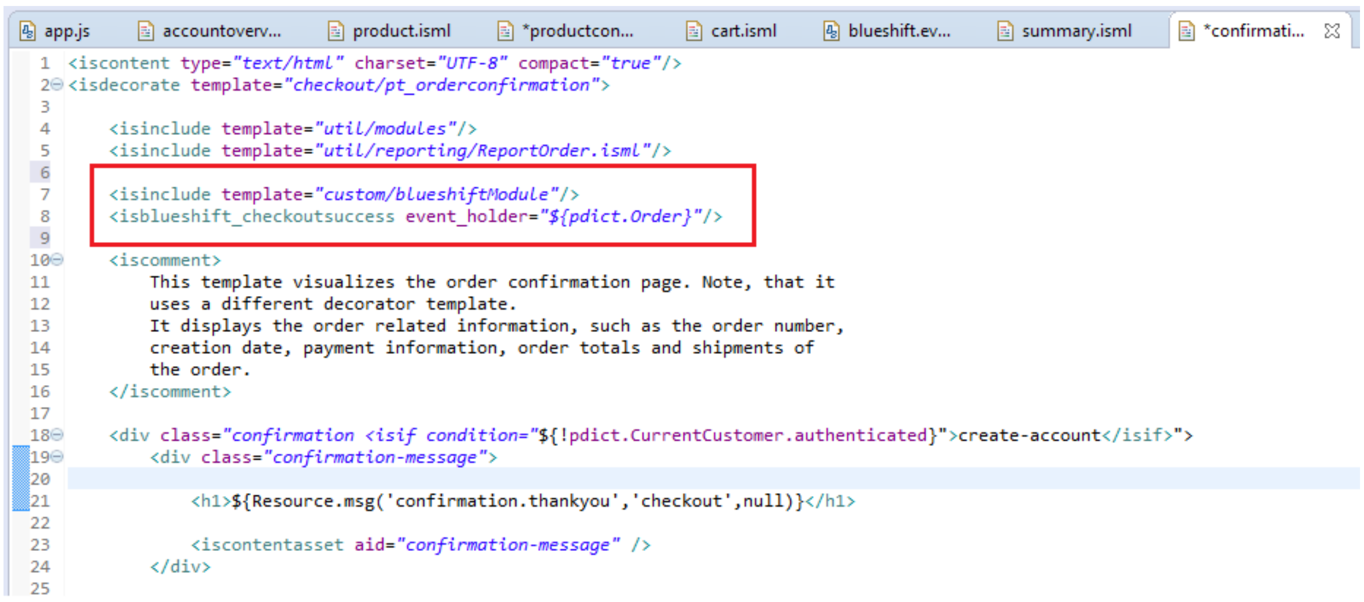
You can put this event on “confirmation.isml” file between <isdecorate> tags.

Add following code between <isdecorate> tag in confirmation.isml

Code Snipet :

```
<isinclude template="custom/blueshiftModule"/>
<isblueshift_checkoutsuccess event_holder="${pdict.Order}"/>
```

Screenshot :



```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<isdecorate template="checkout/pt_orderconfirmation">

    <isinclude template="util/modules"/>
    <isinclude template="util/reporting/ReportOrder.isml"/>

    <isinclude template="custom/blueshiftModule"/>
    <isblueshift_checkoutsuccess event_holder="${pdict.Order}"/>

    <iscomment>
        This template visualizes the order confirmation page. Note, that it
        uses a different decorator template.
        It displays the order related information, such as the order number,
        creation date, payment information, order totals and shipments of
        the order.
    </iscomment>

    <div class="confirmation <isif condition="${!pdict.CurrentCustomer.authenticated}">create-account</isif>">
        <div class="confirmation-message">

            <h1>${Resource.msg('confirmation.thankyou','checkout',null)}</h1>

            <iscontentasset aid="confirmation-message" />
        </div>

```

This code will include “template/default/tag_events/checkoutsuccess.isml” file. Here it filters the Order object and gether the required data and pass it to the blueshift purchase event.

```
BSObject.purchaseSuccess('<isprint value="${details}"
encoding="off">','<isprint value="${products_str}" encoding="off">');
```

The above is the call to blueshift purchase event. For more details about data you can view the isml file.

Search Event :

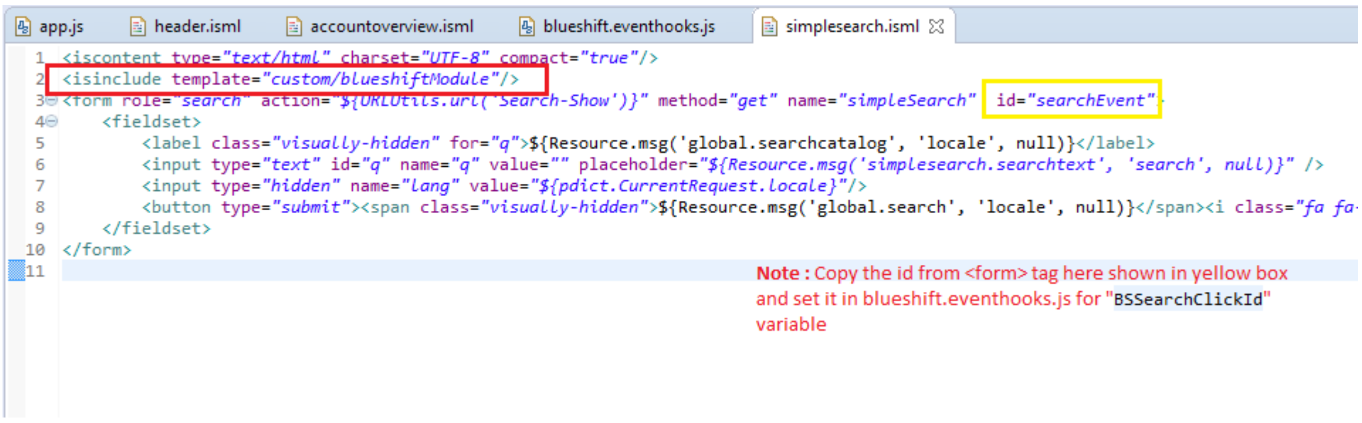
Search event is fired when user searches something and search form is submitted at that time this event is executed.
this event supplies the search string to blueshift api.

you can put this event on search page isml like simplesearch.isml. and copy the class name or id name of search form and configure that in blueshift.eventhooks.js file for **" BSSearchClickId"** this variable.

add bellow code to the simplesearch.isml

```
<isinclude template="custom/blueshiftModule"/>
```

Screenshot :



```
<iscontent type="text/html" charset="UTF-8" compact="true"/>
<isinclude template="custom/blueshiftModule"/>
<form role="search" action="${URLUtils.url('Search-Show')}" method="get" name="simpleSearch"  id="searchEvent"
    <fieldset>
        <label class="visually-hidden" for="q">${Resource.msg('global.searchcatalog', 'locale', null)}</label>
        <input type="text" id="q" name="q" value="" placeholder="${Resource.msg('simplesearch.searchtext', 'search', null)}" />
        <input type="hidden" name="lang" value="${pdict.CurrentRequest.locale}"/>
        <button type="submit"><span class="visually-hidden">${Resource.msg('global.search', 'locale', null)}</span><i class="fa fa
    </fieldset>
</form>
```

Note : Copy the id from <form> tag here shown in yellow box and set it in blueshift.eventhooks.js for "BSSearchClickId" variable

NOTE : There is a sample implementation isml files in int_blueshift cartridge templates in default/checkout , default/product, default/components, default/search. Thisis for reference only. It is recommended to remove the sample data folders from the cartridge once it's successfully integrated.