**Signifyd INTEGRATION**

SFCC Version 22.1.0

Signifyd Version 3.2.0



Table of Contents

[1. Summary 1-3](#_Toc104994157)

[2. Components 2-4](#_Toc104994158)

[3. Component Overview 3-5](#_Toc104994159)

[3.1 Functional Overview & Integration Guide 3-5](#_Toc104994160)

[3.1.1 Use Cases 3-5](#_Toc104994161)

[3.1.2 Setup access to the Site Preference 3-8](#_Toc104994162)

[3.1.3 Setup Site Preference Values 3-14](#_Toc104994163)

[3.1.4 Setup Service Framework Configuration 3-16](#_Toc104994164)

[3.1.5 Setup Job Schedules Configuration 3-17](#_Toc104994165)

[3.1.6 API Integration – SFRA 3-19](#_Toc104994166)

[3.1.7 API Integration – SiteGenesis Controllers 3-19](#_Toc104994167)

[3.1.8 API Integration – SiteGenesis Templates 3-20](#_Toc104994168)

[3.1.9 API Integration - Pipelines 3-21](#_Toc104994169)

[3.1.10 API Integration – Limitations and Constraints 3-24](#_Toc104994170)

[3.2 Other Non-Transactional Operations 3-27](#_Toc104994171)

[4. Configuration Guide 4-27](#_Toc104994172)

[4.1 Setup 4-27](#_Toc104994173)

[4.1.1 Configuration on Signifyd side 4-28](#_Toc104994174)

[4.2 Testing 4-30](#_Toc104994175)

[4.2.1 Automated Testing 4-32](#_Toc104994176)

[4.2.1.1 Unit testing 4-32](#_Toc104994177)

[4.2.1.2 Integration testing 4-32](#_Toc104994178)

[4.3 Troubleshoot 4-32](#_Toc104994179)

[4.3.1 Missing API Key 4-32](#_Toc104994180)

[4.3.2 Wrong API Key 4-32](#_Toc104994181)

[4.3.3 Service not enabled 4-33](#_Toc104994182)

[4.3.4 Order fields not being updated 4-33](#_Toc104994183)

[5. Operations, Maintenance 5-34](#_Toc104994184)

[5.1 Availability 5-34](#_Toc104994185)

[5.2 Support 5-34](#_Toc104994186)

[5.3 Intended Locales 5-34](#_Toc104994187)

[6. Release History 6-34](#_Toc104994188)

[7. Process Flow Diagrams 7-35](#_Toc104994189)

**Intended Audience**

This document is intended for the technical audience that will be directly involved in the setup and/ or integration of this Salesforce Commerce Cloud cartridge.

# Summary

Signifyd is a fraud solution that provides a financial guarantee, allowing businesses to increase sales while reducing fraud losses. The Signifyd cartridge will be integrated into Salesforce Commerce Cloud using three primary API integration points:

1. Pre-payment authorization (Checkout API, Transaction API)
2. Post-payment authorization (Sale API, Webhooks)
3. Post order fulfilment (Fulfilment API)

Sale API is used if merchants are executing a "Post-Auth" Flow where they call into Signifyd after the order has been placed and the payment has been authorized with the Payment Gateway. Checkout API is used if merchants are executing a "Pre-Auth" Flow where they call into Signifyd before authorizing a payment with the Payment Gateway. In "Pre-Auth" flow, for each order, there are at least 2 API calls, a Checkout API call prior to payment authorization and a Transaction API after authorization

This document primarily serves as the LINK implementation guide for setting up Signifyd on SFRA and SiteGenesis.

The setup and custom code configuration described in this document assume the use of SFRA version >= 3.3.0 and SiteGenesis 103.1.11 release of app\_storefront\_core. Custom coding might be required if adapting the cartridge to work with other SiteGenesis releases, pre-2.0 releases, and versions of SiteGenesis that do not include the RequireJS framework.

# Components

**Cartridge Name**Int\_signifyd

**New Signifyd Controller**  
controllers/Signifyd.js

**Modified System Controller**COSummary  
CheckoutServices

**Modified System Pipeline**  
COSummary

**Modified Core Template**  
htmlhead.isml

**Scripts**  
service/signifydInit.js  
service/signifyd.js  
service/pp\_signifyd.ds  
job/CreateMissingOrders.js

**Templates**  
default/signifyd\_device\_fingerprint.isml

**Cartridge Path**

**Pipeline based approach:**int\_signifyd:app\_storefront\_pipelines:app\_storefront\_core...

**Controllers based approach:**int\_signifyd:app\_storefront\_controllers:app\_storefront\_core...

**SFRA based approach:**int\_signifyd\_sfra:int\_signifyd:app\_storefront\_base...

**MetaData**

* metadata.xml
* services.xml
* jobs.xml

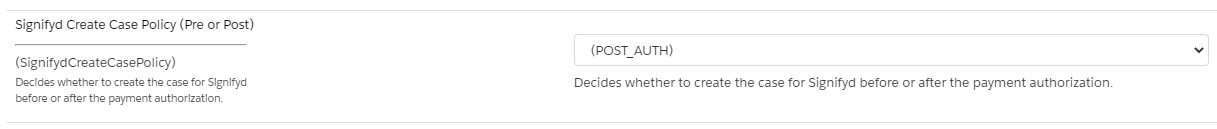
# Component Overview

## Functional Overview & Integration Guide

### Use Cases

**Post-authorization**

If the custom site preference ***SignifydCreateCasePolicy*** is set to “POST\_AUTH”, Signifyd decisions are returned asynchronously, so an HTTP callback (webhook) is used to return their guarantee decision. Refer to section 7 - Process Flow Diagrams for post-auth process flow.



**Action 1.** The Signifyd Sale API is called after the Salesforce Commerce Cloud order has gone through the authorization process against the payment provider right before displaying the order summary page. Because it is only called during the order creation, this will ensure that create case is never called again for that same order. The Sale API is called when all below actions are completed:

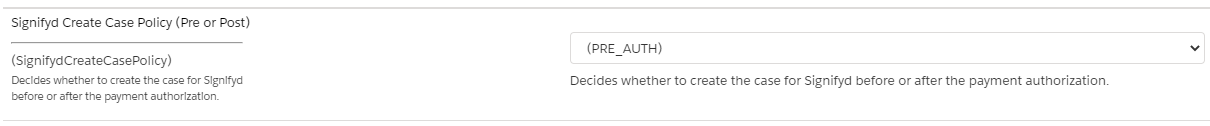
1. Payment is authorized without errors.
2. The order is successfully placed.
3. Confirmation of order is sent to the customer.

**Note**: Any customizations done by the merchant should guarantee that the Sale API is called after the above steps are completed.

**Action 2.** The second integration point is a publicly accessible URL that will be used as the callback/web-hook endpoint. This endpoint will be called when Signifyd has completed its fraud assessment and a decision is made to either approve or decline the order for financial guarantee. This triggers an update to the order in SFCC and could also indicate that the order is ready to export (depending on settings).

**Pre-authorization**

If the custom site preference ***SignifydCreateCasePolicy*** is set to “PRE\_AUTH”, Signifyd decisions are returned synchronously after the relevant Signifyd API is called. Refer to section 7 - Process Flow Diagrams for pre-auth process flow.



**Action 1.** The Signifyd Checkout API is called before the Salesforce Commerce Cloud order has gone through the authorization process against the payment provider. If Signifyd returns a declined/rejected response, the authorization call will not happen, and the order will be failed. The storefront will display a default error message to the customer. If the merchant wants to customize this message, it can be changed directly in the code. If Signifyd returns an accept/approved response, Salesforce Commerce Cloud proceeds with the authorization process against the payment provider.

**Action 2.** For Signifyd "Approved" orders, the Transaction API is called to send payment status after the authorization result. Some of the fields should be manually added by the merchant for the Transaction API. More details in the section 3.1.11 API Integration – Limitations and Constraints.

**Passive Mode**

Passive mode can be used by setting the custom site preference ***SignifydPassiveMode*** to “Yes”. If passive mode is enabled, Signifyd decision will be visible on Business Manager through the order custom attributes, but it won’t impact the order status regardless of Signifyd accept/reject decisions. When the passive mode is switched off, Signifyd decision will be updated on the order and the order status will be impacted according to the accept/reject decisions.

**Order Fulfillment**

A fulfillment represents a shipment of one or more items in an order. The merchant can submit fulfillments details for orders that were shipped, even if the order is partially shipped. The function **sendFulfillment** from the file **signifyd.js** is available to be called at the time from when the order is fulfilled on the merchant order flow. Some of the fields for the Fulfillment API needs to be added manually by the merchant. More details in the section section 3.1.11 API Integration – Limitations and Constraints.

**Manual Review in Pre-Auth Integration**

To use Signifyd’s manual review feature for pre-auth, you would need to configure the DECISION\_MADE webhook. In this scenario, Signifyd decisions (for manual review orders) will be returned asynchronously, so an HTTP callback (webhook) is used to return the guarantee decision.

The expected order flow process for manual review orders in pre-auth setting is mentioned below:

1. Order is sent to Signifyd for pre-auth fraud check
2. Signifyd deems the order to be a manual review order and returns a PENDING decision response back to SFCC (instead of ACCEPT or REJECT)
3. SFCC sends the order for payment authorization
4. Once the payment authorization process is complete, SFCC sends the payment authorization details to Signifyd via the transaction API endpoint
5. Based on the payment authorization details sent, Signifyd makes a manual decision on the order (order moved from Pending to Accept/Reject)
6. DECISION\_MADE webhook is sent to SFCC for this asynchronous decision by Signifyd
7. SFCC consumes Signifyd’s DECISION\_MADE webhook and processes the order as per the defined merchant settings. Example:
   1. For Signifyd approved orders, the order may be exported to merchant’s OMS
   2. For Signifyd declined/rejected orders, the order may not be exported to merchant’s OMS

The below table lists out the different types of scenarios related to Manual Review orders that a merchant may want to consider.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scenario | | Signifyd decision for a Manual Review Order | Merchant Scenario | Action needed from Merchant | Who takes liability? |
| 1 | Accept | | Merchant decides to progress order to fulfillment | No action needed as this should happen automatically via DECISION\_MADE webhooks | Signifyd |
| 2 | Reject | | Merchant decides to fulfill Signifyd declined order | 1. Merchant user would need to mark the order as Good within Signifyd console 2. Manually progress order to OMS for order fulfillment | Merchant |
| 3 | Reject | | Merchant decides to not fulfill Signifyd declined order | - Merchant would need to reverse payment authorization for this order since the authorization was already done.  - In case the payment was captured prior to receiving Signifyd's decision, the payment would need to be refunded | N/A |
| 4 | Reject | | Merchant decides to resubmit order to Signifyd | - Merchant user would need to resubmit order from the console and add additional details related to why this declined order is being resubmitted.  - If Signifyd subsequently approves this order post resubmit, then order should be progressed to the OMS for fulfillment (this would happen automatically via webhooks)  - If Signifyd declines this order again post resubmit, then merchant can decide if they still want to progress the order (scenario 2) or not (scenario 3) |  |

**Receiving Signifyd’s asynchronous decision**

**Webhooks**

Webhooks are sent by Signifyd when Signifyd approves/declines an order for guarantee and will be indicated in the checkpointAction string as “ACCEPT”, “REJECT”, or “HOLD". Typically, for pre-auth integrations, there is no need to configure a webhook as Signifyd’s decision will be synchronous. However, if a merchant wants to use Signifyd’s manual review feature for pre-auth, then you would need to configure the DECISION\_MADE webhook. In this scenario, Signifyd decisions (for manual review orders) will be returned asynchronously, so an HTTP callback (webhook) is used to return the guarantee decision.

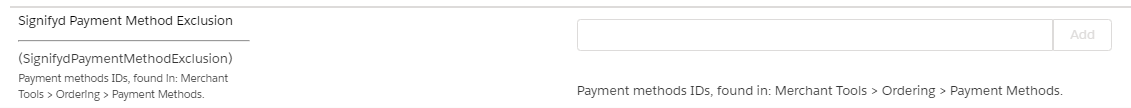
**Please note**: For all Signifyd APIs, “caseId”= “signifydId”, which is Signifyd's unique identifier for the order in our systems.

Webhooks are configured in the **Signifyd Console** > **Settings** > **Notifications**.

Once the Signifyd decision has been made, it will be listed under order attributes.

**Payment Exclusion**

Merchants can add payment methods to an exclusion list by adding the payment method ids in the custom preference **SignifydPaymentMethodExclusion.** The payment method id informed in the preference should be the same id visible in the Business Manager menu **Merchant Tools > Ordering > Payment Methods**.



Orders with an excluded payment method from the preference will be ignored during checkout and will not trigger a request to Signifyd. These orders will instead be directly sent for payment authorisation. The boolean attribute **SignifydPaymentMethodExclusionFlag** in the order will indicate that this order was ignored by Signifyd.

**Create Missing Orders Job**

The **Signifyd-CreateMissingOrders** job can be scheduled by the merchant to run periodically and catch orders that do not have a case ID (SignifydCaseID attribute = null), have a retry count less than the maximum retry count configured in the custom preference **SignifydMaxRetryCount** (default is 3), and do not have the status FAILED or CANCELLED. Additionally, orders that were ignored due to the payment method being listed in the payment exclusion custom preference and have the attribute **SignifydPaymentMethodExclusionFlag** set as true, will not be processed by the job.

### Setup access to the Site Preference

All permissions for customers can be set in Administration →Organization →Role & Permissions. You can allow admin level users to edit Site settings and disallow non-admin users. You may need to make changes to this in order to enable or disable access to the required Signifyd site preferences.





Setup Eclipse

***Steps for Loading the Cartridge in Eclipse***

A workspace is an Eclipse-specific local directory that contains Eclipse projects. Normally Eclipse projects are connected to Java source directories (packages). In Demandware Studio projects are different: they either define a connection to a Salesforce Commerce Cloud instance or they point to a Salesforce Commerce Cloud cartridge. They are never used to compile Java projects since Java code is not used in Salesforce Commerce Cloud application programming.

Each workspace should have only 1 Salesforce Commerce Cloud server connection (covered later in this module). For example, if you are a developer working on numerous client projects, you will want to create a separate workspace for each client. Each client workspace will then have only 1 specific server connection.

**Run the Create a Workspace activity.**

**To install the UX Studio plugin into Eclipse and to create a new workspace (when using UX Studio for the first time), follow these steps:**

1. The first time you use the application, you will be prompted to create a new workspace name. Give your workspace a name that references the client you are working with.



1. Eclipse will first display the Welcome message in the main working area. 

**Creating a Server Connection**

In order to upload your code to a Salesforce Commerce Cloud server, you will need to create a server connection in UX Studio. A server connection allows you to push your code to the server instance, but you will not be able to pull the code onto your personal computer from the Salesforce Commerce Cloud server. The connection is a 1-way push only.

**Create a new server Connection**

1. From UX Studio, click **Fil**e->**New**->**Digital Server Server Connection**. The new server connection box opens.
2. Complete it as follows.

In the **Project name** and **Host name** fields, use the host name provided by your client:   
e. g. **https://signifyd01-tech-prtnr-na05-dw.demandware.net/**Enter your password. Check the **Remember Password** flag.

1. Click **Next**.
2. A security warning regarding an invalid certificate for your sandbox shows up. Click **Yes** to continue.

Select **version1** as the target version you want to upload your files to: 

1. Click **Finish**.

Your connection project is now connected to your sandbox and will be used to upload any cartridge projects to that sandbox, as seen later in this module.

**Import a project in Studio**

1. From within UX Studio, click on **File**->**Import...** an import window will open.
2. From the Import window, click to expand the **General** menu.
3. Click the **Existing Projects into Workspace** option. If you have an SVN server, you can import projects directly from a repository, which is the most common way to obtain cartridges when you are working on a project.



1. Click ‘**Next**’.
2. In the next window, click to ‘**Browse...**’ button. 
3. Locate the folder on your hard drive where cartridges are located. Your instructor will provide a zip file with all solution cartridges for you to install locally. Click **OK**.
4. Any cartridges in the folder structure (including subfolders) will be displayed in the **Projects** box. Click **Select All:**

1. Click ‘**Finish**’.



1. The next dialog allows you to select the specific cartridges you want uploaded to your server connection. Click **Select All.**
2. Click **OK** to upload the cartridges and finish the import process.
3. You might receive a dialog stating to delete projects on the server not matching the ones in your workspace. If you’re the only one working on that instance e.g., it’s your personal sandbox you might recognize the projects there.



1. If you import cartridges before you have an active server connection or somehow forgotten to link a cartridge to the server, do the following to ensure that cartridges will get uploaded correctly: right-click the server connection and select **Properties:**



13. Select **Project References** and then select every cartridge that you want uploaded to the server connection and Click ok.

### Setup Site Preference Values

**Configuration - Metadata import**

First step is to import system object definitions for the Signifyd attributes for Order and Site Preferences. These are provided with cartridge in metadata.xml file

Upload this file via Business manager into your site: 1. Click on button "Upload" in Administration > Site Development > Import & Export



2. Choose your local file and again click "Upload"

3. And Click back button to return to Import page.

4. On the Meta Data section click on the 'Import' button

5. Select the metadata.xml file that you just uploaded and click 'next' to go through import process.



**After import – Preference Entry**

You will now see a 'Signifyd Settings' attribute group in the site preference section. Merchant Tools > Site Preferences > Custom Site Preferences:



You will now also be able to see the Signifyd attribute under an order by clicking on the 'Attributes' tab.

Graphical user interface, text, application, email

Description automatically generated

### Setup Service Framework Configuration

**Configuration – Service Framework Setup Import**

Import the base Signifyd Service Framework configuration required by the Signifyd cartridge.

**Step 1**

In Business Manager go to Administration->Operations->Import Export and select ‘upload’ then browse to the *services.xml* file that is located in the meta folder included with the Signifyd cartridge.



**Step 2**

Once the services file is uploaded click Import and chose *merge* to import the default Service Framework configuration.

**Step 3**

Once imported you will need to navigate to the Signifyd services configuration and make sure they are enabled.

Graphical user interface, application

Description automatically generated

### Setup Job Schedules Configuration

**Configuration – Job Schedules Setup Import**

Import the base Signifyd Job Schedules configuration required by the Signifyd cartridge.

**Step 1**

In Business Manager go to Administration->Operations->Import Export and select ‘upload’ then browse to the *jobs.xml* file that is located in the meta folder included with the Signifyd cartridge.



**Step 2**

Once the jobs file is uploaded click Import and choose *replace* to import the default Job Schedules configuration.

**Step 3**

In Business Manager go to Administration->Operations->Job Schedules. The Signifyd-CreateMissingOrders job will be displayed:



**Step 4**

Select Signifyd-CreateMissingOrders to enter the Job Schedule configuration. Configure your Job Schedule to run once, daily, or on any desired schedule. We recommend you schedule your jobs to run at least once a day.



### API Integration – SFRA

**Script changes**

**Script:** checkoutHelpers.js

**Path**: signifyd\_sfra\_changes/cartridge/scripts/checkout/checkoutHelpers.js

Function **placeOrder()** on **checkoutHelpers.js** was modified to add the following logic:

* If the cartridge is enabled and the **SignifydHoldOrderEnable** is set to **Yes**, then the order export status will be set to Not Exported and will later be updated based on Signifyd’s webhook decision.
* If the cartridge is enabled and the **SignifydHoldOrderEnable** is set to **No**, then the order export status will be set to Ready For Export as in the default **placeOrder()** function.
* Different logic can be added if the merchant wants to have customization for specific payments.

**Adding the necessary changes**

**Case 1 - Merchant didn’t customize the base SFRA checkoutHelpers.js file**

The file **checkoutHelpers.js** contains code to override the SFRA default function **placeOrder()** and add custom logic to set the export status according to the **SignifydHoldOrderEnable** custom preference.

The file should be place on the following path, to extend the existing SFRA file:

**{merchant\_customized\_cartridge}/cartridge/scripts/checkout/checkoutHelpers.js**

**Case 2 - Merchant already customized the base SFRA checkoutHelpers.js file**

If the merchant already has custom logic added on his own checkoutHelpers.js file, only the necessary changes can be added (example file available on signifyd\_sfra\_changes/cartridge/scripts/checkout/checkoutHelpers.js):

Text

Description automatically generated

### API Integration – SiteGenesis Controllers

**Modifications in System Controller**

Enabling SFCC to send requests to Signifyd requires a modification to the system controller file: *controllers/COSummary.js*

1. Add these two rows in the beginning of the *submit()* method:

var Signifyd = require('int\_signifyd/cartridge/scripts/service/signifyd');  
var orderSessionID = Signifyd.getOrderSessionId();

1. Add those two rows near the end of the same function method:

Signifyd.setOrderSessionId(placeOrderResult.Order, orderSessionID);

Signifyd.Call(placeOrderResult.Order);

Those two last lines (number 2) should go within last ‘else if’ statement.

*else if (placeOrderResult.order\_created) {*   
**….<Insert code here>…..**  
}

The final code should look something like this:



### API Integration – SiteGenesis Templates

**Modifications to Core Template**

**Device Fingerprinting**

In order to insert the fingerprint JavaScript snippet in the HTML <head> element, modify the template *default/components/header/htmlhead.isml*.

Add the following lines in the end of the file (around line 78):

<isif condition="${dw.system.Site.getCurrent().getCustomPreferenceValue('SignifydEnableCartridge')}">  
 <isinclude template="signifyd\_device\_fingerprint" />  
</isif>

The result should look like the following:



### API Integration - Pipelines

**Modifications to System Pipeline**

Another way to make SFCC send requests to Signifyd is modification to system pipeline for pipeline-based site implementations. Since the pipeline-based approach could eventually be deprecated by Salesforce Commerce Cloud, the controller-based approach is recommended

Pipeline name is*: pipelines/* *COSummary.xml*

Add a script *pp\_signifyd.ds* to the end of pipeline Submit as this shown on below image:



Input parameter for this script must be a current Order.



### API Integration – Limitations and Constraints

The Signifyd fraud service relies on transaction data passed back from the payment gateway. If a custom payment gateway is implemented, make sure to pass required information to Signifyd by modifying the *signifyd.js* file as shown below:

The values for AVS and CVV Response Code fields MUST map to standard response codes. See [this document](http://www.emsecommerce.net/avs_cvv2_response_codes.htm) for valid response codes. AVS and CVV values should be updated by the merchant for getSendTransactionParams() (in case of Pre-auth enabled) functions on signifyd.js file.

**getSendTransactionParams()**

Text, application

Description automatically generated

Order channel information should be updated by the merchant in the **getParams()** function:

**Graphical user interface, text, application

Description automatically generated**

**Fulfillment API**

For the Fulfillment API, the following request fields should be updated by the merchant on the function **getSendFulfillmentParams()**, according to the merchant’s shipping carrier and following the format specified on the [API documentation](https://docs.signifyd.com/#operation/Fulfillments):

Graphical user interface, text, application, email

Description automatically generated

**Pre-auth default error message**

When an order is failed during Pre-auth flow, the default SFCC technical error message is displayed. The merchant can customize the code to display a different error message on the code below:

**Graphical user interface, text, application

Description automatically generated**

## Other Non-Transactional Operations

You can enable logs for all operations with the Signifyd API for debugging. But don't forget to disable it after debugging to prevent uncontrolled growth of log files.

Go to Administration –>Operation –>Custom Log Settings. You can enable specific levels of logging for Signifyd. Each level brings a different or higher level of detail in the logs



**Note:** DEBUG log level is only available in sandboxes, Development and Staging instances. For Production instances, any log level above DEBUG can be used.

# Configuration Guide

## Setup

The Signifyd cartridge has a configuration setting to hold the order or immediately export depending on Signifyd’s guarantee decision. The site preference setting is called ***SignifydHoldOrderEnable*** and if set to true this indicates that the Salesforce Commerce Cloud order is held until the webhook listener is called and indicates that the order is approved. This is accomplished by marking order as 'Not Exported' to prevent it from exporting until the webhook listener updates the order to 'Ready for Export'.

The site preference ***SignifydCreateCasePolicy*** decides whether the create case policy will be pre-authorization (synchronous) or post-authorization (asynchronous).

Use the site preference ***SignifydDecisionRequest*** to setup the decision request as GUARANTEE (financial liability shift) or DECISION (no financial liability shift).

See below for screenshot of the settings, accessible through **Merchant Tools > Site Preferences > Custom Site Preference Groups > Signifyd Settings.**

Graphical user interface, application

Description automatically generated

### Configuration on Signifyd side

All Signifyd cases created during SFCC order creation can be viewed here: <https://app.signifyd.com/cases>

API key, Profile and all other settings can be set up on this page: <https://app.signifyd.com/settings>

The DECISION\_MADE webhook should be configured in <https://app.signifyd.com/settings/notifications> in order to update SFCC with the latest status from Signifyd.

Salesforce will block any access to the hyphenated demandware.net (e.g: development-xxx.demandware.net) that does not originate from the platform itself, which means that any attempt from third-party integrations that access the Storefront through a link will not be able unless passed through a vanity hostname, such as yourbrand.com, www.yourbrand.com, etc. This should be taken into consideration while configuring the URLs for the webhooks in the Signifyd console.

For the Development and Production instances, it is mandatory to use the vanity hostname, so you will need to add a URL Rule like below, for the Staging and sandbox instances this way can be used as well, but is it not mandatory



And add e.g. https://yourStoreURL.com/s/SiteGenesis/signifydcallback to the Webhook Addresses in <https://app.signifyd.com/settings/notifications>

If the custom site preference ***SignifydCreateCasePolicy*** is set to ‘PRE\_AUTH’, there is no need to configure a webhook to receive the decision back, unless the merchant needs aforementioned Signifyd's manual review services.

## Testing

**Pre-auth flow**

**Pre-auth Test Case 1:** Signifyd response: Accept

**Expected Result**:

1. Before the payment processor authorization, a synchronous call to Checkout API is made and a case is created on Signifyd.
2. Signifyd sends the checkpointAction field as “Accept” in the response. The attributes **SignifydCaseID**, **SignifydOrderURL**, **SignifydFraudScore**, **SignifydPolicy** and **SignifydPolicyName** are saved on the order and the order proceeds to authorization.
3. Depending on the payment processor response:
   1. Authorized: order proceeds to be placed. A Transaction API call is made after the order is placed to update the order details in Signifyd.
   2. Not authorized or error: the order fails. A Transaction API call is made to update the order details in Signifyd.

**Pre-auth Test Case 2:** Signifyd response: Reject and Passive Mode: No.

**Expected Result**:

1. Before the payment processor authorization, a synchronous call to Checkout API is made and a case is created on Signifyd.
2. Signifyd sends the checkpointAction field as “Reject” in the response. The attributes **SignifydCaseID**, **SignifydOrderURL**, **SignifydFraudScore**, **SignifydPolicy** and **SignifydPolicyName** are saved on the order.
3. The order fails and the attribute **SignifydOrderFailedReason** is updated on the order.
4. The default error message is shown to the user on the checkout page

**Pre-auth Test Case 3:** Signifyd response: Reject and Passive Mode: Yes.

**Expected Result**:

1. Before the payment processor authorization, a synchronous call to Checkout API is made and a case is created on Signifyd.
2. Signifyd sends the checkpointAction field as “Reject” in the response. The attributes **SignifydCaseID**, **SignifydOrderURL**, **SignifydFraudScore**, **SignifydPolicy** and **SignifydPolicyName** are saved on the order.
3. The attribute **SignifydOrderFailedReason** is updated on the order and the order proceeds to be authorized.
4. Depending on the payment processor response:
5. Authorized: order proceeds to be placed. A Transaction API call is made after the order is placed to update the order details in Signifyd.
6. Not authorized or error: the order fails. A Transaction API call is made to update the order details in Signifyd.

**Pre-auth Test Case 4:** Signifyd response: Hold

**Expected Result**:

1. Before the payment processor authorization, a synchronous call to Checkout API is made and a case is created on Signifyd.
2. Signifyd sends the checkpointAction field as “Hold” in the response. The attributes **SignifydCaseID**, **SignifydOrderURL**, **SignifydFraudScore**, **SignifydPolicy** and **SignifydPolicyName** are saved on the order.
3. Depending on the payment processor response:
   1. Authorized: order proceeds to be placed. A Transaction API call is made after the order is placed to update the order details in Signifyd.
   2. Not authorized or error: the order fails. A Transaction API call is made to update the order details in Signifyd.
4. Decision\_Made Webhook will be triggered to send a new checkpointAction and update the order **SignifydPolicy** and **SignifydPolicyName** attributes.

**Post-auth flow**

**Post-auth Test Case 1:** Signifyd response: Accept

1. After the order is successfully placed, a call to Sale API is made.
2. The attributes **SignifydCaseID** and **SignifydOrderURL** are saved on the order.
3. The Decision\_Made Webhook is triggered, a callback is made to SFCC to update the order attribute **SignifydPolicy** with the “Accept” checkpointAction value from the response.
4. If **SignifydHoldOrderEnable** custom preference is set to Yes, the order export status is updated to Ready to Export.

**Post-auth Test Case 2:** Signifyd response: Reject

1. After the order is successfully placed, a call to Sale API is made.
2. The attributes **SignifydCaseID** and **SignifydOrderURL** are saved on the order.
3. The Decision\_Made Webhook is triggered, a callback is made to SFCC to update the order attribute **SignifydPolicy** with the with the “Reject” checkpointAction value from the response.
4. If **SignifydHoldOrderEnable** custom preference is set to Yes, the order export status is updated to Not Exported.

### Automated Testing

On this implementation we have provided unit and integration testing

### Unit testing

To do the automated testing open your command prompt (windows) or terminal (MacOS/Linux/Unix) navigate to the folder above the cartridges and tun the command “npm run test”, this will test the functionalities within the cartridge source code

### Integration testing

To run the automated integration testing create a dw.json file on the navigate to the folder above the cartridges

{  
 "hostname" : "somesb.demandware.net",  
 "username" : "someUser",  
 "password" : "somePassoword",  
 "version" : "someversion"  
}

Then run the command npm run test:integration

## Troubleshoot

### Missing API Key

If the API key provided by Signifyd is not correctly informed in the site custom preferences, the following error will be seen in the logs for any of the services:

Null Arguments in The given string to encode was null

**Solution**: The merchant should verify if the custom preference **SignifydApiKey** is correctly setup with the API key.

### Wrong API Key

If the API key is wrong, the following error will be seen in the logs for any of the services:

Provided API Key not found.

**Solution**: The merchant should double check if the custom preference **SignifydApiKey** has the correct API key.

### Service not enabled

If a service is not correctly enabled in Business Manager, the following error will be triggered:

Service={name\_of\_the\_service} status=SERVICE\_UNAVAILABLE errorCode=0 errorMessage=The service is not enabled

**Solution**: The merchant should double check if the service informed in the error message is enabled.

### Order fields not being updated

If the Signifyd related fields in the orders are not being updated, the possible reasons are:

* **Incorrect setup of the code**. The merchant should check if all steps to integrate the cartridge were correctly followed.
* **Incorrect setup of webhooks**. The merchant should check if only the Decision\_Made webhook is being used and correctly setup in the Signifyd console.
* **Code customizations**. The merchant should check if any customizations applied are not interfering in the cartridge functionality.
* **Incorrect data being passed in the service requests**. Some of the fields for Transaction and Fulfillment API need to be manually mapped by the merchant directly in the code during the integration. If a field is not correctly mapped or has a wrong value, this might cause the API to not work correctly, thus not updating the order.
* **Service timeouts**. The service timeout can be increased in the service configurations if needed, following SFCC default steps to do it. If the timeouts persist, Signifyd should be contacted.

# Operations, Maintenance

## Availability

Availability/Uptime is 24/7 is the intended access. But in case of service failure the order will get placed. And a script step job can be configurated to create the missing orders not send to Signifyd

## Support

For implementation questions or issues please contact your assigned Customer Success Manager. For general support questions or issues [contact Support](https://www.signifyd.com/contact/).

## Intended Locales

Out of the box the cartridge supports “en\_Us” locale, but other locales may be added according to Signifyd’s service availability, Signifyd has support for multiple countries and locales.

# Release History

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Changes** |
| 16.1.0 | 08/15/2015 | Initial release |
| 18.1.0 | 04/01/2018 | Added Job Schedule configuration and template modification |
| 19.1.0 | 02/28/2019 | Added modifications to support SFRA |
| 20.1.0 | 03/24/2020 | Added Unit and Integration testing |
| 22.1.0 | 05/31/2022 | V3 API release |

# Process Flow Diagrams

