

FORTER CARTRIDGE

Version 23.0.0



Table of Contents

1.	Summary.....	1-3
2.	Component Overview.....	2-3
2.1	Functional Overview.....	2-3
2.2	Use Cases	2-3
2.3	Limitations and Constraints	2-8
2.4	Compatibility.....	2-8
2.5	Privacy, Payment.....	2-9
3.	Implementation Guide	3-9
3.1	Setup	3-9
3.2	Configuration.....	3-9
3.2.1	Setup	3-9
3.2.2	Metadata import	3-11
3.2.3	Cartridge paths.....	3-12
3.2.4	Forter custom site preferences	3-14
3.3	Custom Code.....	3-16
3.3.1	Pipelines	3-16
3.3.2	Controllers.....	3-25
3.3.3	SFRA.....	3-32
3.3.4	Forter integration in Checkout / Payment Flow.....	3-39
3.3.5	Pre-Authorization Flow.....	3-61
3.3.6	Footer.....	3-67
3.4	Testing	3-67
4.	Operations, Maintenance	4-68
4.1	Data Storage	4-68
4.2	Availability.....	4-68
4.2.1	Forter error	4-68
4.3	Support	4-68
5.	User Guide	5-69
5.1	Roles, Responsibilities	5-69
5.2	Business Manager	5-69
5.2.1	Menu extension	5-69
5.2.2	Forter section	5-70
5.2.3	Forter configuration	5-70
5.2.4	Forter Orders.....	5-71
5.2.5	Order Update job.....	5-73
5.3	Storefront Functionality	5-75
5.3.1	JavaScript Snippet.....	5-75
6.	Known Issues	6-75
7.	Release History	7-75

1. Summary

The Forter cartridge adds the power of Forter's new generation fraud prevention to the Salesforce Commerce Cloud platform, to meet the challenges faced by modern enterprise e-commerce. Only Forter provides fully automated, real-time Decision as a Service™ fraud prevention, backed by a 100% chargeback guarantee.

The system eliminates the need for rules, scores or manual reviews, making fraud prevention friction-free. Every transaction receives an instant approve/decline decision, removing checkout friction and the delays caused by manual reviews.

The Forter cartridge provides fraud prevention that is invisible to buyers and empowers merchants with increased approvals, smoother checkout and the near elimination of false positives - meaning more sales and happier customers.

Behind the scenes, Forter's machine learning technology combines advanced cyber intelligence with behavioral and identity analysis to create a multi-layered fraud detection mechanism.

The result is best for online merchants, and best for online customers.

2. Component Overview

2.1 Functional Overview

The Forter cartridge links your Salesforce Commerce Cloud platform to Forter's sophisticated fraud fighting system. Each order is analyzed and a real-time approve or decline decision returned which is covered by a full fraud chargeback guarantee.

Merchants can configure the capture/void settings according to policy and preference.

All decisions can be seen in the Salesforce Commerce Cloud platform, and merchants can see more details relating to each transaction within the Forter Decision Dashboard.

2.2 Use Cases

There are number of use cases that may be seen with the Forter cartridge. Below are a few examples of use cases, with a description of the role Forter plays in the checkout process and where it fits into the customer experience.

UC - 1	Registered Customer: Approved Order Status Validation
This use case describes the high level steps in which a registered customer successfully creates an order and the order is Approved.	<ol style="list-style-type: none"> 1. The customer creates an account and logs in with the newly created account using email <i>approve@forter.com</i>. 2. The customer selects an item, adds it to the cart and proceeds to the cart page. 3. The customer clicks on the Checkout button and fills out the shipping form requirements. 4. The customer clicks on the Continue button and proceeds to fill in the billing form requirements. 5. The customer clicks on the Continue button, proceeds to the Payment page and clicks on the Place order button. 6. A call to Forter is sent, the transaction is approved and the thank you page is successfully loaded. 7. When the merchant navigates to Merchant Tools > Forter> Order, searches for the placed order and inspects the Forter Decision column it will be seen that the Forter Decision is <i>Approved</i> and the order status is <i>New</i>. <ul style="list-style-type: none"> • Note that a similar flow can be done for guest checkout

UC - 2	Registered Customer: Declined Order Status Validation
This use case describes the high level steps in which a registered customer creates an order and the Forter decides to Decline the order.	<ol style="list-style-type: none"> 1. The customer creates an account and logs in with the newly created account using email <i>decline@forter.com</i>. 2. The customer selects an item, adds it to the cart and proceeds to the cart page. 3. The customer clicks on the Checkout button and fills out the shipping form requirements. 4. The customer clicks on the Continue button and proceeds to fill in the billing form requirements. 5. The customer clicks on the Continue button, proceeds to the Payment page and clicks on the Place order button. 6. A call to Forter is sent, <i>the transaction is declined and based on the Forter configuration a declined message may be displayed</i>. 7. When the merchant navigates to Merchant Tools > Forter> Order, searches for the placed order and inspects the Forter Decision column it will be seen that the Forter Decision is <i>Declined</i> and the order status is <i>Failed</i>. <ul style="list-style-type: none"> • Note that a similar flow can be done for guest checkout

UC - 3	Guest Customer: Failed Order Status Validation
This use case describes the high level steps in which a guest customer creates an order but the status order validation is Failed.	<ol style="list-style-type: none"> 1. The customer selects an item, adds it to the cart and proceeds to the cart page. 2. The customer clicks on the Checkout button > Checkout as Guest and fills out the shipping form requirements. 3. The customer clicks on the Continue button and proceeds to fill in the billing form requirements using an expired credit card. 4. The customer clicks on the Continue button, proceeds to the Payment page and clicks on the Place order button. 5. The payment gateway fails the order. 6. <i>The call to Forter is not sent, the transaction is not approved and the Salesforce Commerce Cloud standard failed message is displayed.</i> 7. When the merchant navigates to Merchant Tools > Forter> Order, searches for the placed order and inspects the Forter Decision column it will be seen that the Forter Decision is <i>Not sent</i> and the order status is <i>Failed</i>. <ul style="list-style-type: none"> • Note that a similar flow can be done for a registered user checkout.

UC - 4	Registered Customer: Login data, Payment data & Address data sent to Forter for Validation
This use case describes the high level steps in which a customer successfully creates an account and modifies the account properties such as addresses, payment information, and wish list items	<ol style="list-style-type: none"> 1. The customer creates an account and logs in with the newly created account. à The login details are sent to Forter. 2. The customer adds a credit card to the account. à The card details are sent to Forter. 3. The customer adds an address to the account. à The address details are sent to Forter. 4. The customer adds an item to the wish list. à The updated wish list is sent to Forter. <p>The latter information is used by Forter to detect fraudulent orders.</p>

UC - 5	Send Forter updated order status information
This use case describes how to change status on orders and send the updated status to Forter	<ol style="list-style-type: none"> 1. Create multiple orders via the storefront. 2. Navigate to Merchant tools > Forter > Order view. For each order, update the order status to a different value (e.g. Completed, Cancelled). 3. Navigate to Administration > Operations > Schedules and run the ForterOrderUpdate job. 4. Navigate to Merchant tools > Forter > Order view to confirm the Forter order status has been updated for the relevant orders.

UC - 6	Guest Customer: Paypal Data Flow
This use case describes how to submit Paypal Express orders to Forter	<ol style="list-style-type: none"> 1. The customer selects an item, adds it to the cart and proceeds to the cart page. 2. The customer clicks on the Checkout with Paypal button > logs into the relevant Paypal account (e.g. approve@forter.com), selects the shipping address, and confirms the order 3. The customer clicks on the Place Order button. 4. A call to Forter is sent, the transaction is approved and the thank you page is successfully loaded. 5. When the merchant navigates to Merchant Tools > Forter> Order, searches for the placed order and inspects the Forter Decision column it will be seen that the Forter Decision is <i>Approved</i> and the order status is <i>New</i>. <ul style="list-style-type: none"> • Note that a similar flow can be done for a declined transaction.

UC - 7	Customer Login
<p>This use case describes the possible Forter response and decisions for account login event.</p>	<ol style="list-style-type: none"> 1. A user accesses the Login page and logs in. 2. Information is sent to Forter. 3. Forter's response has the attribute fortterDecision with possible values: APPROVED, DECLINED, NOT_REVIEWED, VERIFICATION_REQUIRED 4. Merchant can use this attribute to take action (It's up to the merchant to decide what action he wants to take). 5. Custom code can be written and executed after the service response to do their desired action, examples: <ol style="list-style-type: none"> a. In case of "VERIFICATION_REQUIRED", the merchant can use a MFA/OPT of their choice and send the result by using the Authentication Attempt API, this is implemented in the cartridge's code but must be customized, see pages 27 for SiteGenesis implementation and page 31 for SFRA implementation. b. In case of "DECLINED", the merchant can decide to deny the login from the customer. c. In case of "NOT_REVIEWED", the merchant can allow the login or review on their own. d. In case of "APPROVED", the merchant can allow the login.

UC - 8	Customer updates profile, addresses and payment methods.
This use case describes the possible Forter response and decisions for account events, profile update, addresses and payment methods.	<ol style="list-style-type: none"> 1. A user accesses their profile. 2. User tries to update their profile, address or payment method. 3. Information is sent to Forter 4. Forter's response has the attribute <code>forterDecision</code> with possible values: APPROVED, DECLINED, NOT_REVIEWED, VERIFICATION_REQUIRED 5. Merchant can use this attribute to take action (It's up to the merchant to decide what action he wants to take) 6. Custom code can be written and executed after the service response to do their desired action, examples. <ul style="list-style-type: none"> a. In case of "VERIFICATION_REQUIRED", it is recommended the merchant verifies the customer's identity first, to allow the changes, and then update Forter on the verification results b. In case of "DECLINED", the merchant can decide to throw an error to the customer. c. In case of "NOT_REVIEWED", the merchant can allow the changes or review it on their own. d. In case of "APPROVED", the merchant can allow the changes.

2.3 Limitations and Constraints

In order to make use of the Forter cartridge, merchants must have a Forter account. Implementing the cartridge successfully will require the relevant API credentials.

2.4 Compatibility

The Forter integration cartridge was certified with the latest version of Salesforce Commerce Cloud (currently API version 21.20, Site Genesis version 105.2.0 and SFRA version 5.3.0). It is typically backward compatible with older versions since it uses common and stable methods accessing the Customer, Order and Payment system objects. Pipelines installations are uncertified and at your own risk.

The cartridge was validated with the out-of-the-box locales on both *RefArch* and *RefArchGlobal* sites with default locale en_US but can be used with any locale.

2.5 Privacy, Payment

- Forter's order validation call should be executed between payment authorization and payment capture. This positioning is important, because the request to Forter uses the authorization response data. Based on the configuration, the payment capture either can or cannot be executed if Forter returns a "decline" decision.
- When the ***Auto-invoice when transaction is approved*** option is checked the payment capture will be executed on approval.
- When a decline response is received, the action taken regarding voiding the payment depends on the merchant's chosen configuration setting. The merchant can opt to void the payment manually or automatically, either immediately.
- Forter complies with and exceeds the requirements of PCI DSS standard level 1, and Forter is PCI Level 1 Certified. Please note that Forter does not collect full PAN and that Forter is committed to the appropriate protection of the parts of Cardholders' Data that it collects; this is achieved by a thorough hardening of the full Cardholders' Data Environment (CDE).

3. Implementation Guide

3.1 Setup

The following cartridges have been added:

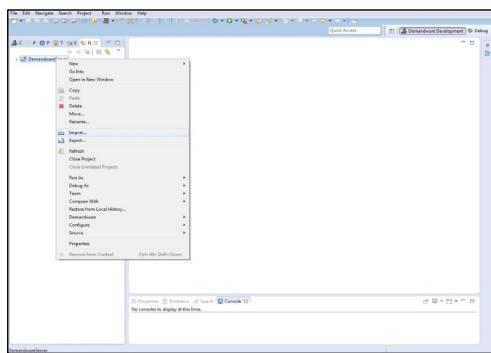
- int_forter
- int_forter_sfra
- bm_forter

3.2 Configuration

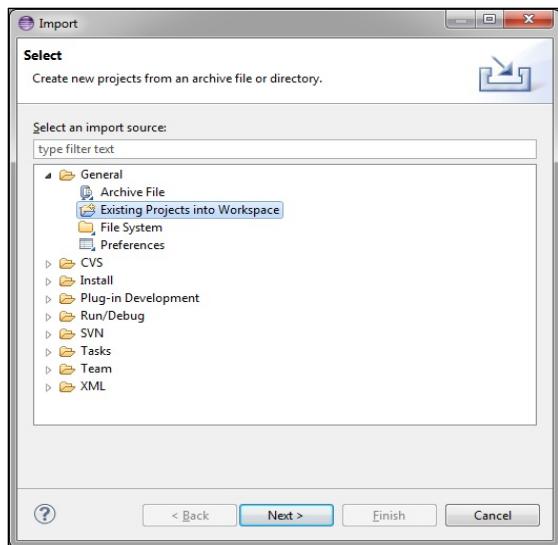
3.2.1 Setup

Importing the Forter cartridges is simple. Follow the steps below in order to import the cartridges:

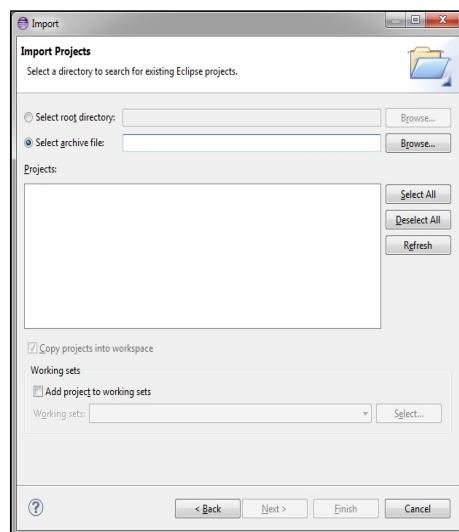
1. Select the connection to the DW server -> Import



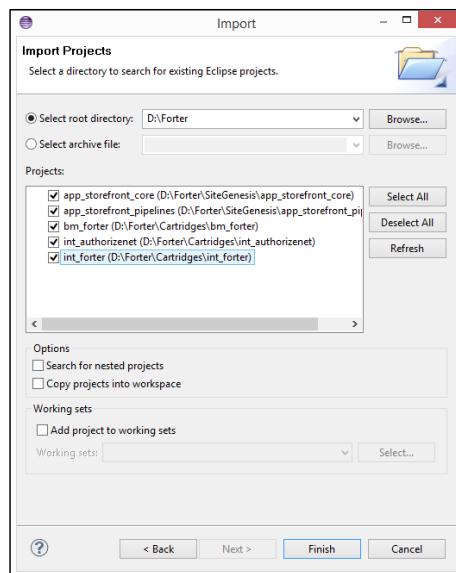
2. Select the “Existing Project into Workspace” option



3. Import Projects -> Select archive file (or “root directory”) if you have already unzipped the cartridge



4. Select the archive from your local source (or the unzipped cartridge)



5. Select all, click Finish and then click Yes in order to complete the import and link the cartridge to the DW server.

3.2.2 *Metadata import*

In the Metadata folder you will find a zip file called 'site_template'.

Go to Administration > Site development > Site Import & Export and upload the zip file.

Select the zip you uploaded, click on Import then on the ok button

The screenshot shows a 'Site Import & Export' page. A modal dialog box is open, asking 'Are you sure that you want to import the selected archive?'. Below the dialog, there is a file selection area with a 'Choose File' button and an 'Upload' button. A file named 'site_template.zip' is selected. To the right of the file list, there are columns for 'Location', 'File Size', and 'Last Modified'. At the bottom of the dialog are 'Import' and 'Delete' buttons, with 'Import' being highlighted with a red box.

3.2.3 Cartridge paths

Go to Administration > Sites > Manage Sites and choose your site. Click on the Settings tab, add in the cartridge path int_forter then click Apply.

The screenshot shows the 'SiteGenesis - Settings' page under 'Manage Sites'. The 'Settings' tab is selected. In the 'Cartridges' field, 'int_forter' is added to the list. Below it, the 'Effective Cartridge Path' field shows the full path including 'int_forter'. The 'Apply' and 'Reset' buttons at the bottom are highlighted with red boxes.

Add in the cartridge path int_forter_sfra in case if site is SFRA based.

The screenshot shows the 'RefArch - Settings' page under 'Manage Sites'. The 'Settings' tab is selected. In the 'Cartridges' field, 'int_forter_sfra' is added to the list. Below it, the 'Effective Cartridge Path' field shows the full path including 'int_forter_sfra'. The 'Apply' and 'Reset' buttons at the bottom are highlighted with red boxes.

Go to Administration > Sites > Manage Sites and click on Business Manager link.

The screenshot shows the 'Administration > Sites > Manage Sites' interface. The 'Storefront Sites' section lists two sites: 'SiteGenesis' and 'SiteGenesisGlobal'. Below it, the 'Business Manager Site' section has a link labeled 'Manage the Business Manager site' which is highlighted with a red box.

Add to the cartridge path `bm_forter` then click Apply.

The screenshot shows the 'Business Manager - Settings' page under the 'Settings' tab. In the 'Cartridges' field, the value `bm_custom_plugin bm_forter` is entered, with `bm_forter` highlighted by a red box. Other cartridges listed include `app_business_manager_plugin`, `apple_pay_plugin`, `facebook_plugin`, `pinterest_commerce_plugin`, and `web_payments_bp_im`.

3.2.4 Forter custom site preferences

A Site Preferences page is added in Business Manager to give merchant the ability to configure the Forter cartridge settings. This page can be accessed in Site Preferences > Custom Preferences > Forter:

- **Forter enabled** – Indicates whether the Forter code will be executed or not.
- **Show customized message when Forter has returned a decline decision and the order was cancelled immediately** – If enabled, this option means that a decline page (with customizable message) is shown in cases when Forter declines the order. This setting is disabled if **Cancel and void order when transaction is declined** is disabled.
- **Customized message** – If a message is entered and enabled, the message is shown on the decline page.
- **Auto-invoice when transaction is approved** – If enabled, when an “approve” decision is returned the payment gateway capture request is called and the order is placed.
- **Cancel and void order when transaction is declined** – If enabled then when Forter returns a “decline” decision, the order is Failed and a request is made to the payment processor to void the order. If this option is not selected, then in cases of a “decline” decision the order is placed (the order status is New).
- **Number of weeks** – the time range (number of weeks) that the Forter Order Update job queries in order to update order status. The default value is 4 weeks.

Force Forter decision – this preference is used to test customer requests for Forter services, it has five values, Disabled will keep this preference disabled which means the services will receive a normal request, this is the value to use on production. Go to Administration > Organization > Roles & Permissions. Choose the Administrator role and click on the Business Manager Modules tab. Select your site from the Select Context section. Select Forter modules and click Apply.

 Storefront Toolkit	Manage the storefront toolkit preferences for this site.	<input type="checkbox"/>	<input type="checkbox"/>
 Storefront URLs	Configure storefront URL preferences.	<input type="checkbox"/>	<input type="checkbox"/>
 Custom Preferences	Configure custom site preferences.	<input type="checkbox"/>	<input type="checkbox"/>
 Pinterest Commerce	Manage Pinterest Commerce configuration.	<input type="checkbox"/>	<input type="checkbox"/>
 Customer Service Center Preferences	Manage the Customer Service Center preferences for this site.	<input type="checkbox"/>	<input type="checkbox"/>
 Apple Pay	Manage Apple Pay configuration.	<input type="checkbox"/>	<input type="checkbox"/>
 Forter		<input checked="" type="checkbox"/>	<input type="checkbox"/>
 Order	Manage the orders.	<input type="checkbox"/>	<input type="checkbox"/>
 Configuration	Configure Forter Frictionless Fraud Prevention.	<input type="checkbox"/>	<input type="checkbox"/>

Configuration

Set your fraud prevention preferences.



Forter is linked to your account

Configure Forter

Enabled

Yes No

Communication

- Show customized message when Forter has returned a decline decision and the order was cancelled immediately

Customized message:

Your order has been declined

Payment Settings

- Auto-invoice when transaction is approved
- Cancel and void order when transaction is declined

Order Status Updates

Please choose the number of weeks after order placement when order status updates should be sent.

4

Change API credentials

Site ID

.....

Secret key

.....

Save

3.3 Custom Code

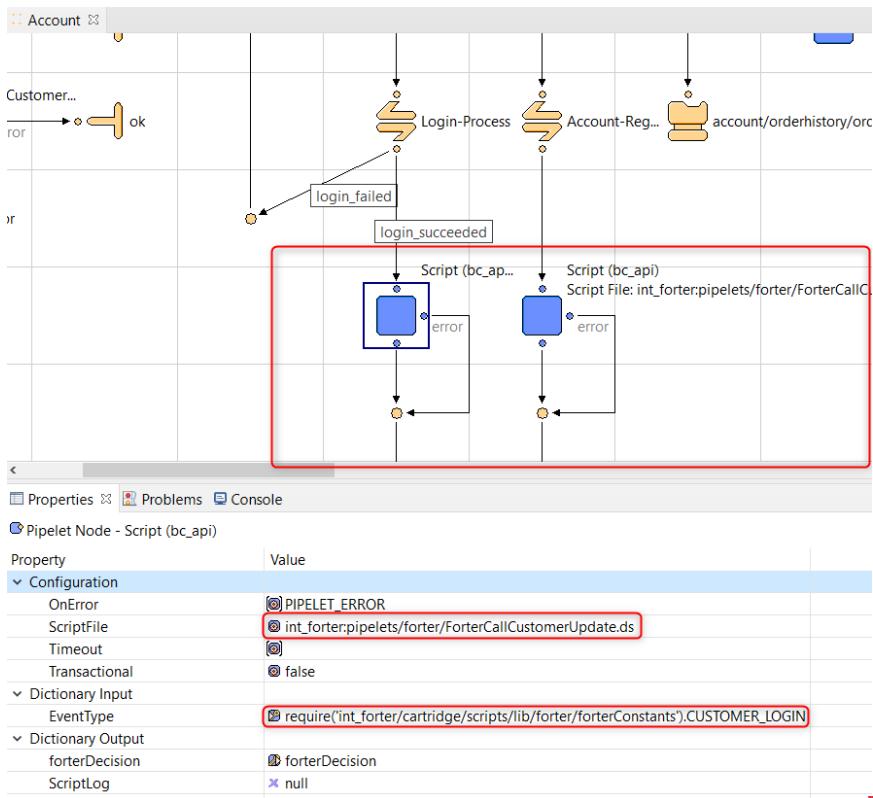
The Forter platform allows for custom attributes to be sent in the request. It is recommended that if you have attributes that are relevant for fraud detection, you should set them in the request objects generated in the cartridge.

3.3.1 Pipelines

In order to integrate the cartridge with Site Genesis based on pipelines, the pipelets/forter/ForterCallCustomerUpdate.ds has to be added to the following pipelines, each using its own eventType as input:

- Account-RequireLogin,

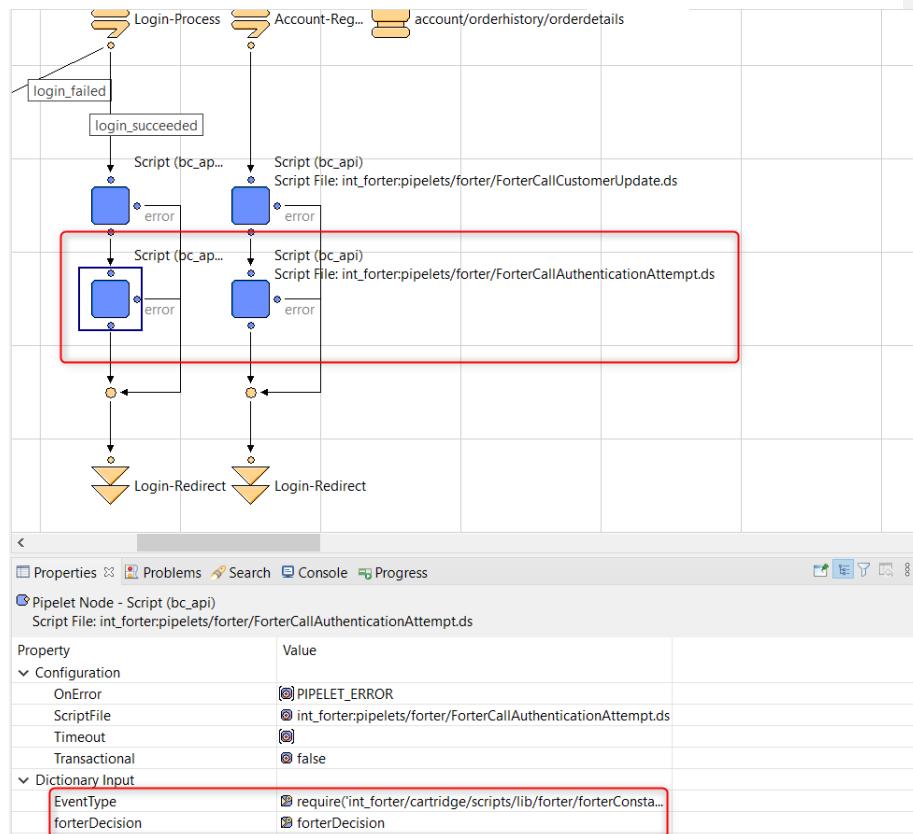
EventType require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_LOGIN



To call the authentication attempt service you'll need to add the following pipelet after the call to process the login

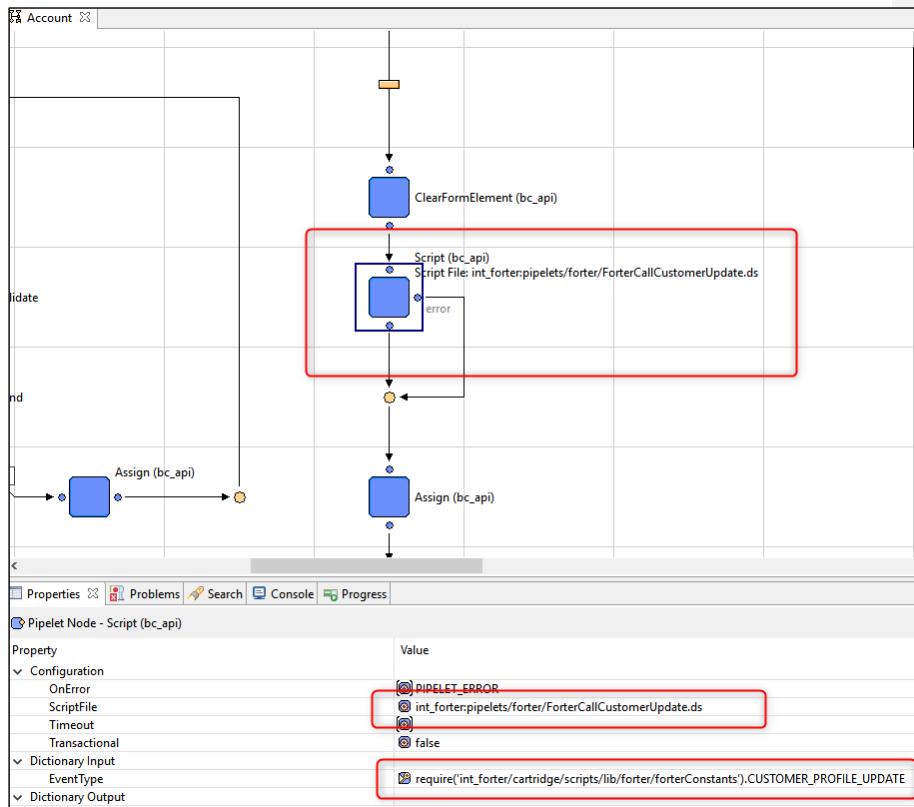
EventType require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_LOGIN

forterDecision is the Forter's response from the login call.



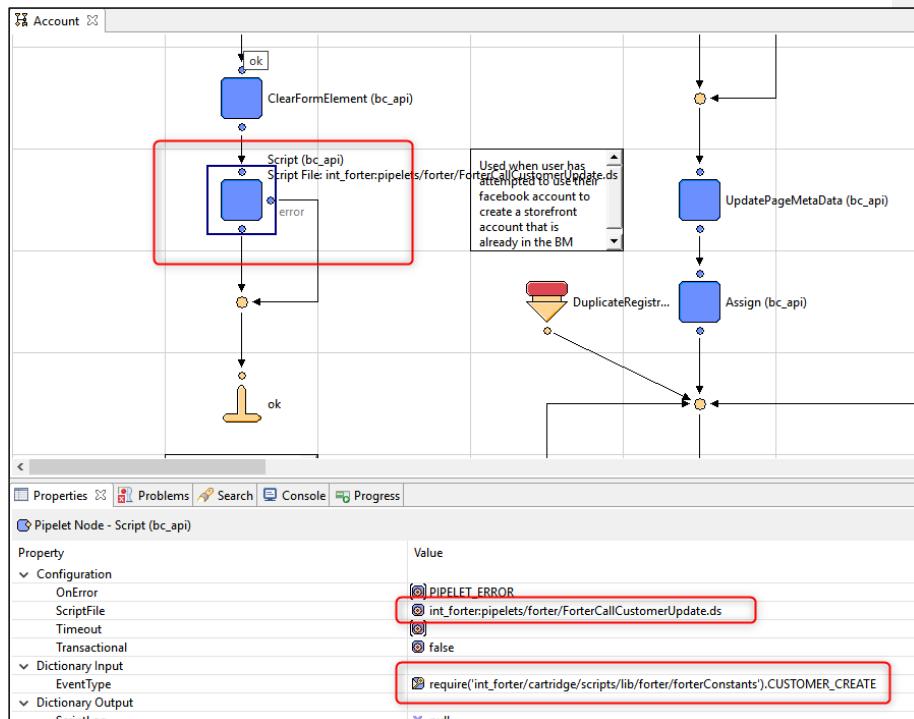
- Account-EditProfile,

EventType require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE



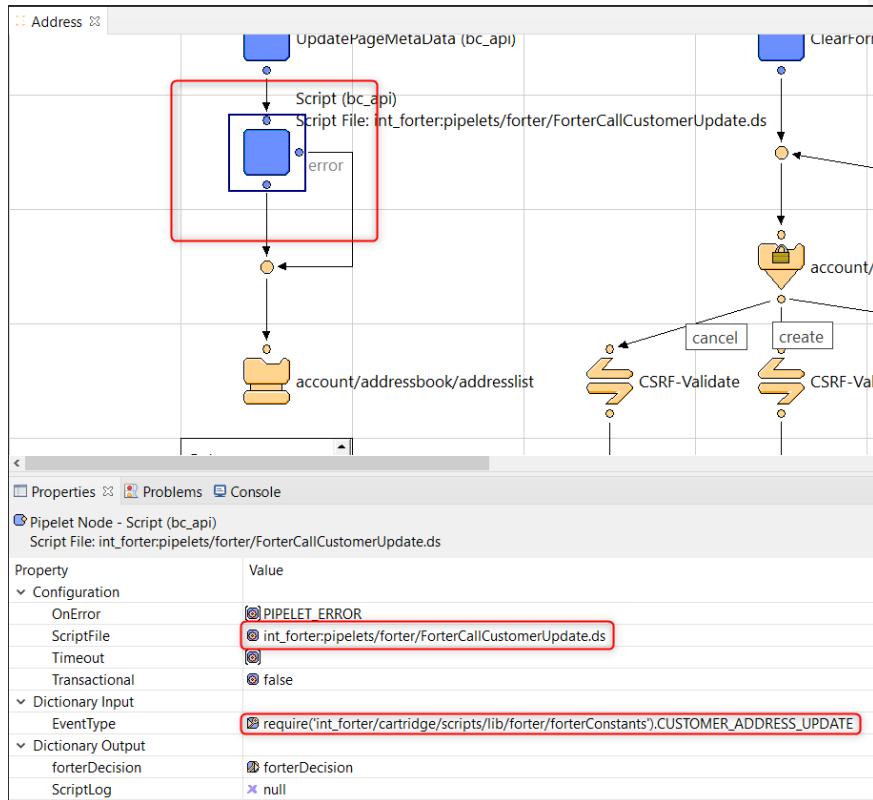
- Account-Register,

EventType require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE



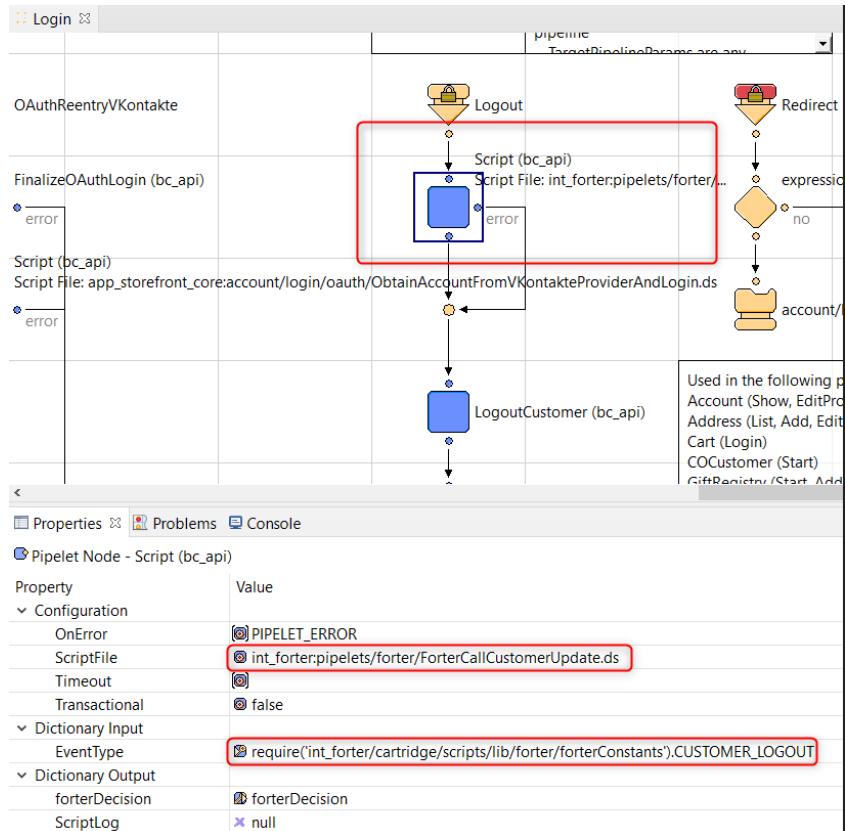
- Address-List, EventType

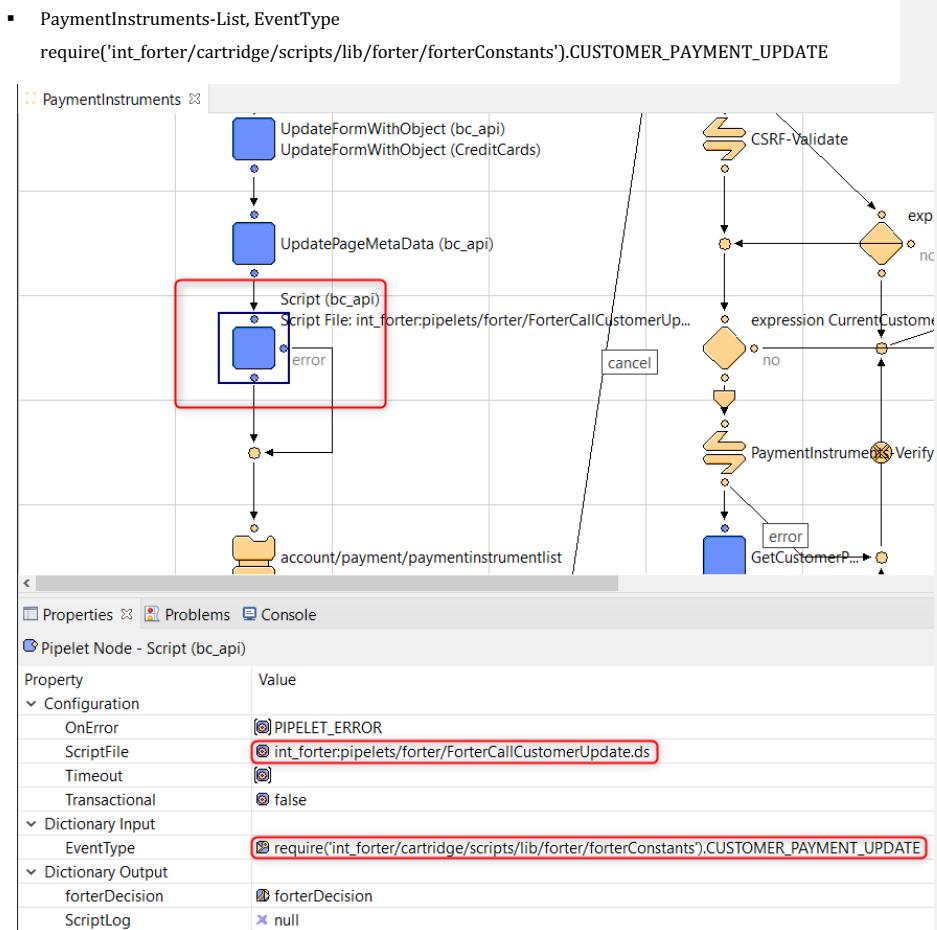
```
require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_ADDRESS_UPDATE
```



- Login-Logout, EventType

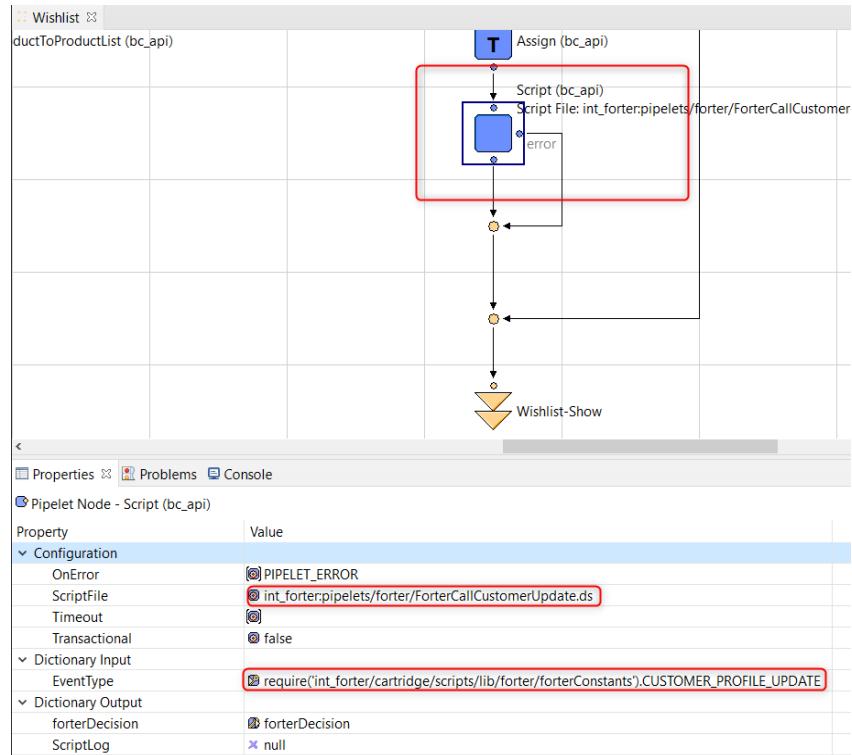
```
require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_LOGOUT
```





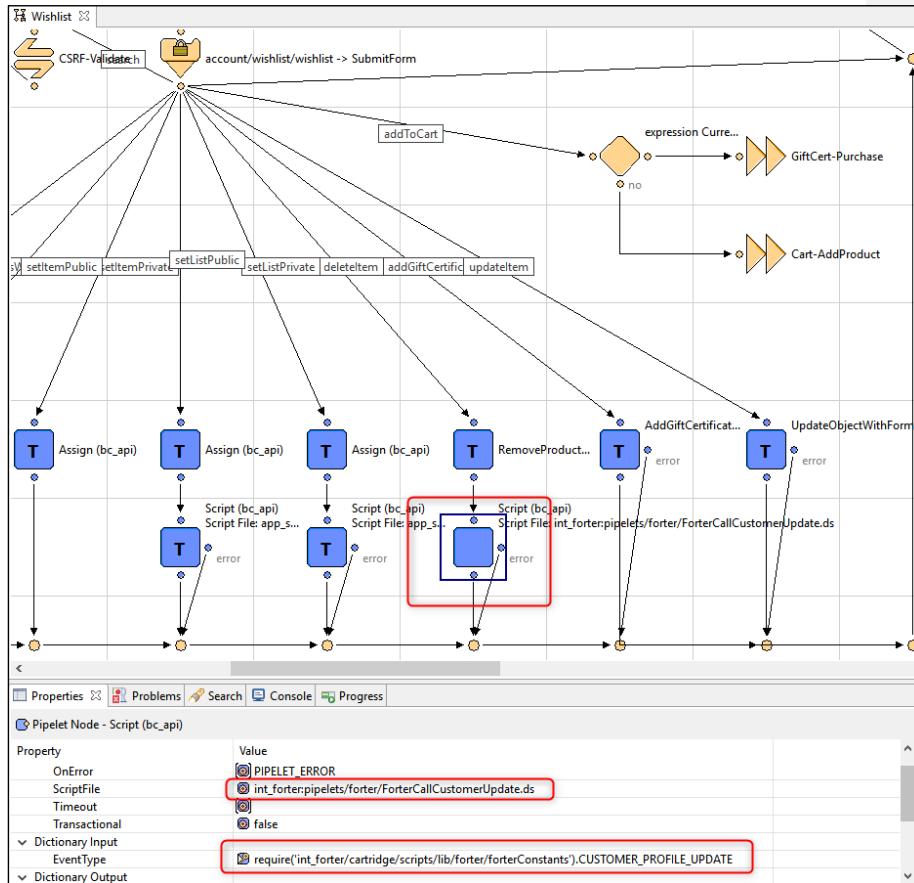
- Wishlist-Add, EventType

```
require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
```



- Wishlist-Show (in the 'deleteItem' transition), EventType

```
require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
```

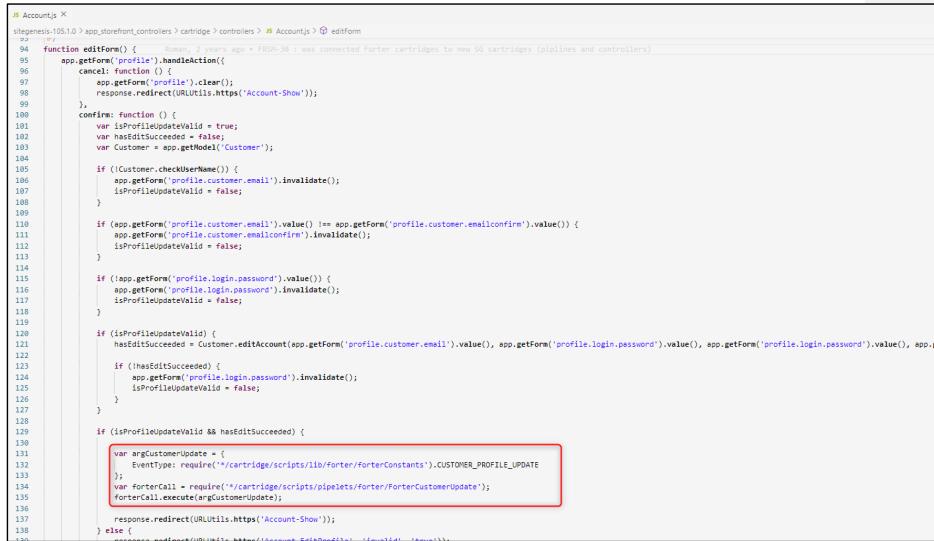


3.3.2 Controllers

In order to integrate the cartridge with Site Genesis based on controllers, the pipelets/forter/ForterCustomerUpdate.js has to be added to the following controllers:

- Account.js (in the editForm() function):

```
var argCustomerUpdate = {  
  EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE};  
var forterCall = require('../cartridge/scripts/pipelets/forter/ForterCustomerUpdate');  
forterCall.execute(argCustomerUpdate);
```



```
1 Account.js X  
sitegenesis-105.1.0 > app_storefront_controllers > cartridge > controllers > Account.js > editform  
122  
94  function editform() {  
95    // ...  
96    app.getForm('profile').handleAction(  
97      cancel: function () {  
98        app.getForm('profile').clear();  
99        response.redirect(URLUtils_https('Account>Show'));  
100      },  
101      confirm: function () {  
102        var isProfileUpdateValid = true;  
103        var hasEditSucceeded = false;  
104        var Customer = app.getModel('Customer');  
105  
106        if (!Customer.checkUserName()) {  
107          app.getForm('profile.customer.email').invalidate();  
108          isProfileUpdateValid = false;  
109        }  
110  
111        if (app.getForm('profile.customer.email').value() != app.getForm('profile.customer.emailConfirm').value()) {  
112          app.getForm('profile.customer.emailConfirm').invalidate();  
113          isProfileUpdateValid = false;  
114        }  
115  
116        if (app.getForm('profile.login.password').value()) {  
117          app.getForm('profile.login.password').invalidate();  
118          isProfileUpdateValid = false;  
119        }  
120  
121        if (isProfileUpdateValid) {  
122          hasEditSucceeded = Customer.editAccount(app.getForm('profile.customer.email').value(), app.getForm('profile.login.password').value(), app.getForm('profile.login.password').value());  
123        }  
124        if (!hasEditSucceeded) {  
125          app.getForm('profile.login.password').invalidate();  
126          isProfileUpdateValid = false;  
127        }  
128      }  
129    }  
130    if (isProfileUpdateValid && hasEditSucceeded) {  
131      var argCustomerUpdate = {  
132        EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE  
133      };  
134      var forterCall = require('../cartridge/scripts/pipelets/forter/ForterCustomerUpdate');  
135      forterCall.execute(argCustomerUpdate);  
136  
137      response.redirect(URLUtils_https('Account>Show'));  
138    } else {  
139      // ...  
140    }  
141  }  
142
```

- Account.js (in the registrationForm() function):

```
var argCustomerUpdate = {
  EventType: require('*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE};
  var forterCall = require('*cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);
```

JS Account.js X

sitegenesis-105.1.0 > app_storefront_controllers > cartridge > controllers > JS Account.js > registrationForm

```
383 function registrationForm() {
  // Mon, 2 years ago x FSH-30 : was connected Forter cartridges to new 56 cartridges (pipelines and controllers)
  384   app.getForm('profile').handleAction({
  385     confirm: function () {
  386       var email, emailConfirmation, orderNo, profileValidation, password, passwordConfirmation, existingCustomer, Customer, target;
  387
  388       Customer = app.getModel('Customer');
  389       email = app.getForm('profile.customer.email').value();
  390       emailConfirmation = app.getForm('profile.customer.emailconfirm').value();
  391       orderNo = app.getForm('profile.customer.orderNo').value();
  392       profileValidation = true;
  393
  394       if (email != emailConfirmation) {
  395         app.getForm('profile.customer.emailconfirm').invalidate();
  396         profileValidation = false;
  397       }
  398
  399       password = app.getForm('profile.login.password').value();
  400       passwordConfirmation = app.getForm('profile.login.passwordconfirm').value();
  401
  402       if (password != passwordConfirmation) {
  403         app.getForm('profile.login.passwordconfirm').invalidate();
  404         profileValidation = false;
  405       }
  406
  407       // Checks if login is already taken.
  408       existingCustomer = Customer.retrieveCustomerByLogin(email);
  409       if (existingCustomer != null) {
  410         app.getForm('profile.customer.email').invalidate();
  411         profileValidation = false;
  412       }
  413
  414       if (profileValidation) {
  415         profileValidation = Customer.createAccount(email, password, app.getForm('profile'));
  416
  417         var argCustomerUpdate = {
  418           EventType: require('*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE
  419         };
  420         var forterCall = require('*cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
  421         forterCall.execute(argCustomerUpdate);
  422
  423         if (orderNo) {
  424           // Mon, 2 years ago x FSH-30 : was connected Forter cartridges to new 56 cartridges (pipelines and controllers)
  425           // Eds: Scoringd (OSF Global), 4 months ago x integrated Forter with the SiteGenesis controller
  426           var argCustomerUpdate = {
  427             EventType: require('*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
  428           };
  429           var forterCall = require('*cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
  430           forterCall.execute(argCustomerUpdate);
  431
  432           app.getView().render('account/addressbook/addresslist');
  433
  434         }
  435
  436       }
  437     }
  438   }
}
```

- Address.js (in the list() function):

```
var argCustomerUpdate = {
  EventType:
    require('*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_ADDRESS_UPDATE};
  var forterCall = require('*cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);
```

JS Address.js X

sitegenesis-105.1.0 > app_storefront_controllers > cartridge > controllers > JS Address.js > list

```
21 */
22 function list() {
23   var pageMeta = require('~/cartridge/scripts/meta');
24
25   var content = app.getModel('Content').get('myaccount-addresses');
26   if (content) {
27     pageMeta.update(content.object);
28   }
29   // Eds: Scoringd (OSF Global), 4 months ago x integrated Forter with the SiteGenesis controller
30   var argCustomerUpdate = {
31     EventType: require('*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
32   };
33   var forterCall = require('*cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
34   forterCall.execute(argCustomerUpdate);
35
36   app.getView().render('account/addressbook/addresslist');
37 }
```

- Login.js (in the handleLoginForm() function):

```
var forterConstants = require('*/*cartridge/scripts/lib/forter/forterConstants');

var argCustomerUpdate = {
  EventType: forterConstants.CUSTOMER_LOGIN
};

var forterCall = require('*/*cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
var forterDecision = forterCall.execute(argCustomerUpdate);

You only need to include this call for the authentication attempt API, if you have an advanced authentication method used for MFA or OTP, in this case you'll need to provide the information returned from your MFA in this request.

if (forterDecision == forterConstants.STATUS_VERIFICATION_REQ) {
  var argAuthenticationAttemptUpdate = {
    EventType: forterConstants.CUSTOMER_AUTH_ATTEMPT
  };

  // example of object with MFA results
  argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
    verificationOutcome: '<Your MFA outcome result>',
    correlationId: '<result from MFA>',
    emailVerification: {
      email: customer.profile.email,
      emailRole: 'ACCOUNT',
      sent: true,
      verified: true
    }
  };
  var forterAuthAttempCall = require('*/*cartridge/scripts/pipelinelets/forter/ForterAuthenticationAttemptUpdate');
  forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
}
```

```

  96     loginForm.handleSubmit(function() {
  97       login: function () {
  98         // Check to see if the number of attempts has exceeded the session threshold
  99         if (RateLimiter.isOverThreshold('FailedLoginCounter')) {
 100           RateLimiter.showCaptcha();
 101         }
 102 
 103         var success = Customer.login(loginForm.getValue('username'), loginForm.getValue('password'), loginForm.getValue('rememberme'));
 104 
 105         if (success) {
 106           loginForm.get('loginsucceeded').invalidate();
 107           app.getView('Login').render();
 108           return;
 109         } else {
 110           loginForm.clear();
 111         }
 112 
 113         RateLimiter.hideCaptcha();
 114 
 115         var forterConstants = require('/cartridge/scripts/lib/forter/forterConstants');
 116         var argCustomerUpdate = {
 117           EventType: forterConstants.CUSTOMER_LOGIN
 118         };
 119         var forterCall = require('/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
 120         var forterResult = forterCall.execute(argCustomerUpdate);
 121 
 122         if (forterResult.forterDecision === forterConstants.STATUS_VERIFICATION_REQ) {
 123           var argAuthenticationAttemptUpdate = {
 124             EventType: forterConstants.CUSTOMER_AUTH_ATTEMPT
 125           };
 126 
 127           // example of object populated with MFA results.
 128           argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
 129             verificationOutcome: 'SUCCESS',
 130             correlationId: 'M07512345SHD0E',
 131             emailVerification: {
 132               customer: customer.profile.email,
 133               emailable: 'ACCOUNT',
 134               sent: true,
 135               verified: true,
 136             }
 137           };
 138           var forterAuthAttempCall = require('/cartridge/scripts/pipelets/forter/ForterAuthenticationAttemptUpdate');
 139           forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
 140         }
 141       }
 142     }
 143   }
 144 
```

- Paymentinstruments.js (in the list() function):

```

var argCustomerUpdate = {
  EventType: require('/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PAYMENT_UPDATE};
var forterCall = require('/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
  
```

```

  28   - CUSTOMER (account/payment/paymentinstrumentlist template).
  29   */
 30   function list() {
 31     var wallet = customer.getProfile().getWallet();
 32     var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD_CREDIT_CARD);
 33     var (local var) paymentForm: any @/scripts/meta';
 34     var paymentForm = app.getForm('paymentinstruments');
 35 
 36     paymentForm.clear();
 37     paymentForm.get('creditcards.storedcards').copyFrom(paymentInstruments);
 38 
 39     pageMeta.update(dw.content.ContentMgr.getContent('myaccount-paymentsettings'));
 40 
 41     var argCustomerUpdate = {
 42       EventType: require('/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
 43     };
 44     var forterCall = require('/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
 45     forterCall.execute(argCustomerUpdate);
 46 
 47     app.getView({
 48       PaymentInstruments: paymentInstruments
 49     }).render('account/payment/paymentinstrumentlist');
 50   }
 51 
```

- Wishlist.js (in the add() function):

```

var argCustomerUpdate = {
  EventType:
    require('/cartridge/scripts/lib/forter/ForterConfig.ds').ForterConfig.CUSTOMER_PROFILE_UPDATE};
  
```

```
var forterCall = require('* /cartridge/scripts/pipeline/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS Wishlist.js X
sitegenesis-105.1.0 > app_storefront_controllers > cartridge > controllers > JS Wishlist.js > add > [x] argCustomerUpdate > ↗ EventType
223 function add() {
224     addProduct();
225
226     var argCustomerUpdate = {
227         EventType: require('* /cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
228     };
229     var forterCall = require('* /cartridge/scripts/pipeline/forter/ForterCustomerUpdate');
230     forterCall.execute(argCustomerUpdate);
231
232     response.redirect(dw.web.URLUtils.https('wishlist-show'));
233 }
234
235 function search () {
236     app.getForm('wishlist.search').clear();
237     app.getView({
238         ContinueURL: URLUtils.https('wishlist-WishListForm')
239     }).render('account/wishlist/wishlistresults');
240 }
241
```

- Wishlist.js (in the wishListForm() function):

```
var argCustomerUpdate = {
  EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE };
var forterCall = require('../cartridge/scripts/pipeline/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS Wishlist.js ×
steegenesis-105.1.0 > app storefront controllers > cartridge > controllers > JS Wishlist.js > wishListForm
67 function wishListForm() {
68   var productList = app.getModel('ProductList').get();
69   var shouldRedirectToShow = true;
70
71   app.getForm(request.triggeredForm.formId).handleAction({
72     addGiftCertificate: function () {
73       Transaction.wrap(function () {
74         productList.createGiftCertificateItem();
75       });
76     },
77     deleteItem: function (formgroup, action) {
78       var sessionCustomer = session.customer;
79       if (sessionCustomer.ID === action.object.list.owner.ID) {
80         productList.removeItem(action.object);
81         var argCustomerUpdate = {
82           EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
83         };
84         var forterCall = require('../cartridge/scripts/pipeline/forter/ForterCustomerUpdate');
85         forterCall.execute(argCustomerUpdate);
86       }
87     },
88     updateItem: function (formgroup, action) {
89       var sessionCustomer = session.customer;
90       if (sessionCustomer.ID === action.object.list.owner.ID) {
91         app.getForm(action.parent).copyTo(action.object);
92       }
93     },
94     setListPrivate: function () {
95       // No need to check for productList ownership at this point because the productList
96       // retrieved from the shopper's profile and in the currently implementation the shopper
97       // can only have one productList of type TYPE_WISH_LIST
98       productList.setPublic(false);
99     }
100   });
101 }
```

- ForterOrder.js (SG) and forterOrder.js(SFRA)

In the ForterPayment function we need to add back this piece of code

```
this.billingDetails.personalDetails.email = order.customerEmail;
this.billingDetails.address = {};
this.billingDetails.address.address1 = billingAddress.address1;
this.billingDetails.address.address2 = !empty(billingAddress.address2) ? billingAddress.address2 : '';
this.billingDetails.address.zip = billingAddress.postalCode;
this.billingDetails.address.city = billingAddress.city;
this.billingDetails.address.region = billingAddress.stateCode;
this.billingDetails.address.country = billingAddress.countryCode.value;
```

```

function ForterPayment(order, authResponse, payment, log) {
    var billingAddress = order.billingAddress;

    this.billingDetails = {};
    this.billingDetails.personalDetails = {};
    this.billingDetails.personalDetails.firstName = billingAddress.firstName;
    this.billingDetails.personalDetails.lastName = billingAddress.lastName;
    this.billingDetails.personalDetails.email = order.customerEmail;

    this.billingDetails.address = {};
    this.billingDetails.address.address1 = billingAddress.address1;
    this.billingDetails.address.address2 = !empty(billingAddress.address2) ? billingAddress.address2 : '';
    this.billingDetails.address.zip = billingAddress.postalCode;
    this.billingDetails.address.city = billingAddress.city;
    this.billingDetails.address.region = billingAddress.stateCode;
    this.billingDetails.address.country = billingAddress.countryCode.value;
    VolleyError: No message
    if (!empty(billingAddress.phone)) {
        this.billingDetails.phone = [];
        this.billingDetails.phone.push(new ForterPhone(billingAddress.phone));
    }

    this.amount = {
        amountLocalCurrency: order.totalGrossPrice.value.toFixed(2),
        currency: order.totalGrossPrice.currencyCode
    };
}

```

- ForterOrder.js (SG) and forterOrder.js(SFRA)

For the current API version (2.88) in the function ForterPhone need to remove the smsVerified object and the function need to be like this:

code to be removed:

```

this.smsVerified = {
    sent: false,
    verified: false
};

function ForterPhone(phone) {
    this.phone = phone;
}

```

- forterConstants.js (both SG and SFRA)

Change the value for CUSTOMER_LOGOUT from login to logout

```

module.exports = {
  STATUS_APPROVED: 'APPROVED',
  STATUS_DECLINED: 'DECLINED',
  STATUS_NOT_REVIEWED: 'NOT_REVIEWED',
  STATUS_VERIFICATION_REQ: 'VERIFICATION_REQUIRED',
  STATUS_ORDER_APPROVE: 'APPROVE',
  STATUS_ORDER_DECLINE: 'DECLINE',
  STATUS_FAILED: 'FAILED',
  STATUS_NOT_SENT: 'NOT_SENT',
  CUSTOMER_LOGIN: 'login',
  CUSTOMER_LOGOUT: 'logout', IonutHrincescuOSF,
  CUSTOMER_CREATE: 'signup',
  CUSTOMER_PROFILE_UPDATE: 'update',
  CUSTOMER_PROFILE_ACCESS: 'profile-access',
  CUSTOMER_AUTH_RESULT: 'authentication-result',
  CUSTOMER_AUTH_ATTEMPT: 'authentication-attempt',
  PHONE_TYPE_PRIMARY: 'Primary',
  PHONE_TYPE_SECONDARY: 'Secondary',
  PHONE_DESC_HOME: 'Home',
  PHONE_DESC_WORK: 'Work',
  PHONE_DESC_MOBILE: 'Mobile',
  ORDER_VALIDATEST: 'validates'
},

```

- **ForterValidate.js**

Add a new functionality for new JavaScript Snippet from which the form can be taken directly from script and not taken from the session via cookie.

```

function updateForterInfo() {
  if (!empty(request.httpParameterMap.ftrToken) && !empty(request.httpParameterMap.ftrToken.value)) {
    session.privacy.ftrToken = request.httpParameterMap.ftrToken.value;
  }
  let r = require('bm_forter/cartridge/scripts/util/Response.js');
  r.renderJSON({
    success: true
  });
  return;
}

function updateForterInfo() {
  if (!empty(request.httpParameterMap.ftrToken) && !empty(request.httpParameterMap.ftrToken.value)) {
    session.privacy.ftrToken = request.httpParameterMap.ftrToken.value;
  }
  let r = require('bm_forter/cartridge/scripts/util/Response.js');
  r.renderJSON([
    success: true
  ]);
  return; You, 2 weeks ago * FRSM-103 Version 2 ...
}

```

3.3.3 SFRA

This section describes integration of the cartridge with site based on SFRA structure and provides a list of all affected controllers. Integration of the cartridge into storefront application does not imply modifications of the core cartridge, SFRA approach of files overwriting should be used instead. In case if storefront application has

same controllers or templates extended or replaced – all mentioned code modifications must be added to the top level cartridge of the storefront application.

- Account.js (prepends the 'Login' with next code include):

```
var forterConstants = require('*cartridge/scripts/lib/forter/forterConstants');
var argCustomerUpdate = {
  EventType: forterConstants.CUSTOMER_LOGIN,
  customer: customer,
  request: request
};
var forterCall = require('*cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
var forterDecision = forterCall.execute(argCustomerUpdate);
```

You only need to include this call for the authentication attempt API, if you have an advanced authentication method used for MFA or OTP, in this case you'll need to provide the information returned from your MFA in this request.

```
if (forterDecision == forterConstants.STATUS_VERIFICATION_REQ) {
  var argAuthenticationAttemptUpdate = {
    EventType: forterConstants.CUSTOMER_AUTH_ATTEMPT
  };

  // example of object with MFA results
  argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
    verificationOutcome: '<Your MFA outcome result>',
    correlationId: '<result from MFA>',
    emailVerification: {
      email: customer.profile.email,
      emailRole: 'ACCOUNT',
      sent: true,
      verified: true
    }
  };
  var forterAuthAttempCall = require('*cartridge/scripts/pipelinelets/forter/ForterAuthenticationAttemptUpdate');
  forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
}
```

```

js Account.js ✘
cartridges > int_forter_sfra > cartridge > controllers > JS Account.js > server.prepend('Login') callback > on('routeBeforeComplete') callback > [6] forterAuthAttempCall
  1
  2   server.prepend(
  3     'Login',
  4     server.middleware.https,
  5     csrfProtection.validateAjaxRequest,
  6     function (req, res, next) {
  7       this.on('routeBeforeComplete', function (req, res) { // eslint-disable-line no-shadow
  8         var viewData = res.getViewData();
  9
 10        if (viewData.authenticatedCustomer) {
 11          var forterConstants = require('../cartridge/scripts/lib/forter/forterConstants');
 12          var argCustomerUpdate = {
 13            EventType: forterConstants.CUSTOMER_LOGIN
 14          };
 15          var forterCall = require('../cartridge/scripts/pipelinelets/forter/forterCustomerUpdate');
 16
 17          var forterResult = forterCall.execute(argCustomerUpdate);
 18
 19          if (forterResult.forterDecision === forterConstants.STATUS_VERIFICATION_REQ) {
 20            var forterAuthAttempCall = require('../cartridge/scripts/pipelinelets/forter/forterAuthenticationAttemptUpdate');
 21            var argAuthenticationAttemptUpdate = {
 22              EventType: forterConstants.CUSTOMER_AUTH_ATTEMPT,
 23
 24              // example of object populated with MFA results.
 25              argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
 26                verificationOutcome: 'SUCCESS',
 27                correlationId: 'H037512345H30E',
 28                emailVerification: {
 29                  email: customer.profile.email,
 30                  emailRole: 'ACCOUNT',
 31                  sent: true,
 32                  verified: true,
 33                }
 34              }
 35            }
 36
 37            forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
 38
 39          }
 40        }
 41      });
 42
 43      return next();
 44    }
 45  );
 46
 47  );
 48
 49  );
 50  );
 51  );
}

```

- Account.js (appends the 'SubmitRegistration' with next code include):

```
var argCustomerUpdate = {
  EventType: require('*/*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE
};

var forterCall = require('*/*cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS Account.js ×
cartridges > int_forter_sfra > cartridge > controllers > JS Account.js > ...
41   server.append(
42     'SubmitRegistration',
43     server.middleware.https,
44     csrfProtection.validateAjaxRequest,
45     function (req, res, next) {
46       this.on('route:BeforeComplete', function (req, res) { // eslint-disable-line no-shadow
47         var viewData = res.getViewData();
48
49         if (viewData.authenticatedCustomer) {
50           var argCustomerUpdate = {
51             EventType: require('*/*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE
52           };
53           var forterCall = require('*/*cartridge/scripts/pipelinelets/forter/forterCustomerUpdate');
54
55           forterCall.execute(argCustomerUpdate);
56         }
57       });
58
59       return next();
60     }
61   );
62 }
```

- Account.js (appends the 'SaveProfile' with next code include):

```
var argCustomerUpdate = {
  EventType: require('*/*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
};

var forterCall = require('*/*cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS Account.js ×
cartridges > int_forter_sfra > cartridge > controllers > JS Account.js > ...
63   server.append(
64     'SaveProfile',
65     server.middleware.https,
66     csrfProtection.validateAjaxRequest,
67     function (req, res, next) {
68       this.on('route:BeforeComplete', function (req, res) { // eslint-disable-line
69         var CustomerMgr = require('dw/customer/CustomerMgr');
70         var customer = CustomerMgr.getCustomerByCustomerNumber(
71           req.currentCustomer.profile.customerNo
72         );
73         var profile = customer.getProfile();
74
75         if (profile) {
76           var argCustomerUpdate = {
77             EventType: require('*/*cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
78           };
79           var forterCall = require('*/*cartridge/scripts/pipelinelets/forter/forterCustomerUpdate');
80
81           forterCall.execute(argCustomerUpdate);
82         }
83       });
84
85       return next();
86     }
87   );
```

- Address.js (appends the 'List' with next code include):

```
var argCustomerUpdate = {
  EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_ADDRESS_UPDATE
};

var forterCall = require('../cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS Address.js
cartridges > int_forter_sfra > cartridge > controllers > JS Address.js > ...
You, 4 days ago | 2 authors (VolodymyrNekhayOSF and others)
1  'use strict';
2
3  // Local Modules
4  var server = require('server');
5  server.extend(module.superModule);
6
7  server.append('List', function (req, res, next) {
8    var argCustomerUpdate = {
9      EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
10    };
11    var forterCall = require('../cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
12
13    forterCall.execute(argCustomerUpdate);
14
15    next();
16  });
17
18  module.exports = server.exports();
19 |
```

- PaymentInstruments.js (appends the 'List' with next code include):

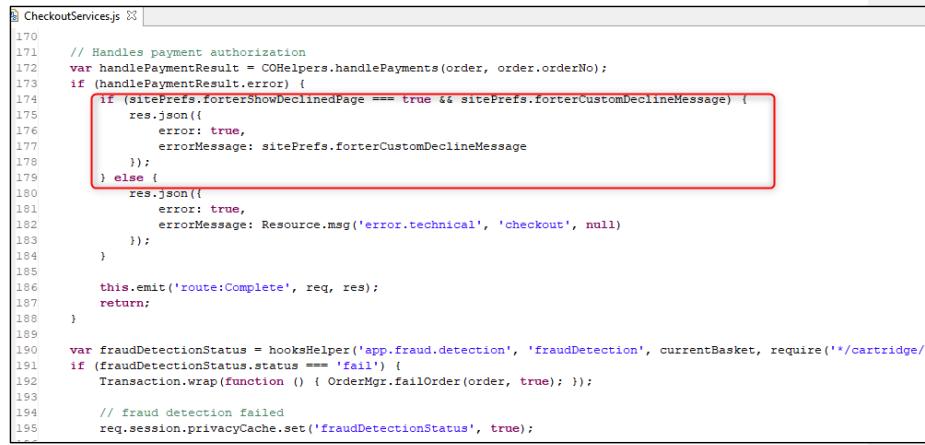
```
var argCustomerUpdate = {
  EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PAYMENT_UPDATE
};

var forterCall = require('../cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);
```

```
JS PaymentInstruments.js
cartridges > int_forter_sfra > cartridge > controllers > JS PaymentInstruments.js > server.append('List') callback > argCustomerUpdate > EventType
82
83  server.append('List', function (req, res, next) {
84    var argCustomerUpdate = {
85      EventType: require('../cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
86    };
87    var forterCall = require('../cartridge/scripts/pipelinelets/forter/ForterCustomerUpdate');
88
89    forterCall.execute(argCustomerUpdate);
90
91    next();
92  });
93 |
```

- CheckoutServices.js (prepends the 'PlaceOrder' with next code includes) in order to handle the customized error message configured in Forter business manager extension:

```
if(sitePrefs.forterShowDeclinedPage === true && sitePrefs.forterCustomDeclineMessage) {
    res.json({
        error: true,
        errorMessage: sitePrefs.forterCustomDeclineMessage
    });
}
```



```
CheckoutServices.js 33
170 // Handles payment authorization
171 var handlePaymentResult = COHelpers.handlePayments(order, order.orderNo);
172 if (handlePaymentResult.error) {
173     if (sitePrefs.forterShowDeclinedPage === true && sitePrefs.forterCustomDeclineMessage) {
174         res.json({
175             error: true,
176             errorMessage: sitePrefs.forterCustomDeclineMessage
177         });
178     } else {
179         res.json({
180             error: true,
181             errorMessage: Resource.msg('error.technical', 'checkout', null)
182         });
183     }
184     this.emit('route:Complete', req, res);
185     return;
186 }
187
188 var fraudDetectionStatus = hooksHelper('app.fraud.detection', 'fraudDetection', currentBasket, require('/cartridge/
189 if (fraudDetectionStatus.status === 'fail') {
190     Transaction.wrap(function () { OrderMgr.failOrder(order, true); });
191
192 // fraud detection failed
193 req.session.privacyCache.set('fraudDetectionStatus', true);
```

- int_forter_sfra/cartridge/templates/default/common/layout/checkout.isml (extends the 'app_storefront_base/cartridge/templates/default/common/layout/checkout.isml' template with next code include):


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



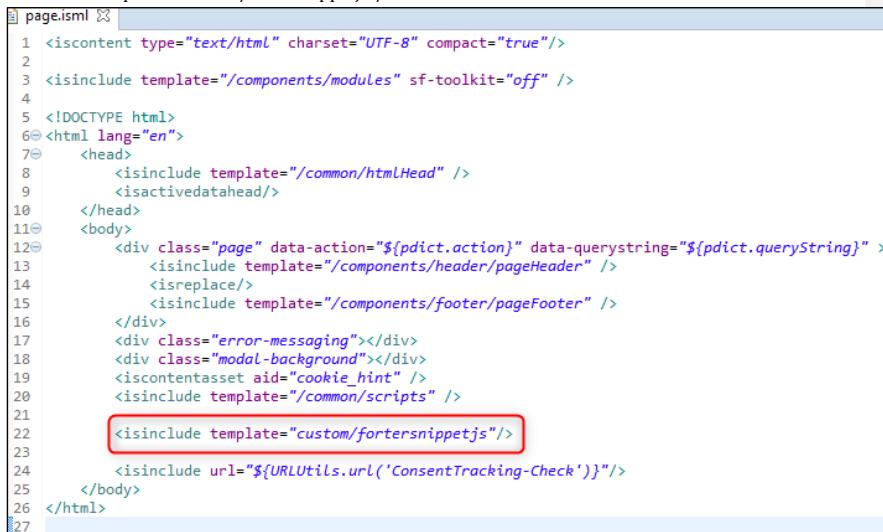
```

1 <iscontent type="text/html" charset="UTF-8" compact="true"/>
2
3 <isinclude template="/components/modules" sf-toolkit="off" />
4
5 <!DOCTYPE html>
6@ <html lang="en">
7@   <head>
8     <isinclude template="/common/htmlHead" />
9     <isactivedatahead/>
10  </head>
11@  <body>
12@    <div class="page">
13      <isinclude template="/components/header/pageHeaderNomenu" />
14      <isreplace/>
15      <isinclude template="/components/footer/pageFooter" />
16    </div>
17    <isinclude template="/common/scripts" />
18
19    <isinclude template="custom/fortersnippetjs"/>
20
21    <isinclude url="${URLUtils.url('ConsentTracking-Check')}" />
22
23 </body>
24
25 </html>
26
27

```

- int_forter_sfra/cartridge/templates/default/common/layout/page.isml (extends the 'app_storefront_base/cartridge/templates/default/common/layout/page.isml' template with next code include):


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



```

1 <iscontent type="text/html" charset="UTF-8" compact="true"/>
2
3 <isinclude template="/components/modules" sf-toolkit="off" />
4
5 <!DOCTYPE html>
6@ <html lang="en">
7@   <head>
8     <isinclude template="/common/htmlHead" />
9     <isactivedatahead/>
10  </head>
11@  <body>
12@    <div class="page" data-action="${pdict.action}" data-querystring="${pdict.queryString}" >
13      <isinclude template="/components/header/pageHeader" />
14      <isreplace/>
15      <isinclude template="/components/footer/pageFooter" />
16    </div>
17    <div class="error-messaging"></div>
18    <div class="modal-background"></div>
19    <iscontentasset aid="cookie_hint" />
20    <isinclude template="/common/scripts" />
21
22    <isinclude template="custom/fortersnippetjs"/>
23
24    <isinclude url="${URLUtils.url('ConsentTracking-Check')}" />
25
26 </body>
27

```

3.3.4 Forter integration in Checkout / Payment Flow

This section explores what occurs when the ForterValidate order validation controller is implemented using sample implementations with authorize.net and Paypal. A merchant using a different payment processor *should customize this logic to fit the merchant's business needs and the Forter configuration*. It gets the information from a previously executed payment authorization request and current order details. A Forter order validation API call is made and, if successful, a response with the Forter decision is received and saved per order. Based on that decision and the configuration, the following scenarios can be executed:

- Decision "APPROVED" – if **Auto-invoice when transaction is approved** is enabled, then the payment capture amount operation is executed and the order is placed (via the decision node with condition `ForterResponse.JsonResponseOutput.processorAction === 'capture'` in the diagram below). If this option is not enabled, no capture is executed, and the order is just placed in Salesforce Commerce Cloud (via the decision node with condition `ForterResponse.JsonResponseOutput.processorAction === 'skipCapture'`).
- Decision "DECLINED" –
 - If only **Cancel and void order when transaction is declined** is checked, then the order is failed and a request to void the order is made to the processor. In this case **Show decline page when Forter has returned a decline decision and the order was cancelled** is enabled, then the buyer is directed to a decline page with a customized message.
 - If **Cancel and void order when transaction is declined** is not selected, then the buyer will be directed to the "thank you" page and an order will be placed (via the decision node with condition `ForterResponse.JsonResponseOutput.processorAction === 'skipCapture'`).
- Decision "NOT REVIEWED" – by default a "Not Reviewed" decision will be routed to the decision node with condition `ForterResponse.JsonResponseOutput.processorAction === 'notReviewed'`. *In order to customize the behavior for this flow, use the `ForterResponse.JsonResponseOutput.processorAction` to split it from the "skipCapture" flow and insert the merchant specific logic for a "Not Reviewed" use case.*

This table represents all the possible output values based on the Forter decision and configuration saved from the Forter Business manager extension.

Forter Decision	Auto-invoice when transaction is approved	Cancel and void order when transaction is declined	ForterResponse.JsonResponseOutput.processorAction
APPROVED	ON	-	capture
	OFF	-	skipCapture
DECLINED	-	ON	void
	-	OFF	skipCapture
NOT REVIEWED	-	-	notReviewed
	-	-	notReviewed

Additional Flows

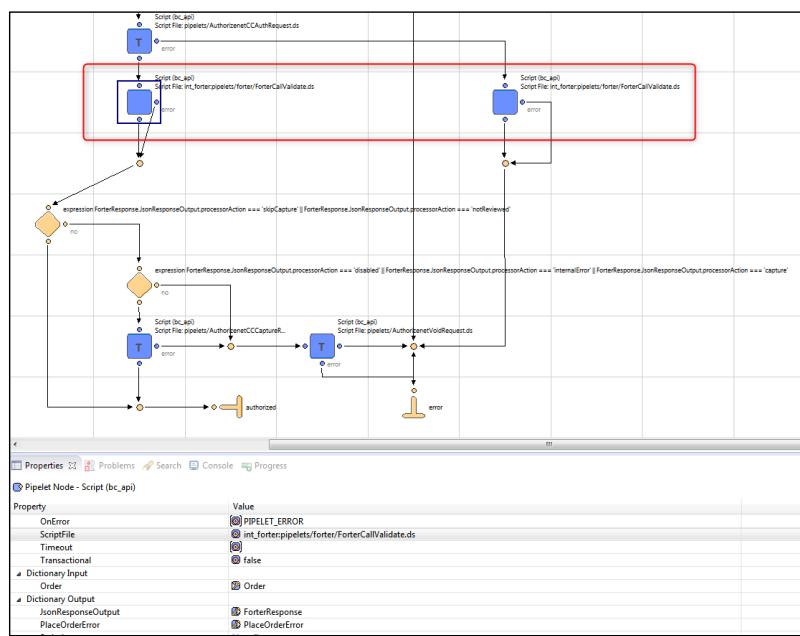
Payment Gateway error (processor declined authorization) - Please note in the example authorize.net diagram below, in case the credit card is not authorized by the payment gateway (e.g. expired credit card), the order is still sent to Forter. The request to Forter in this case should get a "NOT REVIEWED" decision (ForterResponse.JsonResponseOutput.processorAction === 'notReviewed') and the user should get routed to a payment error flow. Please note, it's important to verify your processor "authorization declined" codes are properly mapped and sent to Forter via this flow on the sandbox environment before moving to production.

Forter Cartridge is disabled - Forter will not return a decision. *The merchant should customize this logic according to his preferences and desired flow without Forter.* In the diagram below, you can see that if the order is sent to ForterCallValidate.ds script node we will route it via the decision node with condition ForterResponse.JsonResponseOutput.processorAction === 'disabled' so the order will be captured and finalized.

"Internal cartridge ERROR" – *This should not happen, if it does, please contact Forter customer support. The merchant should customize this logic according to his preferences.* In the example below, the decision node with condition ForterResponse.JsonResponseOutput.processorAction === 'internalError' is configured so the order is still finalized and captured.

Sample Authorize.net checkout flow (pipelines based)

The diagram below is from the Authorize.net "AUTHORIZE_NET-Authorize" Pipeline which is triggered as part of the generic Site Genesis authorization flow.



Sample Authorize.net checkout flow (controllers based)

The code below is from the Authorize.net "AUTHORIZE_NET-Authorize" controller which is triggered as part of the generic Site Genesis authorization flow.

```
function Authorize(args) {
    if (empty(session.forms.billing.paymentMethods.selectedPaymentMethodID.value)) {
        return {error: true};
    }

    var orderNo          = args.OrderNo,
        paymentInstrument = args.PaymentInstrument,
        paymentProcessor   =
            PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor();

    Transaction.wrap(function () {
        paymentInstrument.paymentTransaction.transactionID      = orderNo;
        paymentInstrument.paymentTransaction.paymentProcessor = paymentProcessor;
    });

    var argCCAuth = {
        Order           : args.Order,
        PaymentInstrument : paymentInstrument
    },
        authResponse = doAuth(argCCAuth);

    if (authResponse.result == false) {
        var argOrderValidate = {
            Order: args.Order,
            orderValidateAttemptInput: 1,
            request: request
        },
            forterController = require('int_forter/cartridge/controllers/ForterValidate'),
            forterDecision  = forterController.ValidateOrder(argOrderValidate);
        // in case if no response from Forter, try to call one more time
        if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
            var argOrderValidate = {
                Order: args.Order,
                orderValidateAttemptInput: 2,
                request: request
            },
                forterController = require('int_forter/cartridge/controllers/ForterValidate'),
                forterDecision  = forterController.ValidateOrder(argOrderValidate);
        }

        if (!empty(forterDecision.PlaceOrderError)) {
            return {error: true, forterErrorCode : forterDecision.PlaceOrderError};
        } else {
            return {error: true};
        }
        return {error: true};
    }

    if (authResponse.result == true) {
        var argOrderValidate = {
            Order: args.Order,
            orderValidateAttemptInput: 1,
            request: request
        },
            forterController = require('int_forter/cartridge/controllers/ForterValidate'),
            forterDecision  = forterController.ValidateOrder(argOrderValidate);
        // in case if no response from Forter, try to call one more time
        if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
            var argOrderValidate = {
                Order: args.Order,
                orderValidateAttemptInput: 2,
                request: request
            },
                forterController = require('int_forter/cartridge/controllers/ForterValidate'),
                forterDecision  = forterController.ValidateOrder(argOrderValidate);
        }

        if ((forterDecision.JsonResponseOutput.processorAction      ===      'skipCapture' ||
            forterDecision.JsonResponseOutput.processorAction === 'notReviewed') {
            return {authorized: true};
        }
    }
}
```

```

        } else if (forterDecision.JsonResponseOutput.processorAction === 'disabled' || 
forterDecision.JsonResponseOutput.processorAction === 'internalError' || 
forterDecision.JsonResponseOutput.processorAction === 'capture') {
    var argCCCapture = {
        AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
        Order : args.Order,
        PaymentInstrument : paymentInstrument
    },
    captureResponse = doCapture(argCCCapture);

    if (captureResponse.result == true) {
        return {authorized: true};
    }

    if (captureResponse.result == false) {
        var argVoid = {
            AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
            Order : args.Order,
            PaymentInstrument : paymentInstrument
        },
        voidResponse = doVoid(argVoid);

        if (!empty(forterDecision.PlaceOrderError)) {
            return {error : true, forterErrorCode : forterDecision.PlaceOrderError};
        } else {
            return {error : true};
        }
    } else {
        var argVoid = {
            AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
            Order : args.Order,
            PaymentInstrument : paymentInstrument
        },
        voidResponse = doVoid(argVoid);

        if (!empty(forterDecision.PlaceOrderError)) {
            return {error : true, forterErrorCode : forterDecision.PlaceOrderError};
        } else {
            return {error : true};
        }
    }
}

function doAuth(argCCAuth) {
    var authorizenetCCAuthRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetCCAuthRequest'),
        authResponse = authorizenetCCAuthRequest.execute(argCCAuth);
    return authResponse;
}

function doCapture(argCCCapture) {
    var authorizenetCCCaptureRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetCCCaptureRequest'),
        captureResponse = authorizenetCCCaptureRequest.execute(argCCCapture);
    return captureResponse;
}

function doVoid(argVoid) {
    var authorizenetVoidRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetVoidRequest'),
        voidResponse = authorizenetVoidRequest.execute(argVoid);
    return voidResponse;
}

```

Sample Authorize.net checkout flow (SFRA based)

The code below is from the Authorize.net "AUTHORIZE_NET-Authorize" controller which is triggered as part of authorization flow.

```
function Authorize(orderNumber, paymentInstrument, paymentProcessor) {
    var serverErrors = [],
        fieldErrors = {},
        error        = false;

    try {
        Transaction.wrap(function () {
            paymentInstrument.paymentTransaction.setTransactionID(orderNumber);
            paymentInstrument.paymentTransaction.setPaymentProcessor(paymentProcessor);
        });

        var argCCAuth = {
            orderNumber : orderNumber,
            PaymentInstrument : paymentInstrument
        },
            authResponse = doAuth(argCCAuth);

        if (authResponse.result === false) {
            var argOrderValidate = {
                orderNumber : orderNumber,
                orderValidateAttemptInput : 1,
                request: request
            },
                forterCall = require('int_forter_sfra/cartridge/scripts/pipelinelets/forter/ForterValidate'),
                forterDecision = forterCall.validateOrder(argOrderValidate);

            // in case if no response from Forter, try to call one more time
            if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
                var argOrderValidate = {
                    orderNumber : orderNumber,
                    orderValidateAttemptInput : 2,
                    request: request
                },
                    forterCall = require('int_forter_sfra/cartridge/scripts/pipelinelets/forter/ForterValidate'),
                    forterDecision = forterCall.validateOrder(argOrderValidate);
            }
            error = true;
            serverErrors.push(
                Resource.msg('error.technical', 'checkout', null)
            );
        }
    }

    if (authResponse.result === true) {
        var argOrderValidate = {
            orderNumber : orderNumber,
            orderValidateAttemptInput : 1,
            request: request
        },
            forterCall = require('int_forter_sfra/cartridge/scripts/pipelinelets/forter/ForterValidate'),
            forterDecision = forterCall.validateOrder(argOrderValidate);

        // in case if no response from Forter, try to call one more time
        if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
            var argOrderValidate = {
                orderNumber : orderNumber,
                orderValidateAttemptInput : 2,
                request: request
            };
            forterCall = require('int_forter_sfra/cartridge/scripts/pipelinelets/forter/ForterValidate'),
            forterDecision = forterCall.validateOrder(argOrderValidate);
        }

        if (forterDecision.JsonResponseOutput.processorAction === 'skipCapture' ||
forterDecision.JsonResponseOutput.processorAction === 'notReviewed') {
            error = false;
        } else if (forterDecision.JsonResponseOutput.processorAction === 'disabled' ||
forterDecision.JsonResponseOutput.processorAction === 'internalError' ||
forterDecision.JsonResponseOutput.processorAction === 'capture') {
            var argCCCapture = {

```

```

        AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
        orderNumber          : orderNumber,
        PaymentInstrument   : paymentInstrument
    },
    captureResponse = doCapture(argCCCapture);

    if (captureResponse.result === true) {
        error = false;
    }

    if (captureResponse.result === false) {
        var argVoid = {
            AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
            orderNumber          : orderNumber,
            PaymentInstrument   : paymentInstrument
        },
        voidResponse = doVoid(argVoid);

        error = true;
        serverErrors.push(
            Resource.msg('error.technical', 'checkout', null)
        );
    }
} else {
    var argVoid = {
        AuthorizeNetResponse : authResponse.AuthorizeNetResponse,
        orderNumber          : orderNumber,
        PaymentInstrument   : paymentInstrument
    },
    voidResponse = doVoid(argVoid);

    error = true;
    serverErrors.push(
        Resource.msg('error.technical', 'checkout', null)
    );
}
}

} catch (e) {
    error = true;
    serverErrors.push(
        Resource.msg('error.technical', 'checkout', null)
    );
}

return { fieldErrors: fieldErrors, serverErrors: serverErrors, error: error };
}

function doAuth(argCCAuth) {
    var authorizenetCCAuthRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetCCAuthRequest'),
        authResponse             = authorizenetCCAuthRequest.execute(argCCAuth);
    return authResponse;
}

function doCapture(argCCCapture) {
    var authorizenetCCCaptureRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetCCCaptureRequest'),
        captureResponse         = authorizenetCCCaptureRequest.execute(argCCCapture);
    return captureResponse;
}

function doVoid(argVoid) {
    var authorizenetVoidRequest = require('~/cartridge/scripts/pipelinelets/AuthorizenetVoidRequest'),
        voidResponse           = authorizenetVoidRequest.execute(argVoid);
    return voidResponse;
}

```

Sample Paypal Express Checkout Flow

When integrating with Paypal, we suggest you modify your Papal Cartridge settings (under Site Preferences -> Customer Site Preference Groups) so Forter will be able to receive relevant information and the Forter decision will control whether a transaction is captured or voided. Please note we do not handle Billing Agreement checkout flow in this example.

- The Express Checkout should include Authorization before Forter is called. This can be done by setting the Payment Action to "Authorization" or by setting Run Authorization in case of Order to Yes.

Merchant Tools / Site Preferences / Custom Site Preference Groups / [Paypal ExpressCheckout/Credit/EasyPayments Configuration](#)

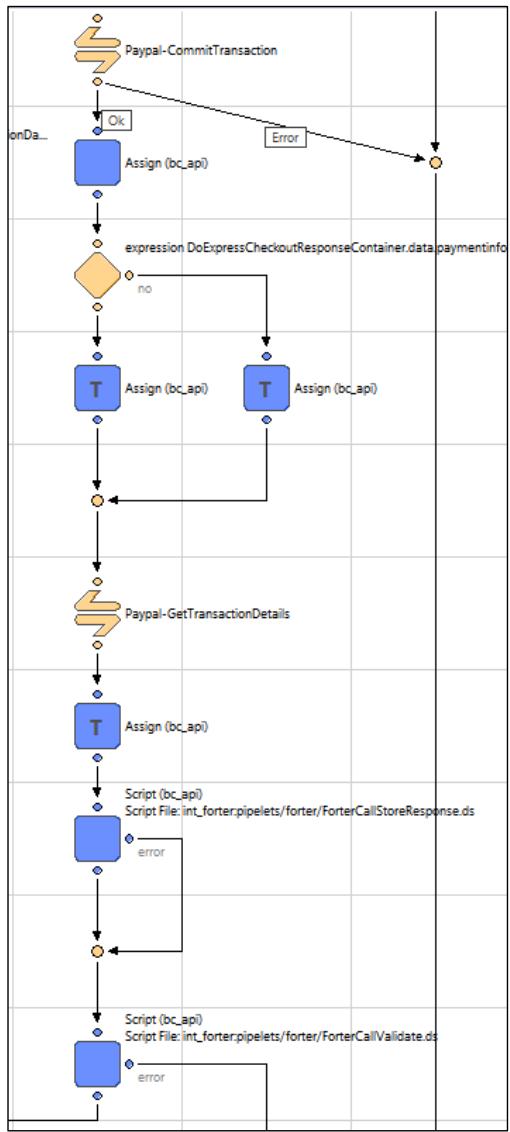
Name	Value	Default Value
Visibility of PayPal Express Checkout Button on the Cart page	Yes	No
Visibility of PayPal Credit Button on the Cart page	Yes	No
Payment Action	Authorization (Authorization)	Authorization
Reference Transaction Payment Action	None	Sale
Run Authorization in Case of Order	Yes	No
Billing Agreement State	Allow buyers to choose whether to create a billing agreement (BuyersChoose)	Do not create a billing agreement

- You must make sure the Paypal cartridge is set to send to Salesforce Commerce Cloud the billing information. Please note this configuration may need external permissions added by Paypal support to your Paypal account.

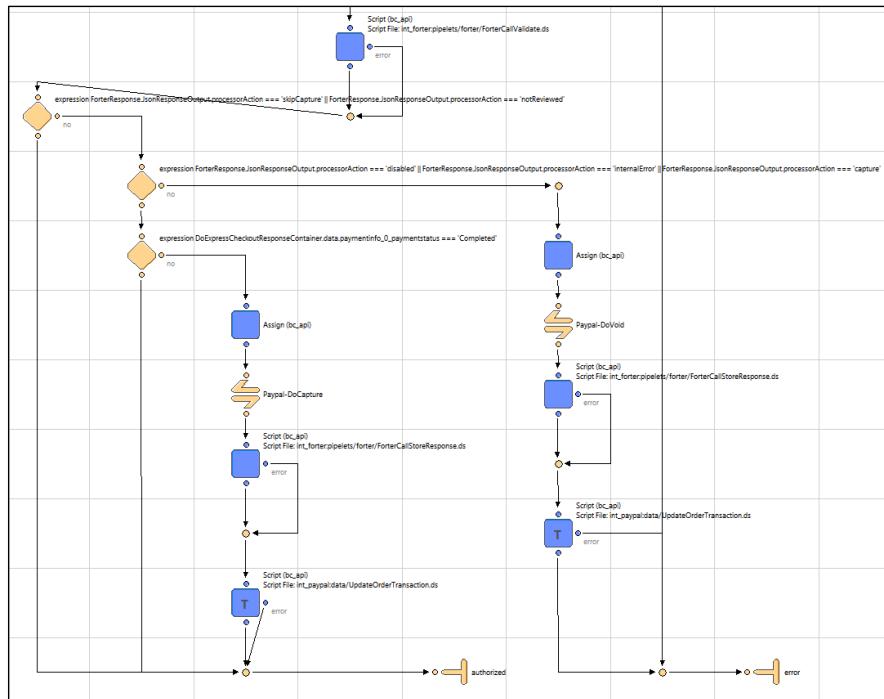
Shipping Address Override	None	No
Retrieve Billing Address From PayPal	Yes	Yes
Accept only confirmed shipping addresses	No	No

The diagrams below are from the PAYPAL_EXPRESS-Authorize Pipeline.

After the "CommitTransaction" is called Forter requests additional information about the transaction from Paypal by triggering the Paypal-GetTransactionDetails API call. After the details are stored Forter API is called in order to provide a decision on the order.

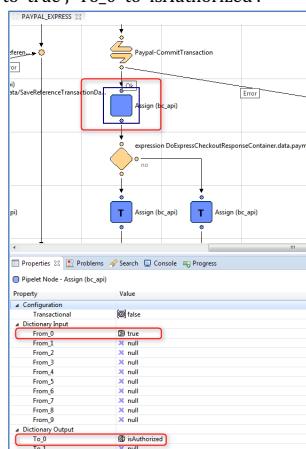


Based on the Forter Decision and the Forter Cartridge configuration, additional API calls are made to Paypal in order to capture or void the order as needed.

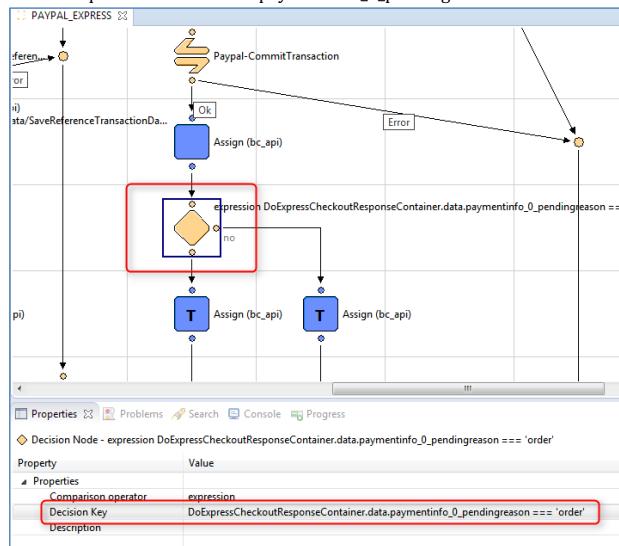


Below is detailed information about modifications from the previous images:

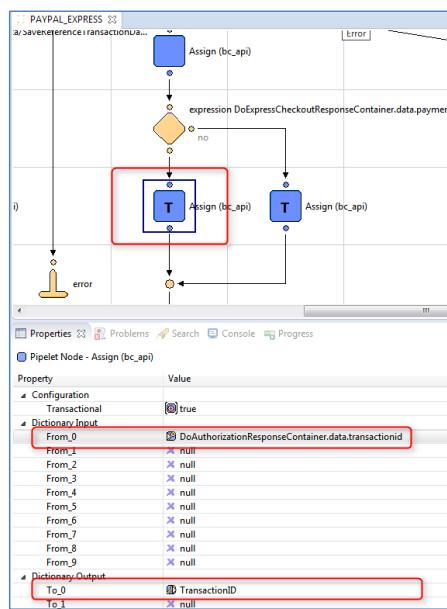
- Add an Assign node. Set 'From_0' to 'true'; 'To_0' to 'isAuthorized'.



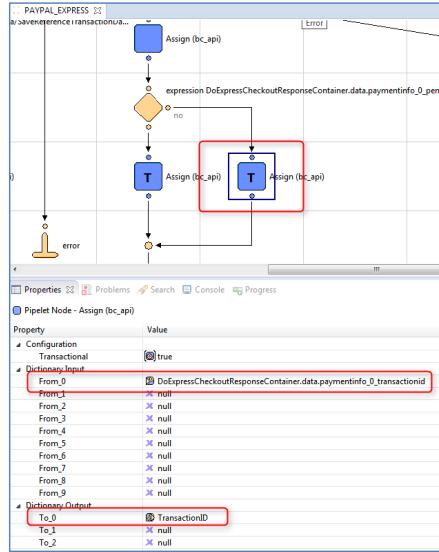
2. Add a Decision node. Set 'Decision Key' to
 'DoExpressCheckoutResponseContainer.data.paymentinfo_0_pendingreason === 'order''.



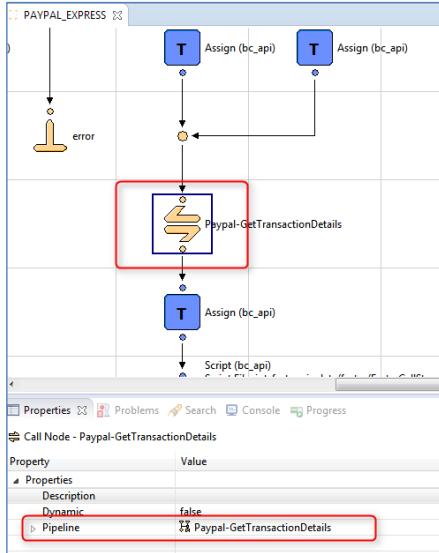
3. Add an Assign node. Set 'From_0' to 'DoAuthorizationResponseContainer.data.transactionid'; 'To_0' to 'TransactionID'.



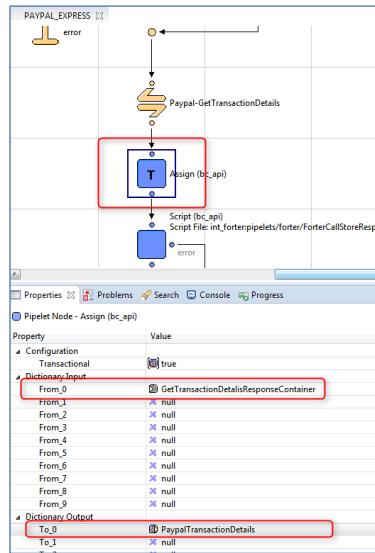
4. Add an Assign node. Set 'From_0' to 'DoExpressCheckoutResponseContainer.data.paymentinfo_0_transactionid'; 'To_0' to 'TransactionID'.



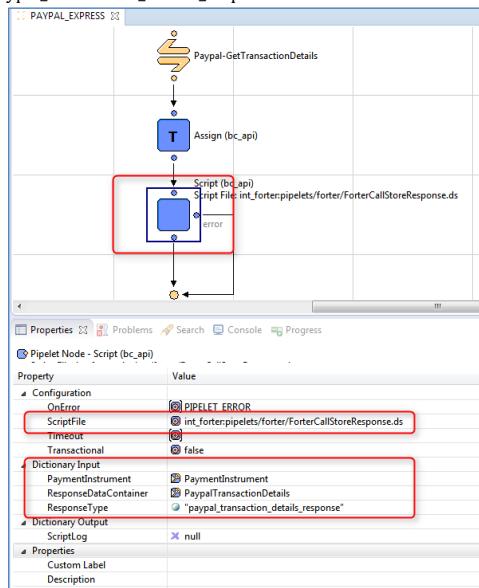
5. Add a Call node. Set 'Pipeline' to 'Paypal-GetTransactionDetails'.



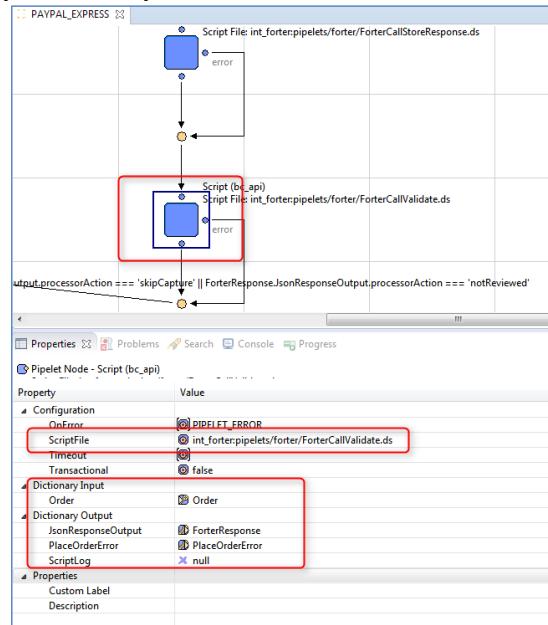
6. Add an Assign node. Set 'From_0' to 'GetTransactionDetailsResponseContainer'; 'To_0' to 'PaypalTransactionDetails'.



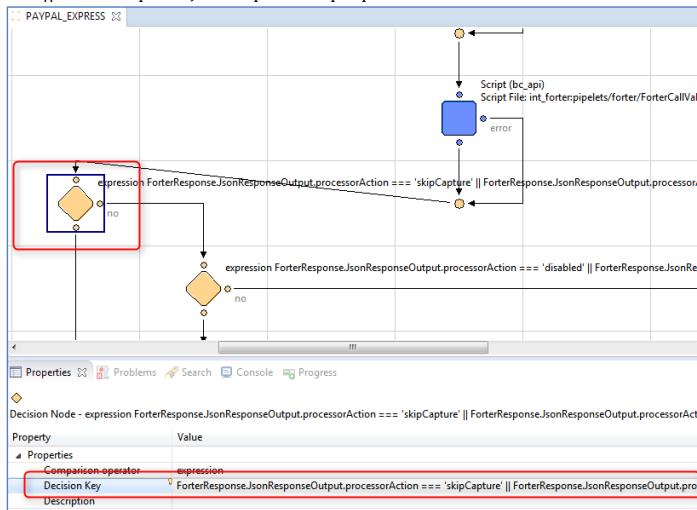
- Add a Script node. Set 'ScriptFile' to 'int_forter:pipeline/ForterCallStoreResponse.ds'; 'PaymentInstrument' to 'PaymentInstrument'; 'responseDataContainer' to 'PaypalTransactionDetails'; 'ResponseType' to "paypal_transaction_details_response".



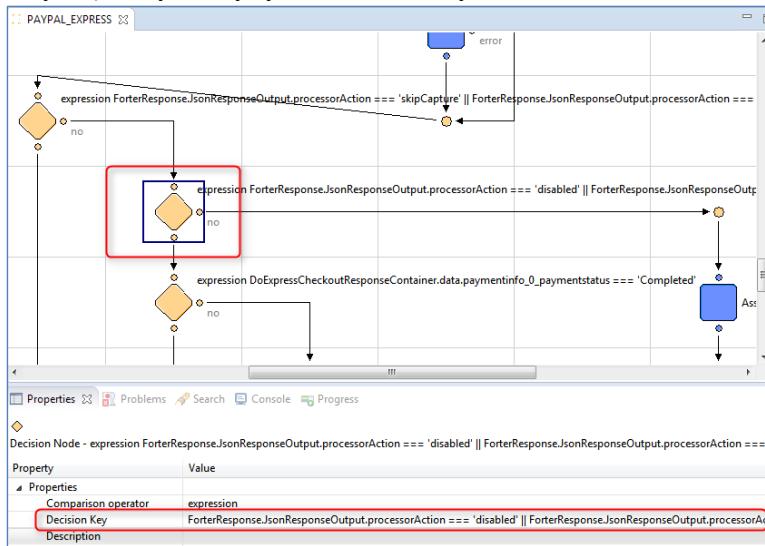
8. Add a Script node. Set 'ScriptFile' to 'int_forter:pipelets/forter/ForterCallValidate.ds'; 'Order' to 'Order'; 'JsonResponseOutput' to 'ForterResponse'; 'PlaceOrderError' to 'PlaceOrderError'.



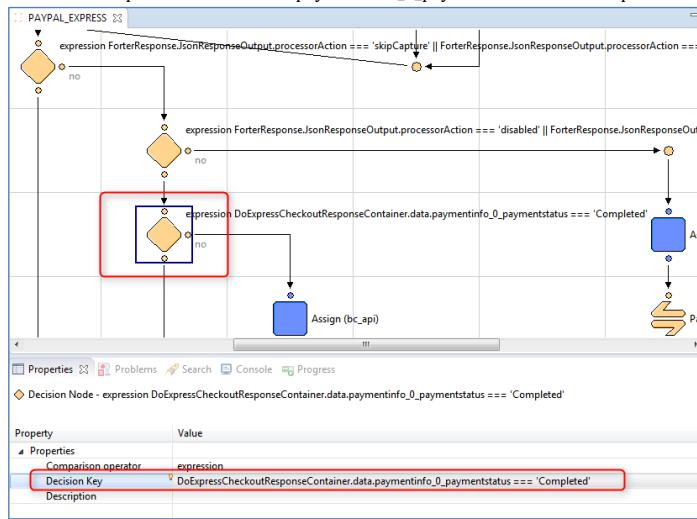
9. Add a Decision node. Set 'Decision Key' to 'ForterResponse.JsonResponseOutput.processorAction === 'skipCapture' || ForterResponse.JsonResponseOutput.processorAction === 'notReviewed"'.



10. Add a Decision node. Set 'Decision Key' to 'ForteResponse.JsonResponseOutput.processorAction === 'disabled' || ForterResponse.JsonResponseOutput.processorAction === 'internalError' || ForterResponse.JsonResponseOutput.processorAction === 'capture'.

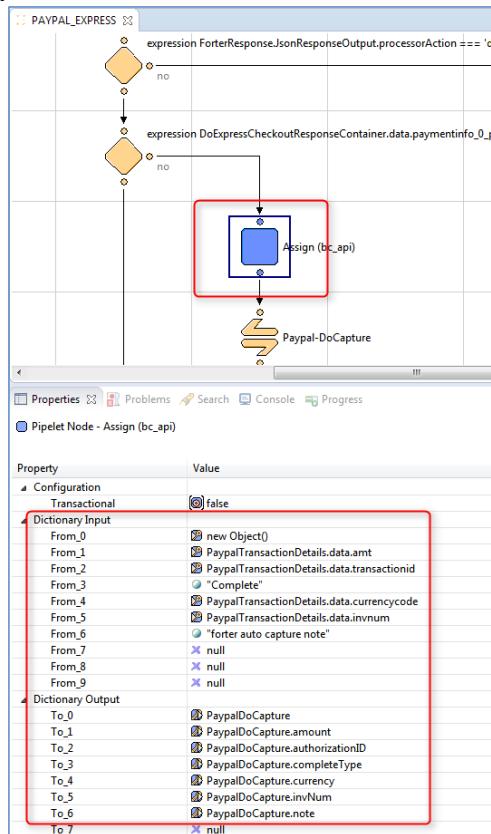


11. Add a Decision node. Set 'Decision Key' to 'DoExpressCheckoutResponseContainer.data.paymentinfo_0_paymentstatus === 'Completed''.
 'DoExpressCheckoutResponseContainer.data.paymentinfo_0_paymentstatus === 'Completed'.

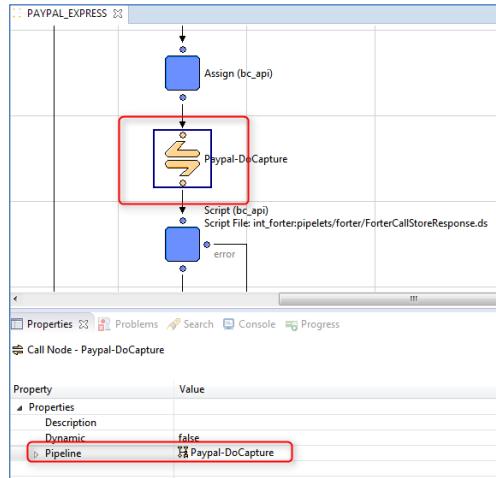


12. Add an Assign node. Set

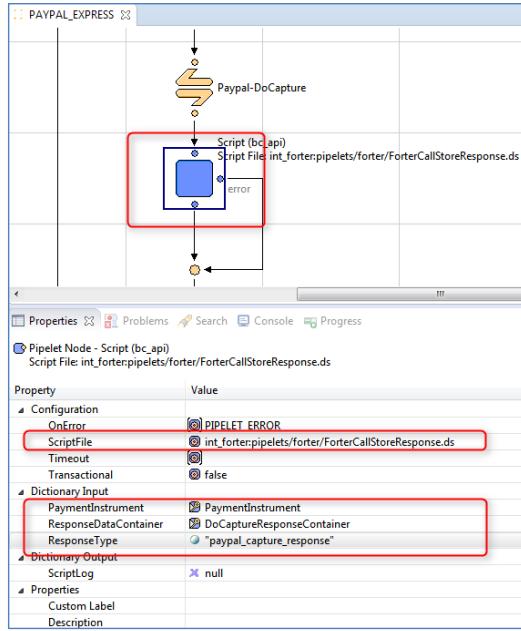
```
'From_0' to 'new Object()';
'From_1' to 'PaypalTransactionDetails.data.amt';
'From_2' to 'PaypalTransactionDetails.data.transactionid';
'From_3' to "Complete";
'From_4' to 'PaypalTransactionDetails.data.currencycode';
'From_5' to 'PaypalTransactionDetails.data.invnum';
'From_6' to "forter auto capture note";
'To_0' to 'PaypalDoCapture';
'To_1' to 'PaypalDoCapture.amount';
'To_2' to 'PaypalDoCapture.authorizationID';
'To_3' to 'PaypalDoCapture.completeType';
'To_4' to 'PaypalDoCapture.currency';
'To_5' to 'PaypalDoCapture.invNum';
'To_6' to 'PaypalDoCapture.note';
```



13. Add a Call node. Set 'Pipeline' to 'Paypal-DoCapture'.

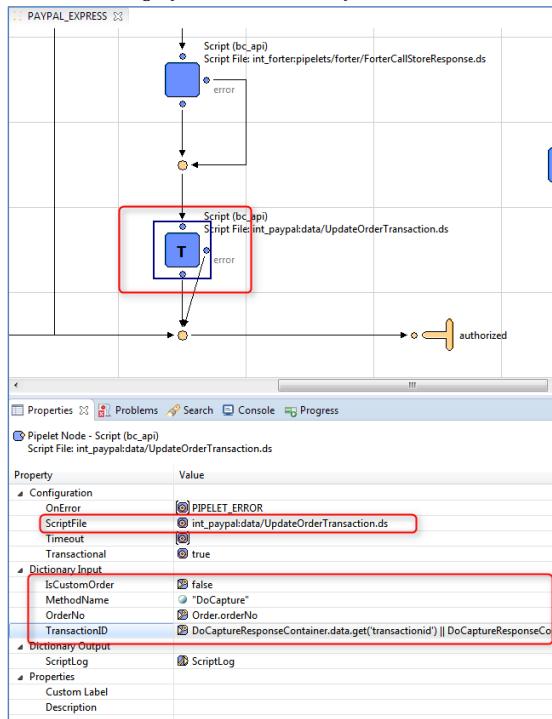


14. Add a Script node. Set 'ScriptFile' to 'int_forter:pipelets/forter/ForterCallStoreResponse.ds'; 'PaymentInstrument' to 'PaymentInstrument'; 'ResponseDataContainer' to 'DoCaptureResponseContainer'; 'ResponseType' to "paypal_capture_response".



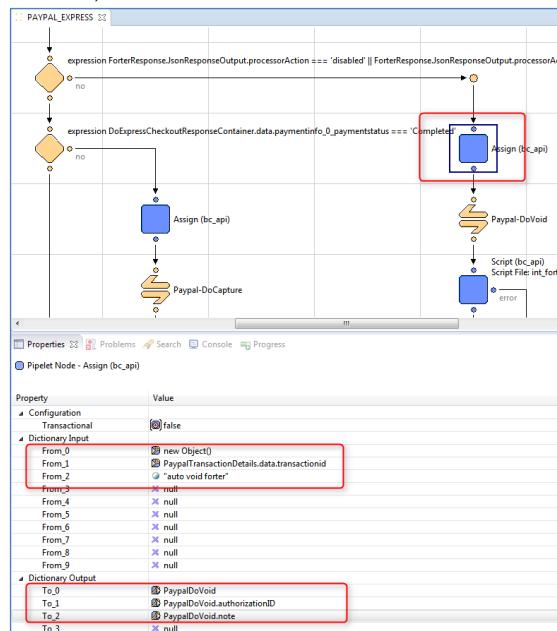
15. Add a Script node. Set

```
'ScriptFile' to 'int_paypal:data/UpdateOrderTransaction.ds';
'IsCustomOrder' to 'false';
'MethodName' to "DoCapture";
'OrderNo' to 'Order.orderNo';
'TransactionID' to 'DoCaptureResponseContainer.data.get('transactionid') ||
DoCaptureResponseContainer.data.get('authorizationid') ||
DoCaptureResponseContainer.data.get('refundtransactionid').
```

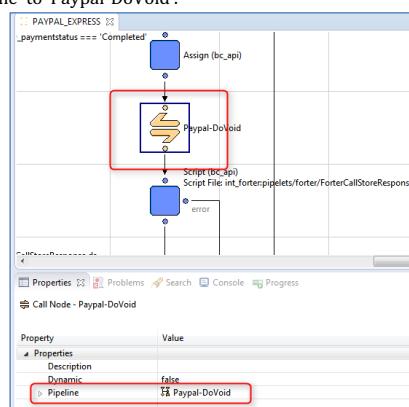


16. Add an Assign node. Set

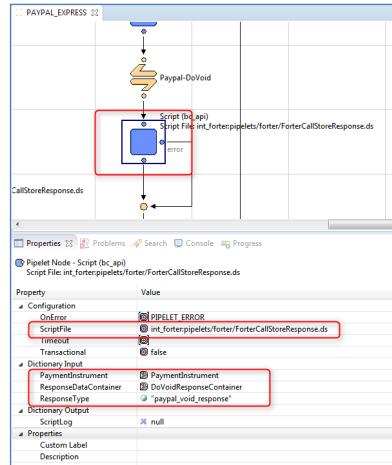
```
'From_0' to 'new Object';
'From_1' to 'PaypalTransactionDetails.data.transactionid';
'From_2' to "auto void forter";
'To_0' to 'PaypalDoVoid';
'To_1' to 'PaypalDoVoid.authorizationID';
'To_2' to 'PaypalDoVoid.note';
```



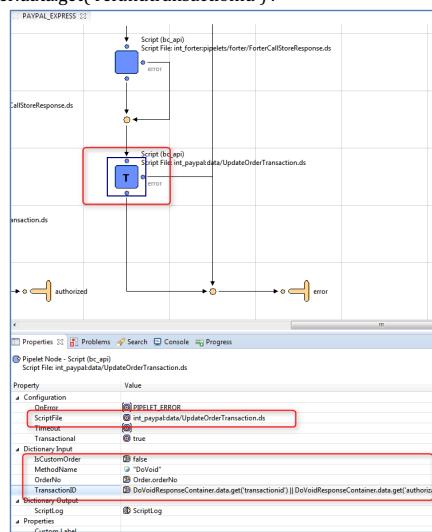
17. Add a Call node. Set 'Pipeline' to 'Paypal-DoVoid'.



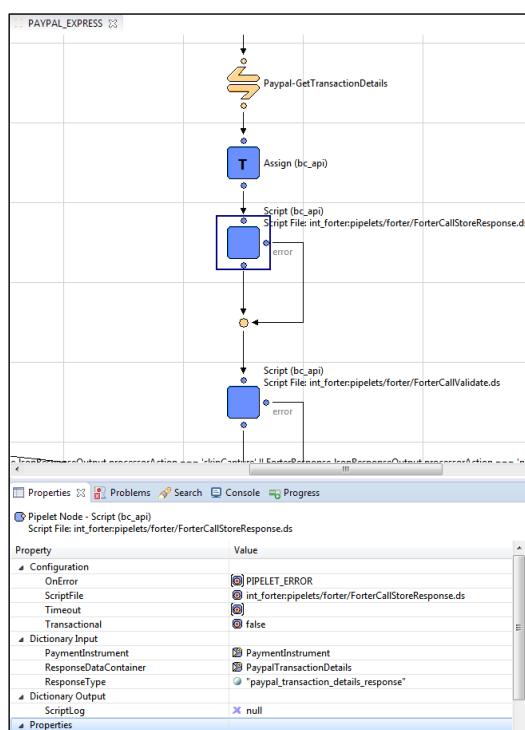
18. Add a Script node. Set 'ScriptFile' to 'int_forter:pipelets/forter/ForterCallStoreResponse.ds';
 'PaymentInstrument' to 'PaymentInstrument'; 'ResponseDataContainer' to 'DoVoidResponseContainer';
 'ResponseType' to "paypal_void_response".



19. Add a Script node. Set
 'ScriptFile' to 'int_paypal:data/UpdateOrderTransaction.ds';
 'IsCustomOrder' to 'false';
 'MethodName' to "DoVoid";
 'OrderNo' to 'Order.orderNo';
 'TransactionID' to 'DoVoidResponseContainer.data.get('transactionid') ||
 DoVoidResponseContainer.data.get('authorizationid') ||
 DoVoidResponseContainer.data.get('refundtransactionid')'.



The Forter cartridge has built-in functionality for saving the PayPal API responses – the transaction details, capture, void, authorization and expresscheckout responses. The data is stored as a JSON string in the custom attribute of the Order Payment Instrument object. Each response is saved to its dedicated custom attribute. The recommended type for these custom attributes is Text since in some cases the response is over 4K characters. In order to save the PayPal response a script node must be added in the place where the required data exists. For example PayPal transaction details may be saved right after a call to PayPal-GetTransacrionDetails and passed into the int_forter:pipelets/forter/ForterCallStoreResponse.ds script. In the script Dictionary Input the “paypal_transaction_details_response” is actually the type of the response custom attribute, “PaypalTransactionDetails” is an object which contains information about current transaction and “PaymentInstrument” is current payment instrument.



In order to handle the customized error message configured in Forter business manager extension, for case if PayPal payment processor called via hooks (if the main site built on controllers), the int_paypal/cartridge/scripts/payment/processor/PAYPAL_EXPRESS.js must check if any error exists in pdict, for example via the if statement **if(!empty(pdict.ForterResponse.PlaceOrderError)){}**:

```

1 PAYPAL_EXPRESS.js :: 
2
3 'use strict';
4
5 /* API Includes */
6 var Pipeline = require('dw/system/Pipeline');
7 var OrderMgr = require('dw/order/OrderMgr');
8 var URLUtils = require('dw/web/URLUtils');
9
10 function Handle(args) {
11     var pdict = Pipeline.execute('PAYPAL_EXPRESS-Handle', {
12         Basket: args.Basket,
13         ContinueURL: URLUtils.https('Paypal-ContinueExpressCheckout')
14     });
15     if(pdict.isSuccess) {
16         return {success: true};
17     } else {
18         return {error: true};
19     }
20 }
21 function Authorize(args) {
22     var pdict = Pipeline.execute('PAYPAL_EXPRESS-Authorize', {
23         Order: OrderMgr.getOrder(args.OrderNo),
24         PaymentInstrument: args.PaymentInstrument
25     });
26     if(pdict.isAuthorized) {
27         if (!empty(pdict.ForterResponse.PlaceOrderError)) {
28             return {error: true, fortterErrorCode: pdict.ForterResponse.PlaceOrderError};
29         } else if (
30             (!empty(pdict.ForterResponse.JsonResponseOutput.actionEnum) && pdict.ForterResponse.JsonResponseOutput.actionEnum == 'DECLINED')
31             ||
32             (!empty(pdict.ForterResponse.JsonResponseOutput.processorAction) && pdict.ForterResponse.JsonResponseOutput.processorAction != 'skipCapture')
33         ) {
34             return {error: true};
35         } else {
36             return {authorized: true};
37         }
38     } else {
39         return {error: true};
40     }
41 /*
42 * Module exports
43 */
44 /*
45 * Local methods
46 */
47 exports.Handle = Handle;
48 exports.Authorize = Authorize;
49

```

In order to handle the customized error message configured in Forter business manager extension the COPlaceOrder.js must be adjusted to check if any error exists in the authorizationResult:

- inside the handlePayments(order) function


```
if (authorizationResult.not_supported || authorizationResult.error) {
    if (!empty(authorizationResult.forterErrorCode)) {
        return {
            error : true,
            forterErrorCode : authorizationResult.forterErrorCode
        };
    } else {
        return {error : true};
    }
}
```

```
COPlaceOrder.js 59
59     if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).getPaymentProcessor() === null) {
60
61         Transaction.wrap(handlePaymentTransaction);
62
63     } else {
64
65         var authorizationResult = PaymentProcessor.authorize(order, paymentInstrument);
66
67         if (authorizationResult.not_supported || authorizationResult.error) {
68             if (!empty(authorizationResult.forterErrorCode)) {
69                 return {
70                     error : true,
71                     forterErrorCode : authorizationResult.forterErrorCode
72                 };
73             } else {
74                 return {error : true};
75             }
76         }
77     }
78 }
79
80     return {};
81 }
82 }
```

- inside the start() function


```
return {
    error: true,
    PlaceOrderError: new Status(Status.ERROR, handlePaymentsResult.forterErrorCode
? handlePaymentsResult.forterErrorCode.code : 'confirm.error.technical')
};
```

```
COPlaceOrder.js 152
152     if (order) {
153         // TODO - need to pass BasketStatus to Cart-Show ?
154         app.getController('Cart').Show();
155
156         return {};
157     }
158     var handlePaymentsResult = handlePayments(order);
159
160     if (handlePaymentsResult.error) {
161         return Transaction.wrap(function () {
162             OrderMgr.failOrder(order);
163             return {
164                 error: true,
165                 PlaceOrderError: new Status(Status.ERROR, handlePaymentsResult.forterErrorCode ? handlePaymentsResult.forterErrorCode.code : 'confirm.error.technical')
166             });
167         });
168     } else if (handlePaymentsResult.missingPaymentInfo) {
169         return Transaction.wrap(function () {
```

Adjusting the Forter Cartridge to include your processor response

The ForterOrder.ds file must be edited to use the response from your payment processor. This script is used to generate the request object for Forter validation. In the example below, we store the response from authorize.net on the payment instrument itself in a custom attribute to be used in this script. If your implementation does not store the response on the payment instrument, you can pass the response object from your payment processor to the validateOrder function (ForterValidate.js controller) as input, which will be sent as a parameter to the ForterOrder.ds file to generate the request object. The script has commented code as an example to see which values need to be sent (optional/required).

```
2
3 /**
4 * ForterOrder class is the DTO object for request.
5 *
6 * To include this script use:
7 * var ForterOrder = require("int_forter/cartridge/scripts/lib/forter/dto/ForterOrder.ds");
8 */
9 function ForterOrder(args) {
10    var order      = args.Order,
11    isRetryJob   = args.IsRetryJob,
12    site        = dw.system.Site.getCurrent(),
13    paymentInstruments = order.getPaymentInstruments(),
14    payment      = null,
15    authResponse = null,
16    shipment     = null;
17
18
19 for each (var paymentInstrument in paymentInstruments) {
20    if (paymentInstrument.paymentMethod == 'CREDIT_CARD') {
21        payment      = paymentInstrument;
22        authResponse = new XML(paymentInstrument.custom.authorize_net_authorization);
23    } else if (paymentInstrument.paymentMethod == 'PayPal' || paymentInstrument.paymentMethod == 'BML') {
24        payment      = paymentInstrument;
25        authResponse = paymentInstrument.custom.paypal_transaction_details_response;
26    }
27 }
```

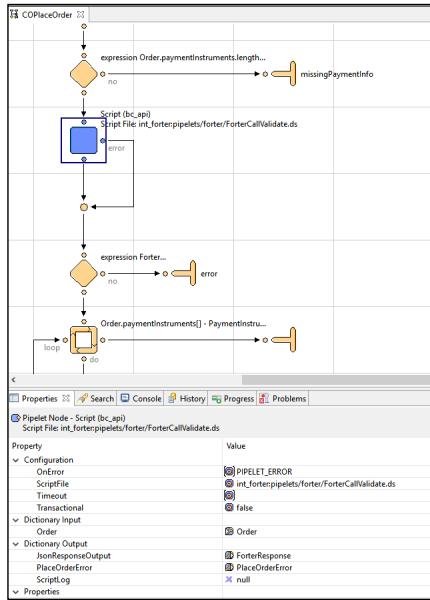
The verificationResults and paymentGatewayData object in the ForterCreditCard function must be adjusted according to the payment gateway used

Pre-Authorization Flow

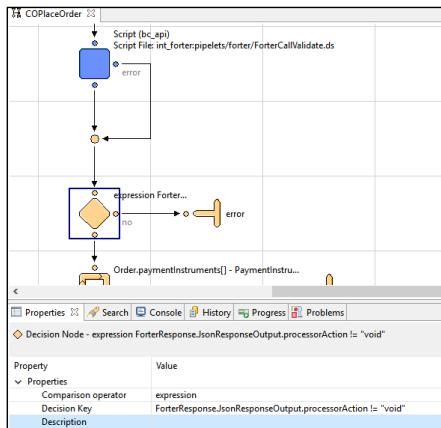
This is not part of the default integration with Forter. This section should be implemented only on specific use cases. Please consult your account manager/integration engineer in Forter before implementing.

For Pipeline-based websites update pipeline:

1. COPlaceOrder-HandlePayments. Add script int_forter:pipelets/forter/ForterCallValidate.ds with parameters shown below. Input: Order, Output: ForterResponse and PlaceOrderError.



2. Add a Decision node with expression: ForterResponse.JsonResponseOutput.processorAction != "void"



For Controllers-based websites update controller:

- COPlaceOrder.js (in the handlepayments (order) function):

```
var argOrderValidate = {
    Order: order,
    orderValidateAttemptInput: 1,
    authorizationStep: "PRE_AUTHORIZATION"
},  
forterController = require('int_forter/cartridge/controllers/ForterValidate'),
```

```

forterDecision = forterController.ValidateOrder(argOrderValidate);
// in case if no response from Forter, try to call one more time
if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
    var argOrderValidate = {
        Order: order,
        originalOrderNo : orderNumber,
        orderValidateAttemptInput: 2,
        authorizationStep: "PRE_AUTHORIZATION"
    },
    forterController = require('int_forter/cartridge/controllers/ForterValidate'),
    forterDecision = forterController.ValidateOrder(argOrderValidate);
}

// IMPORTANT: The forterDecision variable holds the reasonCode from the authorization call,
// which can be used to customize any type of response or flow.

if (forterDecision.JsonResponseOutput.processorAction == 'void') {
    if (!empty(forterDecision.PlaceOrderError)) {
        return {error : true, forterErrorCode : forterDecision.PlaceOrderError.code};
    } else {
        return {error : true};
    }
}

```

```

Forter (Workspace) - COPlaceOrder.js

155 // these lines must be uncommented in case if you want to activate the pre-authorization flow or being included in a top-level cartridge
156 var orderNumber = order.getCurrentOrderNo();
157 var argOrderValidate = {
158     Order : order,
159     originalOrderNo : orderNumber,
160     orderValidateAttemptInput : 1,
161     authorizationStep : "PRE_AUTHORIZATION"
162 };
163 var forterCall = require('/cartridge/controllers/ForterValidate');
164 var forterDecision = forterCall.ValidateOrder(argOrderValidate);
165
166 in case if no response from Forter, try to call one more time
167 if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
168     argOrderValidate = {
169         Order : order,
170         originalOrderNo : orderNumber,
171         orderValidateAttemptInput : 2,
172         authorizationStep : "PRE_AUTHORIZATION"
173     };
174     forterCall = require('/cartridge/controllers/ForterValidate');
175     forterDecision = forterCall.ValidateOrder(argOrderValidate);
176 }
177
178 // IMPORTANT: The forterDecision variable holds the reasonCode from the authorization call,
179 // which can be used to customize any type of response or flow.
180
181 if (forterDecision.JsonResponseOutput.processorAction == 'void') {
182     Transaction.wrap(function () { OrderMgr.failOrder(order, true); });
183
184     if (!empty(forterDecision.PlaceOrderError)) {
185         return {
186             error : true,
187             errorMessage: forterDecision.PlaceOrderError.code
188         };
189     } else {
190         return {
191             error : true,
192             errorMessage: Resource.msg('error.technical', 'checkout', null)
193         };
194     }
195 }

```

2. COPlaceOrder.js (in the start() function):

```

if (!empty(handlePaymentsResult.forterErrorCode)) {
    return {
        error: true,
        PlaceOrderError: new Status(Status.ERROR, handlePaymentsResult.forterErrorCode)
}

```

```

    };
} else {
    return {
        error: true,
        PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
    };
}

COPlaceOrder.js 33
175
176     app.getController('Cart').Show();
177
178     return {};
179 }
180 var handlePaymentsResult = handlePayments(order);
181
182 if (handlePaymentsResult.error) {
183     return Transaction.wrap(function () {
184         OrderMgr.failOrder(order);
185
186         if (!empty(handlePaymentsResult.forterErrorCode)) {
187             return {
188                 error: true,
189                 PlaceOrderError: new Status(Status.ERROR, handlePaymentsResult.forterErrorCode)
190             };
191         } else {
192             return {
193                 error: true,
194                 PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
195             };
196         }
197     });
198
199 } else if (handlePaymentsResult.missingPaymentInfo) {
200     return Transaction.wrap(function () {
201         OrderMgr.failOrder(order);
202
203         return {
204             error: true,
205             PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
206         };
207     });
208
209     var orderPlacementStatus = Order.submit(order);
210     if (!orderPlacementStatus.error) {
211         clearForms();
212     }
213
214     return orderPlacementStatus;
215

```

For SFRA-based websites, the CheckoutServices.js has been updated. Please note that the Forter cartridge replaces the 'PlaceOrder' endpoint, so these lines must be uncommented in case if you want to activate the pre-authorization flow or being included in a top-level cartridge:

1. CheckoutServices.js (replaces the 'PlaceOrder' with next code includes) in order send order information to the Forter endpoint before authorization call and after to handle the customized error massage configured in Forter business manager extension:

```

var orderNumber = order.getCurrentOrderNo();
var argOrderValidate = {
    orderNumber : orderNumber,
    orderValidateAttemptInput: 1,
    authorizationStep: "PRE_AUTHORIZATION"
},
forterCall = require('/cartridge/scripts/pipelinelets/forter/forterValidate'),
forterDecision = forterCall.validateOrder(argOrderValidate);

// in case if no response from Forter, try to call one more time
if(forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {

```

```

var argOrderValidate = {
    orderNumber      : orderNumber,
    orderValidateAttemptInput : 2,
    authorizationStep: "PRE_AUTHORIZATION"
},
forterCall     = require('*/*cartridge/scripts/pipelinelets/forter/forterValidate'),
forterDecision = forterCall.validateOrder(argOrderValidate);
}

// IMPORTANT: The forterDecision variable holds the reasonCode from the authorization call,
// which can be used to customize any type of response or flow.

if (forterDecision.JsonResponseOutput.processorAction == 'void') {
    Transaction.wrap(function () { OrderMgr.failOrder(order, true); });
    if (!empty(forterDecision.PlaceOrderError)) {
        res.json({
            error : true,
            errorMessage : forterDecision.PlaceOrderError.code
        });
    } else {
        res.json({
            error : true,
            errorMessage : Resource.msg('error.technical', 'checkout', null)
        });
    }
    this.emit('route:Complete', req, res);
    return;
}

```

```

Forter (Workspace) - CheckoutServices.js

31 // these lines must be uncommented in case if you want to activate the pre-authorization flow or being included in a top-level cartridge
32 var orderNumber = order.getCurrentOrder();
33 var argOrderValidate = {
34     orderNumber      : orderNumber,
35     orderValidateAttemptInput : 1,
36     authorizationStep : "PRE_AUTHORIZATION"
37 };
38 var forterCall     = require('*/*cartridge/scripts/pipelinelets/forter/forterValidate');
39 var forterDecision = forterCall.validateOrder(argOrderValidate);
40
41 in case if no response from Forter, try to call one more time
42 if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
43     argOrderValidate = {
44         orderNumber      : orderNumber,
45         orderValidateAttemptInput : 2,
46         authorizationStep : "PRE_AUTHORIZATION"
47     };
48     forterCall     = require('*/*cartridge/scripts/pipelinelets/forter/forterValidate');
49     forterDecision = forterCall.validateOrder(argOrderValidate);
50 }
51
52 // IMPORTANT: The forterDecision variable holds the reasonCode from the authorization call,
53 // which can be used to customize any type of response or flow.
54
55 if (forterDecision.JsonResponseOutput.processorAction == 'void') {
56     Transaction.wrap(function () { OrderMgr.failOrder(order, true); });
57
58     if (!empty(forterDecision.PlaceOrderError)) {
59         res.json({
60             error : true,
61             errorMessage: forterDecision.PlaceOrderError.code
62         });
63     } else {
64         res.json({
65             error : true,
66             errorMessage: Resource.msg('error.technical', 'checkout', null)
67         });
68     }
69     this.emit('route:Complete', req, res);
70     return;
71 }
72

```

You can use the code below to update the order pre-authorized with a new status, generally, this will be placed on the script responsible to handle the credit card process, you can find this code in the AUTHORIZE_NET script as an example of implementation.

The code applies for all implementation, SFRA and SG.

```
var argOrderUpdate = {
    orderNumber: orderNumber,
    updateAttempt: 1
},
forterCall = require('/cartridge/scripts/pipelinelets/forter/forterValidate'),
forterDecision = forterCall.postAuthOrderStatusUpdate(argOrderUpdate, "PROCESSING");

if (forterDecision.result === false && forterDecision.updateAttempt == 2) {
    forterDecision.updateAttempt = 2;
    forterCall.postAuthOrderStatusUpdate(argOrderUpdate, "PROCESSING");
}
```

By default the status is CANCELED_BY_MERCHANT or PROCESSING.

```
// these lines must be uncommented in case if you want to activate the pre-authorization flow + post-auth order status update - uncomment this and
var argOrderUpdate = {
    orderNumber: orderNumber,
    updateAttempt: 1
},
forterCall = require('/cartridge/scripts/pipelinelets/forter/forterValidate'),
forterDecision = forterCall.postAuthOrderStatusUpdate(argOrderUpdate, "PROCESSING");

if (forterDecision.result === false && forterDecision.updateAttempt == 2) { You, a few seconds ago * Uncommitted changes
    forterDecision.updateAttempt = 2;
    forterCall.postAuthOrderStatusUpdate(argOrderUpdate, "PROCESSING");
}
```

3.3.5 *Footer*

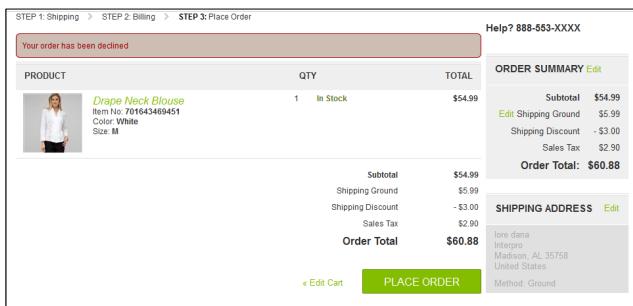
Please note that Forter's custom JavaScript snippet, which captures vital behavioral data, has been added to the SiteGenesis footer. SiteGenesis footer.isml has been modified in order to add the mentioned functionality.

```
<!-- ===== -->
<!-- == -->
<!-- == FORTER INTEGRATION == -->
<!-- == -->
<!-- == -->
<!-- ===== -->
<!--
    Renders the Forter.js snippet
-->
<isinclude template="custom/fortersnippetjs"/>
```

3.4 Testing

In order to see if the cartridge is installed and configured correctly, you need to go to the Storefront and place some test orders, both as a registered customer and as a guest customer. Then, go to check the order status in the Forter dedicated page Merchant Tools > Forter > Orders.

A registered customer will log in, add an item to the cart and proceed to checkout, while the guest customer may add the item in the cart and proceed directly to checkout page. After the shipping, billing and payment information has been provided the customer will be able to place the order. If the payment information is not valid, the payment gateway will fail the order and the Salesforce Commerce Cloud standard failed message is displayed. If the Forter configuration is to cancel the order immediately and show a decline page when Forter declines the order than a customized error message will be displayed.



Otherwise the order will be successfully processed and the thank you page will be displayed. The merchant should also check the Forter decision and the order status in the site preferences section.

4. Operations, Maintenance

4.1 Data Storage

The cartridge stores response information from Forter in the Order system object definition via custom attributes in order to process the orders. It also stores the first 6 digits of the customer credit card number if there are errors in the request to Forter.

4.2 Availability

4.2.1 Forter error / Failover

Every error that is related to Forter is reported to the Forter errors API endpoint. The request payload contains error description, order ID and stack trace payload in JSON format. In case of service unavailability second attempt is performed. In case of second attempt fail – an order will contain 'Error' in the Forter decision attribute.

General Attributes Payment Notes History

Attributes for Order '00145401'

On this page you can edit the attributes of the order. Fields with a red asterisk (*) are mandatory. Click **Apply** to save changes. Click **Reset** to revert your changes.

Forter group

Forter decision*: ERROR (Error)

Forter Order Link: No data is available

Forter Order Status: No data is available

Mapping to Forter statuses according the order status: No data is available

Forter User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.100 Safari/537.36

Forter Token Cookie: ffcbe98ad9e4da7a9c44680be0b54db_1581593197649_UDF43_9ok

Retry number: 2

Apply Reset

4.3 Support

Supporting documentation and data will be provided:

- **Archived cartridge**
- **Configuration / installation files**

Please contact your Forter sales representative at info@forter.com for more details about the integration process.

5. User Guide

5.1 Roles, Responsibilities

Salesforce Commerce Cloud merchants who have purchased the cartridge and have access rights to the Salesforce Commerce Cloud Business Manager and to configure the cartridge will benefit from the services that the Forter cartridge provides.

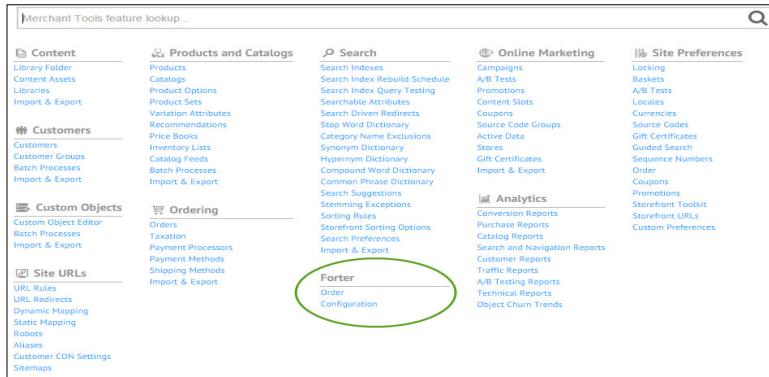
5.2 Business Manager

The Forter cartridge adds a Business Manager extension. It is used to add and test Forter's configuration, and to track Forter orders.

The following screenshots show the changes added.

5.2.1 Menu extension

A Forter site specific extension is added to the Business Manager. It adds two new functionalities: *Forter configuration* and *Forter orders*.



5.2.2 Forter section

This new section has been added to the Site Preferences and here the merchant can set up the Forter configurations and/or check the status of orders.

The screenshot shows a box titled 'Forter' with a sub-section 'Order'. It contains the text 'Manage the orders.' and a link 'Configuration' which is described as 'Configure Forter: Frictionless Fraud Prevention'.

5.2.3 Forter configuration

In the Forter dedicated section a specific SiteID and Secret Key should be provided. When a call to Forter is made, if the combination is valid then a second page for advanced Forter Configuration is shown.

The screenshot shows a 'Forter > Configuration' page with a 'Change API credentials' section. It includes fields for 'Site ID' and 'Secret key', and a 'Link' button.

The screenshot shows a 'Forter > Configuration' page with several sections: 'Configuration' (with a note 'You are one step away from activating Forter'), 'Change API credentials' (with fields for Site ID and Secret key), 'Configure Forter' (with 'Enabled' set to 'Yes'), 'Communication' (with a checkbox for 'Show customized message when Forter has returned a decline decision and the order was cancelled immediately'), 'Payment Settings' (with checkboxes for 'Auto-invoice when transaction is approved' and 'Cancel and void order when transaction is declined'), 'Order Status Updates' (with a dropdown for 'Please choose the number of weeks after order placement when order status updates should be sent' set to '4'), and a 'Save' button.

Forter configurations are saved in Salesforce Commerce Cloud site preferences. They can be manipulated through the Forter -> Configuration extension screen. Please do not manipulate them directly through site preferences because this way no Forter call for verification is made.

5.2.4 Forter Orders

A Forter orders page is added. Thus the merchant can easily search for orders based on the Forter decision. The Forter Decision will influence the Order Status.

These are the possible Forter Decision values:

- **Approved** – Forter approves the transaction; **Merchant should capture the transaction funds, communicate the successful checkout to the customer and produce an invoice.**
- **Declined** – The order is suspected as fraudulent and Forter declines the transaction; **Merchant should cancel the transaction and communicate with the customer**
- **Not reviewed** – Forter does not have sufficient information in order to approve or decline the transaction; **Forter does not review the transaction, according to policy. Merchant should act according to the policy that was in place before integration with Forter** à merchant verifies data: approve/decline the order.
- **Error** – The information is not sent to Forter (**due to issues with the cartridge**). As noted above, *the merchant should configure the desired flow for this use case*. In the sample flow above orders are captured. *This should not happen. In case it does the merchant should reach out Forter customer support to investigate the issue.*
- **Not sent** – Orders not sent to Forter. This is the default status for all previous cartridge installation orders that are in the system.

Order										
Search for your orders										
Order Number	Order Date	Created By	Registration Status	Customer	Email	Forter Decision	Forter Order Link	Total	Status	
00003803	03/30/2016 9:43 am	Customer	Unregistered	Guy Zucker	approve@forter.com	Approved	View Order	\$1,160.22	Completed	
00003802	03/30/2016 9:30 am	Customer	Unregistered	Guy Zucker	approve@forter.com	Approved	View Order	\$148.67	Open	
00003703	03/30/2016 8:58 am	Customer	Unregistered	Guy Zucker	approve@forter.com	Approved	View Order	\$218.37	Open	
00003602	03/29/2016 3:16 pm	Customer	Unregistered	Guy Zucker	approve@forter.com	Approved	View Order	\$207.88	Completed	
00003403	03/29/2016 7:15 am	Customer	Unregistered	Guy Zucker	approve@forter.com	Approved	View Order	\$661.47	Open	

In addition, you may want to add the Forter Decision, the Forter Order Link and the Forter Reason Code to the default order search view.

Order Search										
<input type="checkbox"/> Order Number <input type="button" value="Find"/> Simple Advanced By Number										
Number	Order Date	Site	Created By	Registration Status	Customer	Email	Total	Status	Forter decision	Forter Order Link
00007403	4/27/23 2:20:30 pm Etc/UTC	RefArch	Customer	Unregistered	as as	declined@forter.com	\$69.29	Failed	Declined	https://portal.fortec.com/dashboard/00007403
00007402	4/27/23 2:14:14 pm Etc/UTC	RefArch	Customer	Unregistered	as as	alba.com	\$44.09	Open	Approved	https://portal.fortec.com/dashboard/00007402
00007401	4/27/23 1:59:40 pm Etc/UTC	RefArch	Customer	Unregistered	aid aid	alba.com	\$11.29	Open	Approved	https://portal.fortec.com/dashboard/00007401
00007205	3/1/23 4:05:24 pm Etc/UTC	RefArch	Customer	Registered	test test	test@gmail.com	\$57.74	Failed	Declined	https://portal.fortec.com/dashboard/00007205
00007204	3/1/23 4:04:42 pm Etc/UTC	RefArch	Customer	Registered	test test	test@gmail.com	\$57.74	Failed	Declined	https://portal.fortec.com/dashboard/00007204

Steps to add custom fields described below:

1. Go to Administration > Global Preferences > Order Search.

Administration > [Global Preferences](#) > Order Search Preferences

Order Search Preferences

Order List Columns

Define up to five additional order attributes that will be shown in order search list views. Note that custom attributes must be prefixed with 'custom.'

Custom Order Column 1:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 2:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 3:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 4:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 5:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 6:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 7:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 8:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 9:	<input type="text"/>	<input type="button" value="..."/>
Custom Order Column 10:	<input type="text"/>	<input type="button" value="..."/>

[<< Back](#)

2. Click on '...' button, search for the 'forterDecision' attribute and click on it.

Select Object Type Attribute

Select the attributes you want from the list below by clicking the attribute ID or name. You can close this window without selecting an attribute by clicking Cancel.

Search Attribute Definitions

ID	Attribute Name	Type	Attribute Settings
affiliatePartnerID	Affiliate Partner ID	String	
affiliatePartnerName	Affiliate Partner Name	String	
businessType	Business Type	Enum of Integers	
cancelCode	Cancel Code	Enum of Strings	
cancelDescription	Cancel Description	String	
chargeType	Charge Type	Enum of Integers	
confirmationStatus	Confirmation Status	Enum of Integers	
createdBy	Created By	String	
creationDate	Creation Date	Date/Time	
currencyCode	Currency Code	String	
customerEmail	Customer Email	String	
customerName	Customer Name	String	
customerNo	Customer Number	String	
customerOrderReference	Customer Order Reference	String	
exportAfter	Export After	Date/Time	
exportOrder	Export Order	Enum of Integers	
externalOrderID	External Order ID	String	
externalOrderNumber	External Order Number	String	
externalOrderStatus	External Order Status	String	
externalOrderText	External Order Text	String	
forterDecision	Forter decision	Enum of Strings	

3. Repeat the steps noted above for the 'forterOrderLink' and 'forterReasonCode' attributes.

[Administration](#) > [Global Preferences](#) > Order Search Preferences

Order Search Preferences

Order List Columns

Define up to five additional order attributes that'll be shown in order search list views. Note that

Custom Order Column 1:	custom.forterDecision	...
Custom Order Column 2:	custom.forterOrderLink	...
Custom Order Column 3:	custom.forterReasonCode	...
Custom Order Column 4:		...

5.2.5 Order Update job

The order update job checks for a Salesforce Commerce Cloud order status change, and sends it to the relevant Forter API endpoint via HTTPS. *It is recommended that the job be run every 6 hours.* Please note that in the Custom Site Preferences section you can configure the parameter **Number of weeks** that is used by this job to determine the time range (number of weeks) that the job queries in order to update order status. The default value is 4 weeks.

Administration / Operations / Job Schedules /

Edit Job ForterOrderUpdate 

Run Now

General Schedule and His... Resources Step Configurator Notification Failure Handling

Enabled

Active

Trigger Recurring Interval

From* 4/23/2017 4:17 pm  To 

Run Time

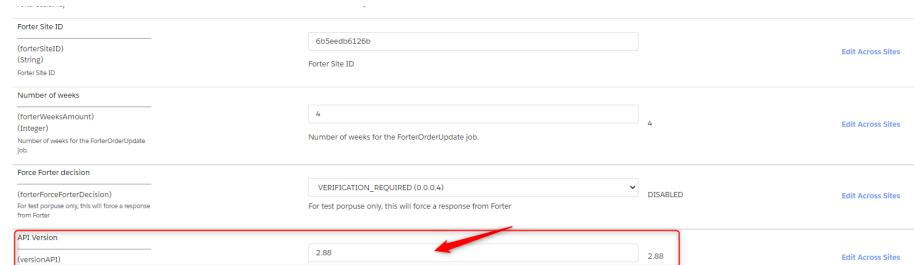
Every

Amount* 5 Interval* Minutes

Run only on these days:
 Monday Tuesday Wednesday Thursday Friday Saturday Sunday

5.2.6 Forter API Version

We have a new functionality for changing the API version. Now, the API version can be changed directly from BM. You need to go to the Merchant Tools → Site Preferences, click on the fortter preference and search for API Version attribute. The default value is 2.88 (the current version for fortter API). You can change from there when a version shows up.



The screenshot shows the Site Preferences form for a Forter Site ID. The 'API Version' field is highlighted with a red arrow pointing to it. The field contains the value '2.88'. Other fields visible include 'Forter Site ID' (value: 605ebedb126b), 'Number of weeks' (value: 4), and 'Force Forter decision' (value: VERIFICATION_REQUIRED (0.0.0.4)).

Forter Site ID	605ebedb126b	Edit Across Sites
(forterSiteId) (String)		
Forter Site ID		
Number of weeks	4	4
(forterWeeksAmount) (Integer)		
Number of weeks for the ForterOrderUpdate job.		
Force Forter decision	VERIFICATION_REQUIRED (0.0.0.4)	DISABLED
(forterForceForterDecisions)		
For test purpose only, this will force a response from Forter		
API Version	2.88	2.88
(versionAPI)		

5.3 Storefront Functionality

5.3.1 JavaScript Snippet

The Forter JavaScript snippet should be injected into the site footer section for all web pages. For this purpose, the following template has been built: int_forter/cartridge/templates/default/custom/fortersnippetjs.isml

```
<script>
  (function () {
    document.addEventListener('ftr:tokenReady', function(evt) {
      var token = evt.detail;
      console.log(token);
      if (token != null) {
        var postInfo = {
          ftrtoken : token
        };
        jQuery.ajax({
          type: "POST",
          url: "${dw.web.URLUtils.url('ForterValidate_UpdateForterInfo').toString()}",
          data: postInfo,
          success: function(data) {}
        });
      }
    });
  })();
</script>
<script type="text/javascript" id="<ispint value="${dw.system.Site.getCurrent().getCustomPreferenceValue('forterSiteID)}">" />>
<(function () {
  var merchantConfig = {
    csp: false
  };
  You now have uncommitted changes
  var siteId = "<ispint value="${dw.system.Site.getCurrent().getCustomPreferenceValue('forterSiteID)}">"/>";
  function t(t,n){for(var e=t.split(""),r=0;r<e.length;++r)e[r]=String.fromCharCode(e[r].charCodeAt(0)+n);return e.join("")}function n(n){return t(n,-5).replace(/\%S%/g,siteId))}>
</script>
```

6. Known Issues

There are currently no known issues.

7. Release History

Version	Date	Changes
16.1.0	7.03.2016	Initial release
17.1.0	11.05.2017	Updated calls for Order validation. Updated calls for Account info update. Updated the cartridge to run on both pipeline and controllers based sites. Include support for Paypal orders.
17.1.3	24.11.2017	Improvement for the Forter Orders grid view.
17.1.4	02.02.2018	Added call for order info sending to Forter in case of failed payment authorization. Added two additional fields for payment processor response code and response text into the VerificationResults object.
18.1.0	04.13.2018	Implementation of the cartridge to work with SFRA SiteGenesis. Deprecate the Forter Void job.
21.1.0	02.14.2021	Implementation of the Pre-authorization flow (optional). Update of deprecated API methods.
23.0.0	08.08.2023	Added new Reason Code attribute to the Authorization response from Forter and updated Documentation