FORTER CARTRIDGE

Version 23.0.0

Table of Contents

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.	1. The customer creates an account and logs in with the newly created account using email *approve@forter.com*. 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 *Approved* and the order status is *New*. • 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.	1. The customer creates an account and logs in with the newly created account using email *decline@forter.com*. 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 declined and based on the Forter configuration a declined message may be displayed*. 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 *Declined* and the order status is *Failed*. • 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.	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. *The call to Forter is not sent, the transaction is not approved and the Salesforce Commerce Cloud standard failed message is displayed*. 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 *Not sent* and the order status is *Failed*. • 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	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. The latter information is used by Forter to detect fraudulent orders.

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	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	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 *Approved* and the order status is *New*. • Note that a similar flow can be done for a declined transaction.

UC – 7	Customer Login
This use case describes the possible forter response and decisions for account login event.	1. A user accesses the Login page and logs in. 2. Information is sent to Forter. 3. Forter's response has the attribute forterDecision 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: 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.	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 forterDecision 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. 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:

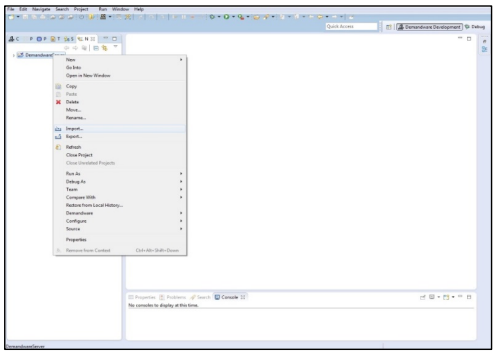
- int_forter
- int_forter_sfra
- bm_forter

3.2 Configuration

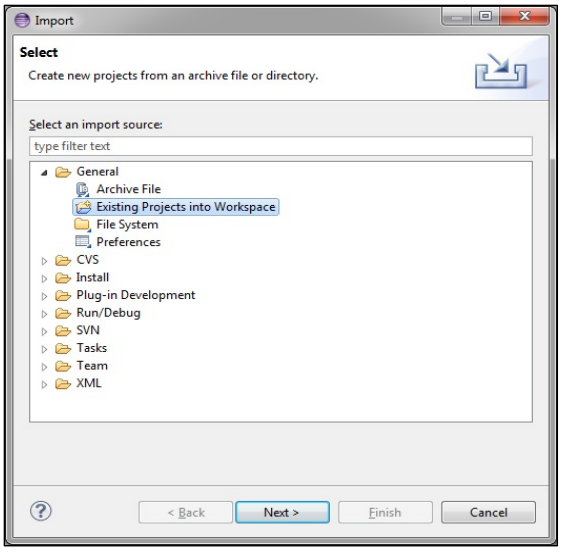
3.2.1 *Setup*

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

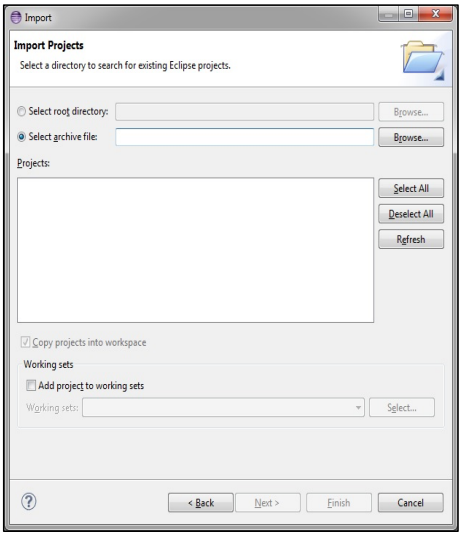
1. Select the connection to the DW server -> Import



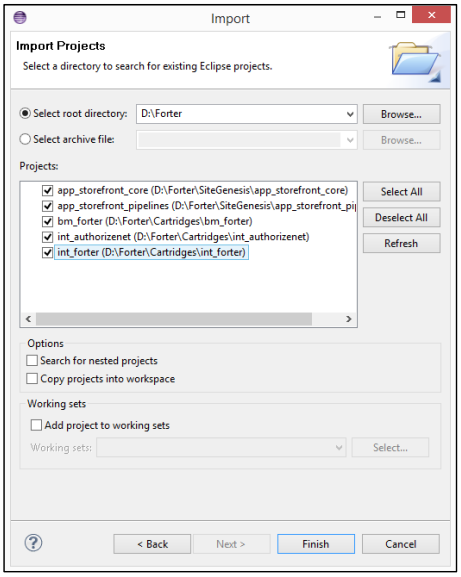
2. Select the “Existing Project into Workspace” option



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*

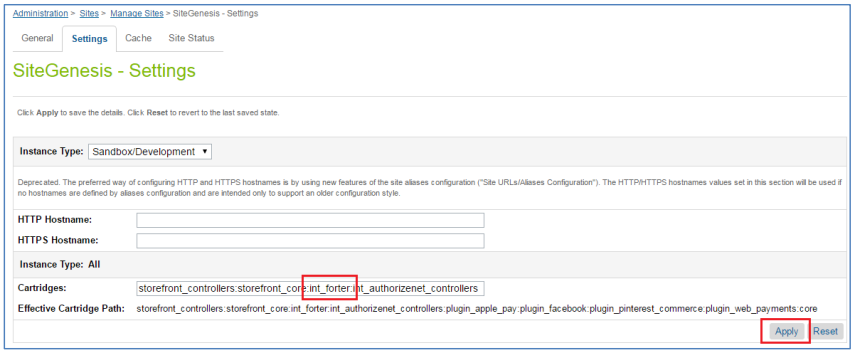
In the Metadata folder you will find a zip file called 'site_template'.

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

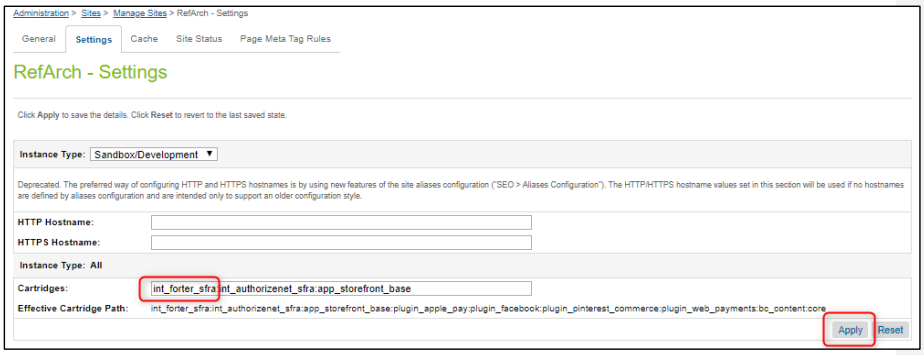
Select the zip you uploaded, click on Import then on the ok button

3.2.3 *Cartridge paths*

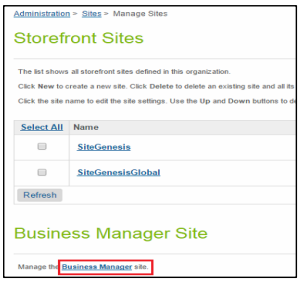
Go to Administration > Sites > Manage Sites and choose your site. Click on the Settings tab, add in the cartridge path int_forter then click Apply.



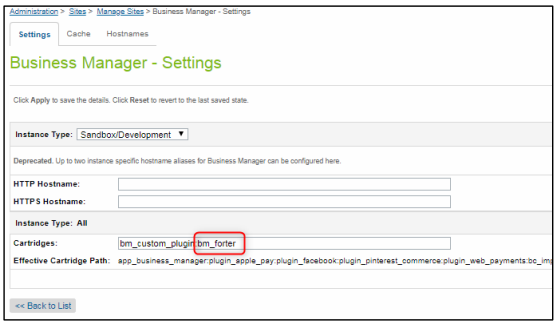
Add in the cartridge path int_forter_sfra in case if site is SFRA based.



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



Add to the cartridge path bm_forter then click Apply.

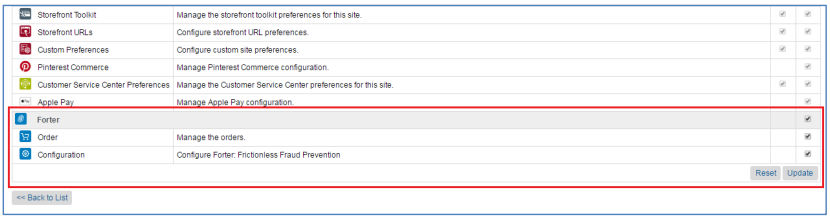


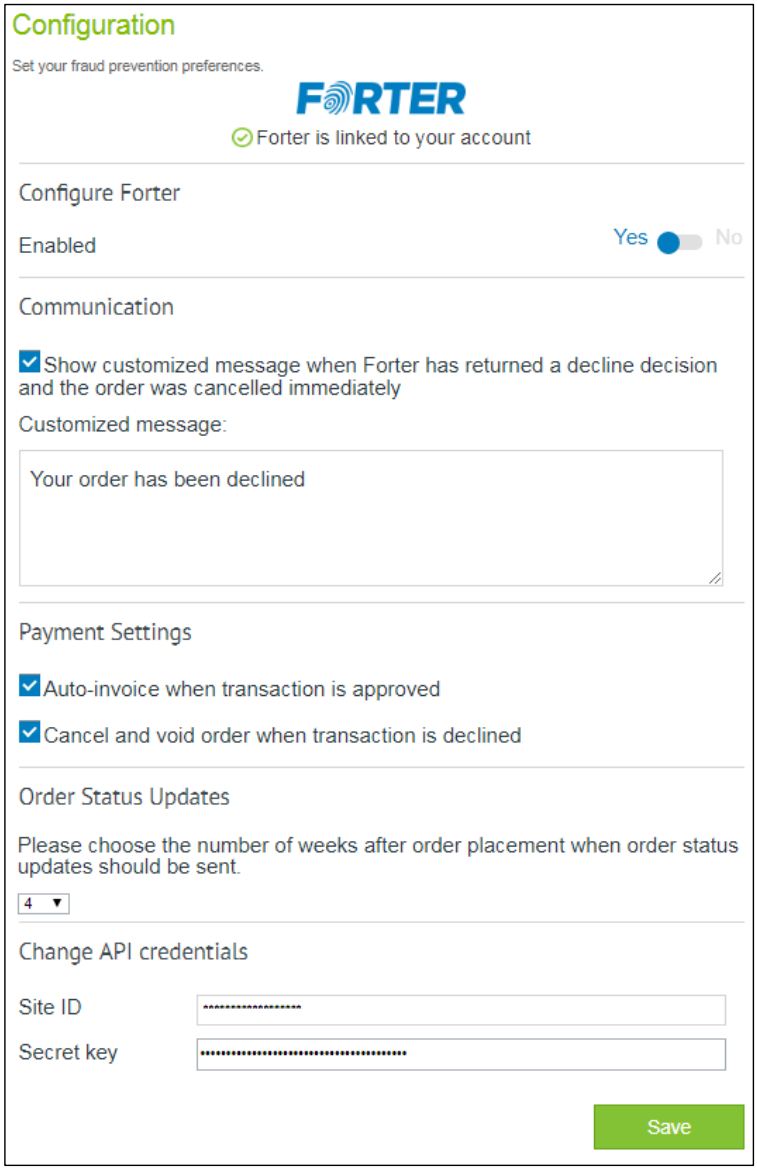
3.2.4 *Forter custom site preferences*

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.




Configuration
Set your fraud prevention preferences.
FORTER
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

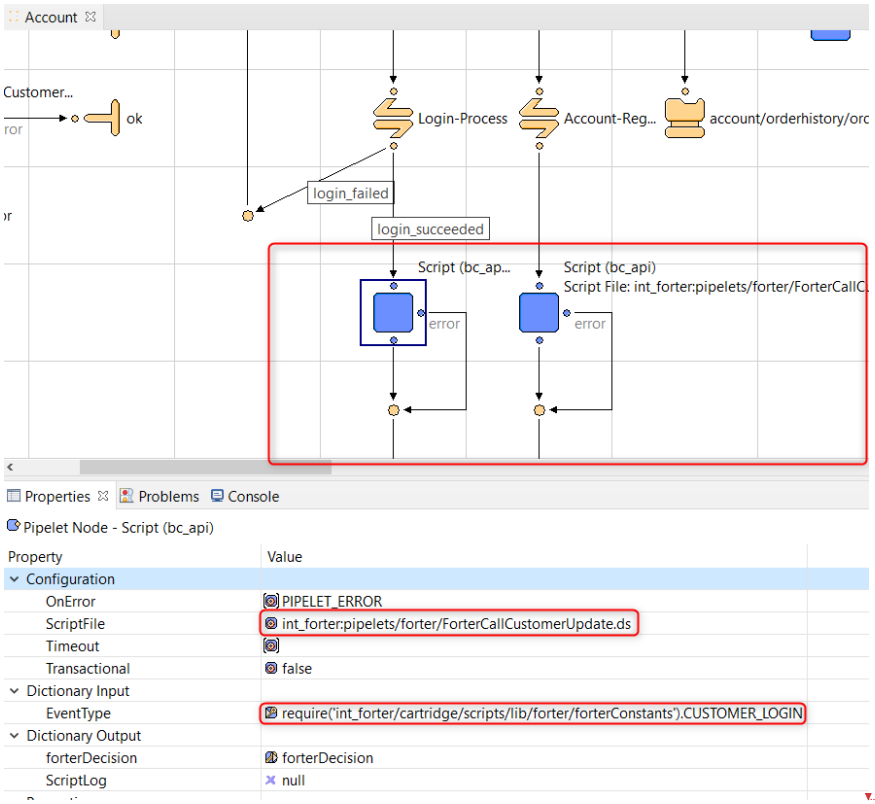
The Forter platform allows for custom attributes to be sent in the request. It is recommended that if you have attributes that are relevant for fraud detection, you should set them in the request objects generated in the cartridge.

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

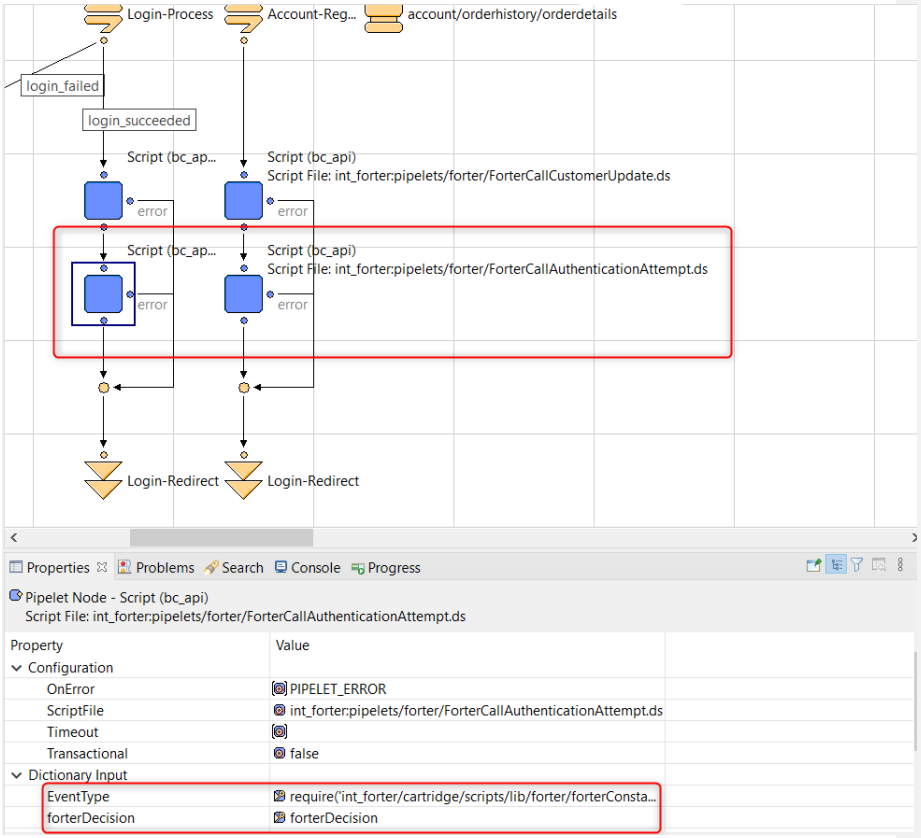


Deleted: ¶

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

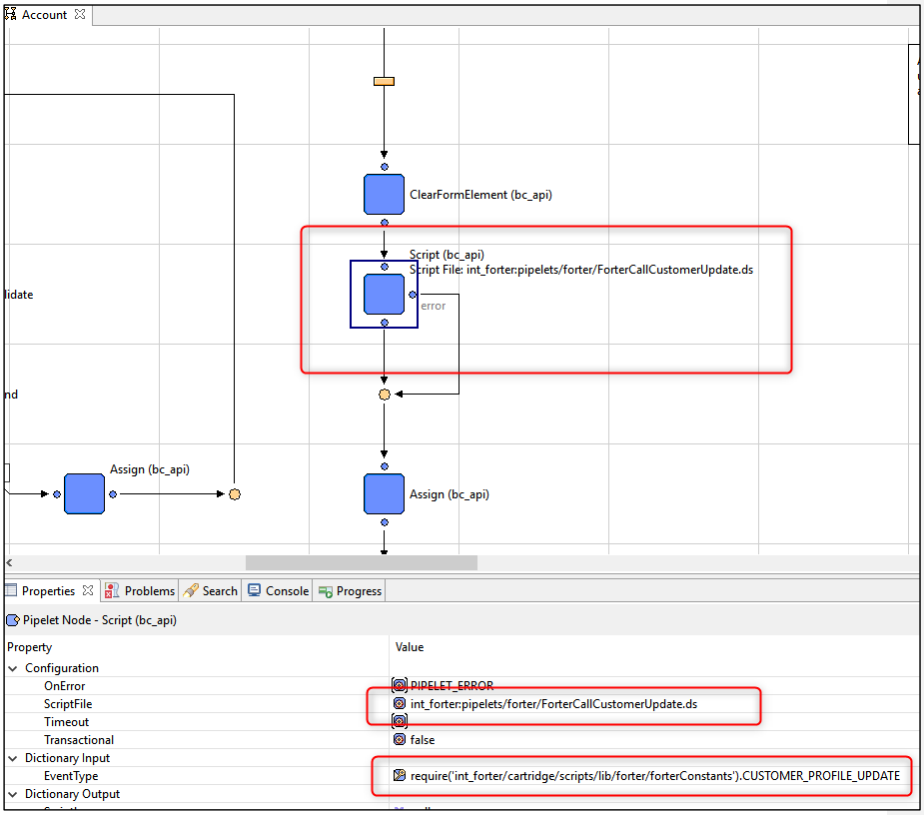
EventType require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_LOGIN

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



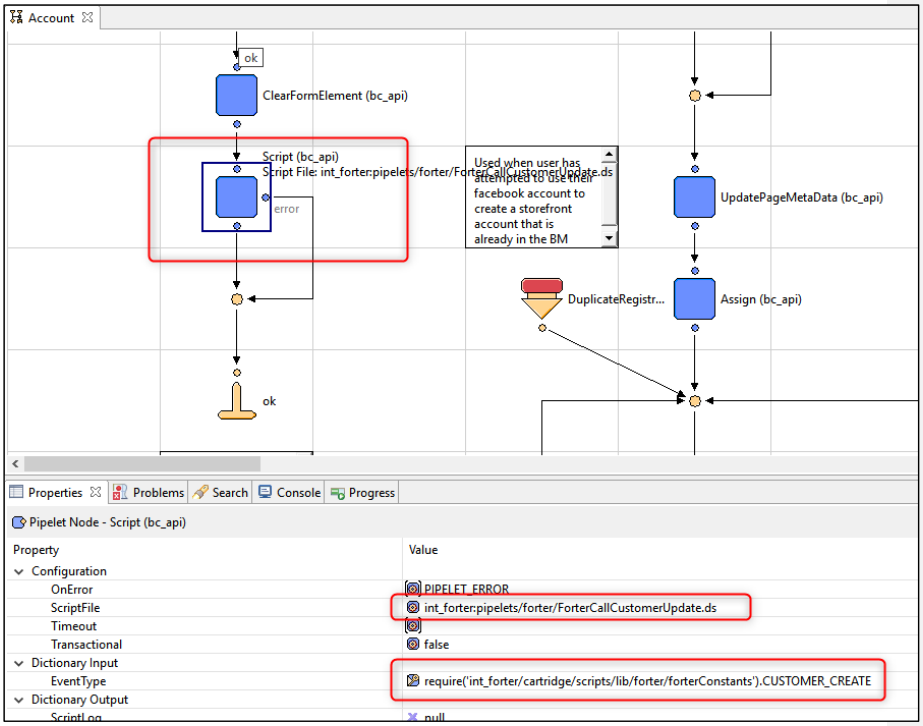
- Account-EditProfile,

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



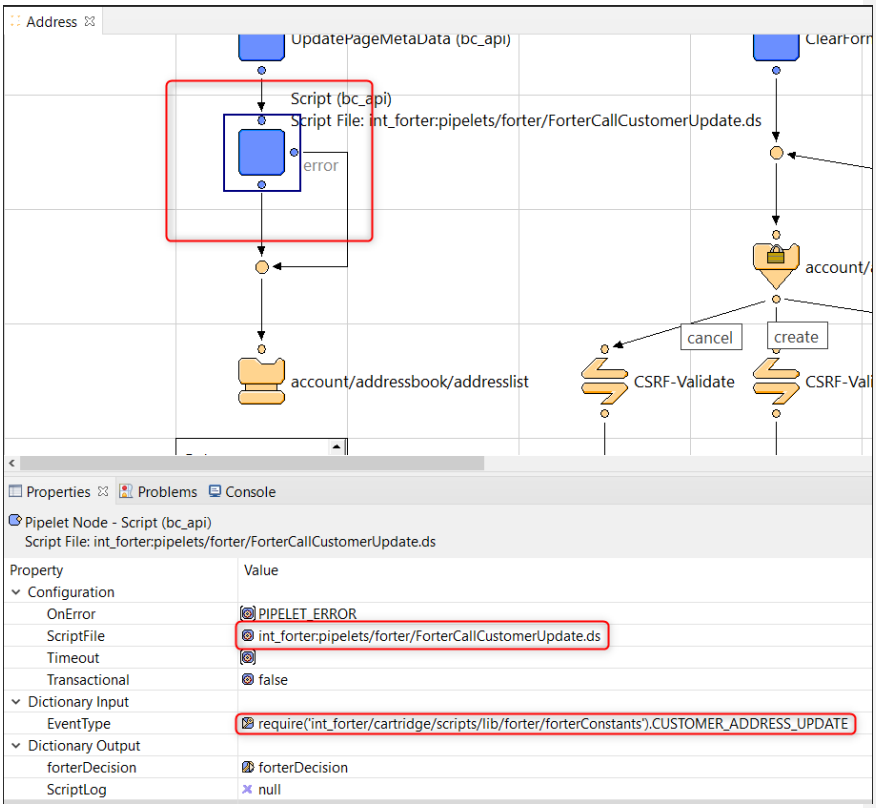
- Account-Register,

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



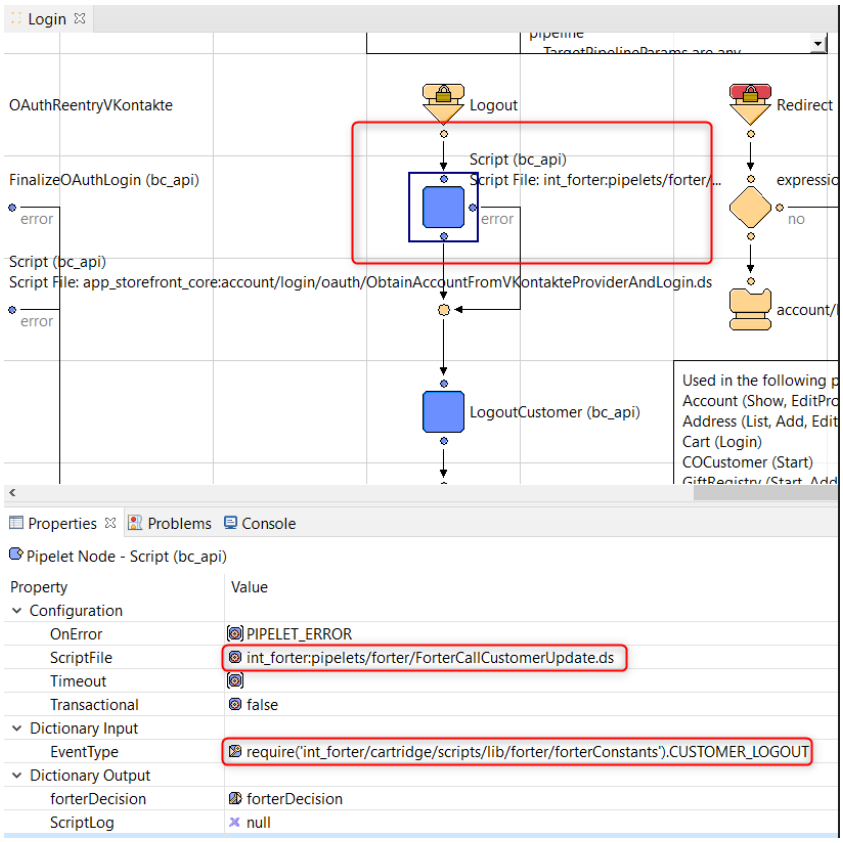
- Address-List, EventType

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

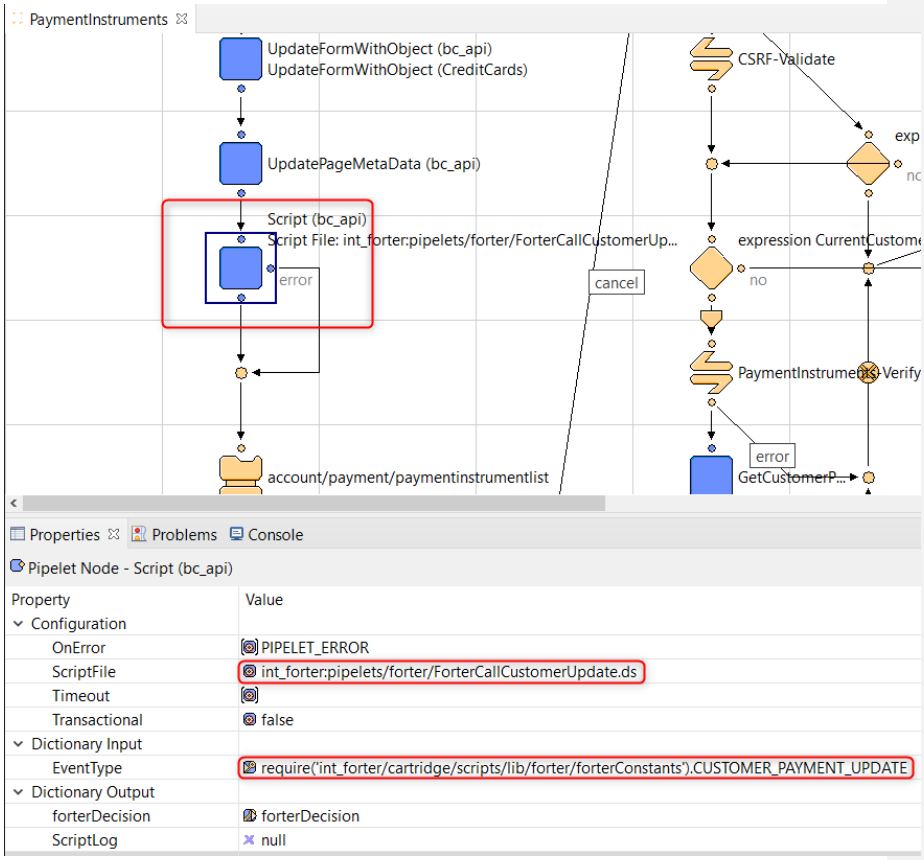


- Login-Logout, EventType

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

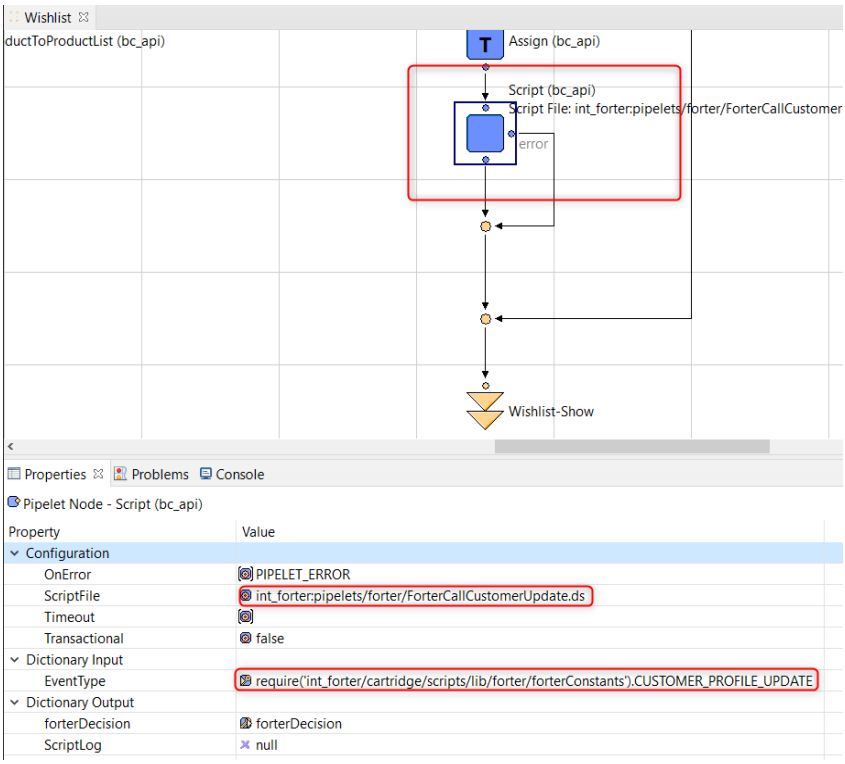


- PaymentInstruments-List, EventType
  require('int_forter/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PAYMENT_UPDATE

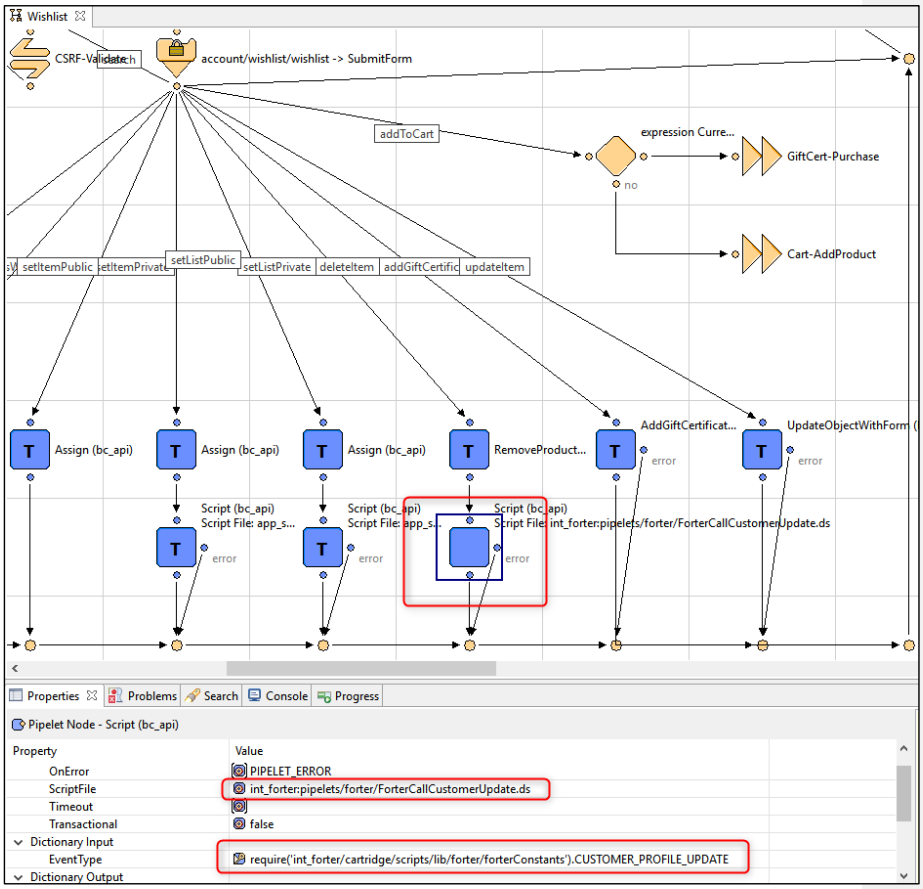


- Wishlist-Add, EventType

  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);

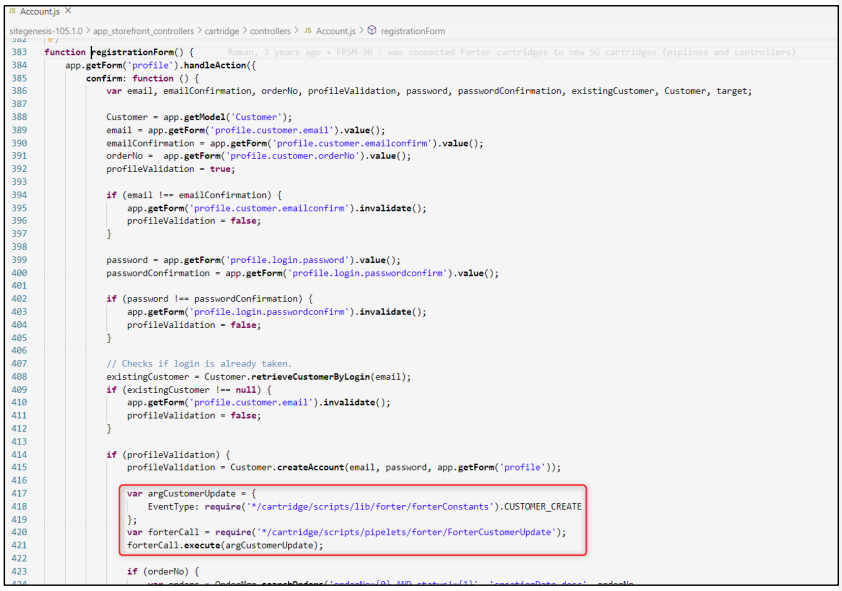
- Account.js (in the registrationForm() function):

  var argCustomerUpdate = {

  EventType: require('*/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_CREATE};

  var forterCall = require('*/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');

  forterCall.execute(argCustomerUpdate);



```
function registrationForm() {
    app.getForm('profile').handleAction({
        confirm: function () {
            var email, emailConfirmation, orderNo, profileValidation, password, passwordConfirmation, existingCustomer, Customer, target;

            Customer = app.getModel('Customer');
            email = app.getForm('profile.customer.email').value();
            emailConfirmation = app.getForm('profile.customer.emailconfirm').value();
            orderNo =  app.getForm('profile.customer.orderNo').value();
            profileValidation = true;

            if (email !== emailConfirmation) {
                app.getForm('profile.customer.emailconfirm').invalidate();
                profileValidation = false;
            }

            password = app.getForm('profile.login.password').value();
            passwordConfirmation = app.getForm('profile.login.passwordconfirm').value();

            if (password !== passwordConfirmation) {
                app.getForm('profile.login.passwordconfirm').invalidate();
                profileValidation = false;
            }

            // Checks if login is already taken.
            existingCustomer = Customer.retrieveCustomerByLogin(email);
            if (existingCustomer !== null) {
                app.getForm('profile.customer.email').invalidate();
                profileValidation = false;
            }

            if (profileValidation) {
                profileValidation = Customer.createAccount(email, password, app.getForm('profile'));

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

                if (orderNo) {
```

- 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);

```
function list() {
    var pageMeta = require('~/cartridge/scripts/meta');

    var content = app.getModel('Content').get('myaccount-addresses');
    if (content) {
        pageMeta.update(content.object);
    }
    var argCustomerUpdate = {
        EventType: require('*/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE
    };
    var forterCall = require('*/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');
    forterCall.execute(argCustomerUpdate);

    app.getView().render('account/addressbook/addresslist');
}
```

- 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/pipelets/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/pipelets/forter/ForterAuthenticationAttempt
tUpdate');
  forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
}
```

```
loginForm.handleAction({
    login: function () {
        // Check to see if the number of attempts has exceeded the session threshold
        if (RateLimiter.isOverThreshold('FailedLoginCounter')) {
            RateLimiter.showCaptcha();
        }

        var success = Customer.login(loginForm.getValue('username'), loginForm.getValue('password'), loginForm.getValue('rememberme'));

        if (!success) {
            loginForm.get('loginsucceeded').invalidate();
            app.getView('Login').render();
            return;
        } else {
            loginForm.clear();
        }

        RateLimiter.hideCaptcha();

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

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

            // example of object populated with MFA results.
            argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
                verificationOutcome: 'SUCCESS',
                correlationId: 'HG1751234567DE',
                emailVerification: {
                    email: customer.profile.email,
                    emailRole: 'ACCOUNT',
                    sent: true,
                    verified: true,
                }
            };

            var forterAuthAttempCall = require('*/cartridge/scripts/pipelets/forter/ForterAuthenticationAttemptUpdate');
            forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
        }
```

- 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);

```
function list() {
    var wallet = customer.getProfile().getWallet();
    var paymentInstruments = wallet.getPaymentInstruments(dw.order.PaymentInstrument.METHOD_CREDIT_CARD);
    var paymentForm = app.getForm('paymentinstruments');

    paymentForm.clear();
    paymentForm.get('creditcards.storedcards').copyFrom(paymentInstruments);

    pageMeta.update(dw.content.ContentMgr.getContent('myaccount-paymentsettings'));

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

    app.getView({
        PaymentInstruments: paymentInstruments
    }).render('account/payment/paymentinstrumentlist');
}
```

- 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/pipelets/forter/ForterCustomerUpdate');
forterCall.execute(argCustomerUpdate);

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

  var argCustomerUpdate = {

  EventType: require('*/cartridge/scripts/lib/forter/forterConstants').CUSTOMER_PROFILE_UPDATE };

  var forterCall = require('*/cartridge/scripts/pipelets/forter/ForterCustomerUpdate');

  forterCall.execute(argCustomerUpdate);

- 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;
    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',
    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_VALIDATES: '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;
}
```

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

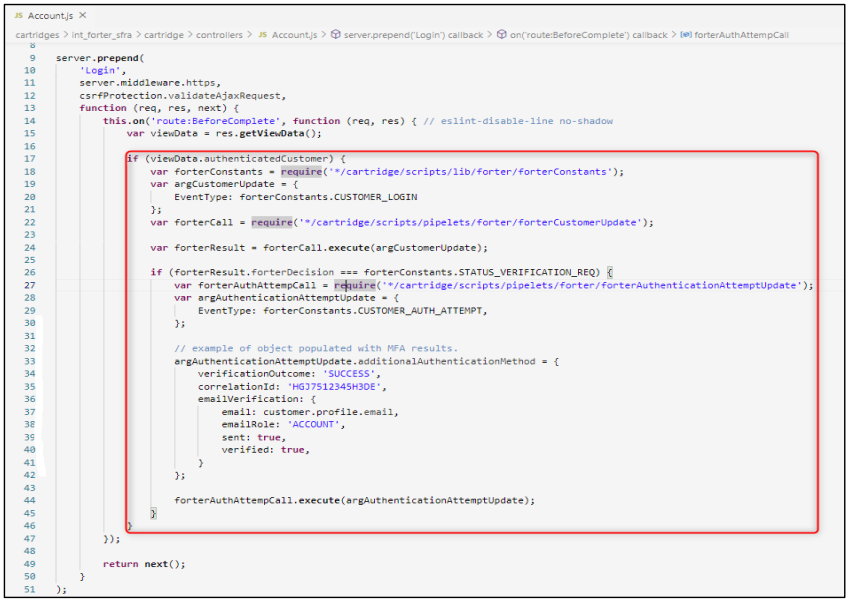
same controllers or templates extended or replaced – all mentioned code modifications must be added to the top level cartridge of the storefront application.

- 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/pipelets/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/pipelets/forter/ForterAuthenticationAttemptUpdate');
  forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
}
```



```
Account.js
cartridges > int_forter_sfra > cartridge > controllers > Account.js > server.prepend('Login') callback > on('route:BeforeComplete') callback > forterAuthAttempCall

server.prepend(
    'Login',
    server.middleware.https,
    csrfProtection.validateAjaxRequest,
    function (req, res, next) {
        this.on('route:BeforeComplete', function (req, res) { // eslint-disable-line no-shadow
            var viewData = res.getViewData();

            if (viewData.authenticatedCustomer) {
                var forterConstants = require('*/cartridge/scripts/lib/forter/forterConstants');
                var argCustomerUpdate = {
                    EventType: forterConstants.CUSTOMER_LOGIN
                };
                var forterCall = require('*/cartridge/scripts/pipelets/forter/forterCustomerUpdate');

                var forterResult = forterCall.execute(argCustomerUpdate);

                if (forterResult.forterDecision === forterConstants.STATUS_VERIFICATION_REQ) {
                    var forterAuthAttempCall = require('*/cartridge/scripts/pipelets/forter/forterAuthenticationAttemptUpdate');
                    var argAuthenticationAttemptUpdate = {
                        EventType: forterConstants.CUSTOMER_AUTH_ATTEMPT,
                    };

                    // example of object populated with MFA results.
                    argAuthenticationAttemptUpdate.additionalAuthenticationMethod = {
                        verificationOutcome: 'SUCCESS',
                        correlationId: 'HGJ7512345H3DE',
                        emailVerification: {
                            email: customer.profile.email,
                            emailRole: 'ACCOUNT',
                            sent: true,
                            verified: true,
                        }
                    };

                    forterAuthAttempCall.execute(argAuthenticationAttemptUpdate);
                }
            }
        });

        return next();
    }
);
```

- 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/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);

- 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/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);

- 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/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);

- 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/pipelets/forter/ForterCustomerUpdate');
  forterCall.execute(argCustomerUpdate);

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

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

CheckoutServices.js

```
170
171     // Handles payment authorization
172     var handlePaymentResult = COHelpers.handlePayments(order, order.orderNo);
173     if (handlePaymentResult.error) {
174         if (sitePrefs.forterShowDeclinedPage === true && sitePrefs.forterCustomDeclineMessage) {
175             res.json({
176                 error: true,
177                 errorMessage: sitePrefs.forterCustomDeclineMessage
178             });
179         } else {
180             res.json({
181                 error: true,
182                 errorMessage: Resource.msg('error.technical', 'checkout', null)
183             });
184         }
185
186         this.emit('route:Complete', req, res);
187         return;
188     }
189
190     var fraudDetectionStatus = hooksHelper('app.fraud.detection', 'fraudDetection', currentBasket, require('*/cartridge/
191     if (fraudDetectionStatus.status === 'fail') {
192         Transaction.wrap(function () { OrderMgr.failOrder(order, true); });
193
194         // fraud detection failed
195         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"/>

checkout.isml

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

<isinclude template="/components/modules" sf-toolkit="off" />

<!DOCTYPE html>
<html lang="en">
    <head>
        <isinclude template="/common/htmlHead" />
        <isactivedatahead/>
    </head>
    <body>
        <div class="page">
            <isinclude template="/components/header/pageHeaderNomenu" />
            <isreplace/>
            <isinclude template="/components/footer/pageFooter" />
        </div>
        <isinclude template="/common/scripts" />

        <isinclude template="custom/fortersnippetjs"/>

        <isinclude url="${URLUtils.url('ConsentTracking-Check')}"/>
    </body>
</html>
```

- 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"/>

page.isml

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

<isinclude template="/components/modules" sf-toolkit="off" />

<!DOCTYPE html>
<html lang="en">
    <head>
        <isinclude template="/common/htmlHead" />
        <isactivedatahead/>
    </head>
    <body>
        <div class="page" data-action="${pdict.action}" data-querystring="${pdict.queryString}" >
            <isinclude template="/components/header/pageHeader" />
            <isreplace/>
            <isinclude template="/components/footer/pageFooter" />
        </div>
        <div class="error-messaging"></div>
        <div class="modal-background"></div>
        <iscontentasset aid="cookie_hint" />
        <isinclude template="/common/scripts" />

        <isinclude template="custom/fortersnippetjs"/>

        <isinclude url="${URLUtils.url('ConsentTracking-Check')}"/>
    </body>
</html>
```

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

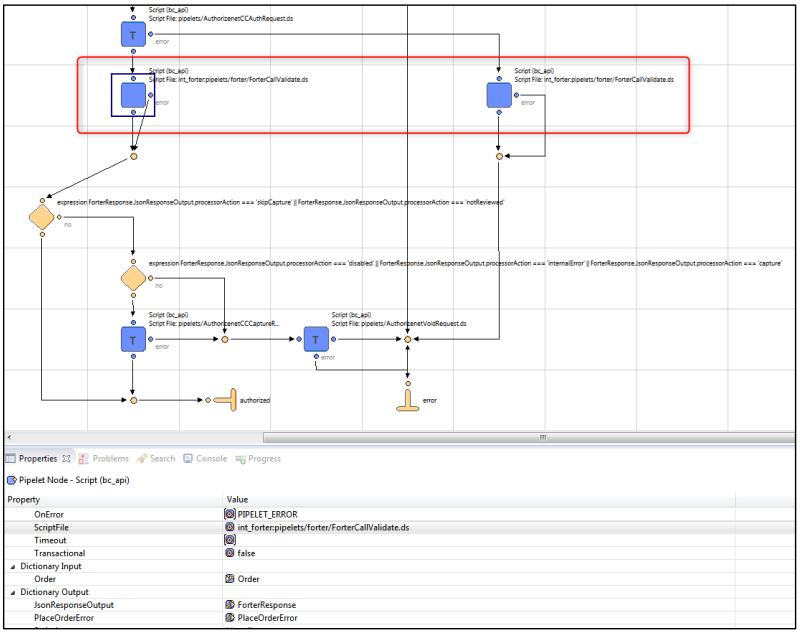
Payment Gateway error (processor declined authorization) - Please note in the example authorize.net diagram below, in case the credit card is not authorized by the payment gateway (e.g. expired credit card), the order is still sent to Forter. The request to Forter in this case should get a "NOT REVIEWED" 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/pipelets/AuthorizenetCCAuthRequest'),
        authResponse              = authorizenetCCAuthRequest.execute(argCCAuth);
    return authResponse;
}

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

function doVoid(argVoid) {
    var authorizenetVoidRequest = require('~/cartridge/scripts/pipelets/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.

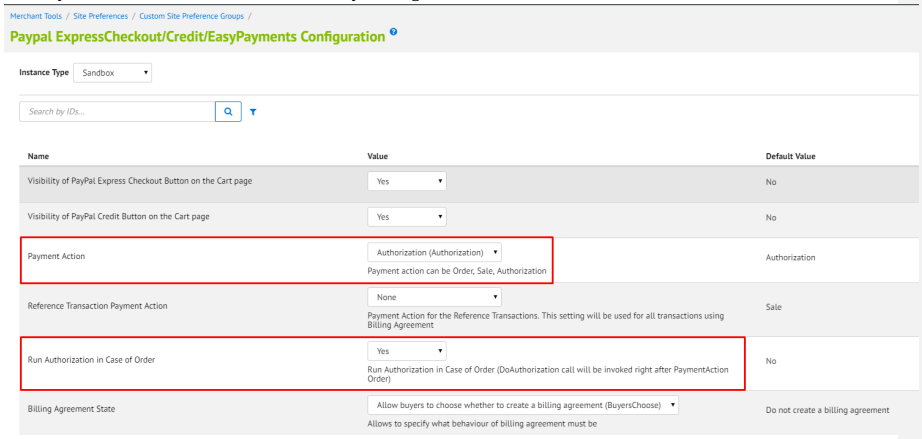
```
function Authorize(orderNumber, paymentInstrument, paymentProcessor) {
    var serverErrors = [],
        fieldErrors  = {},
        error        = false;

    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/pipelets/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/pipelets/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/pipelets/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/pipelets/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/pipelets/AuthorizenetCCAuthRequest'),
        authResponse              = authorizenetCCAuthRequest.execute(argCCAuth);
    return authResponse;
}

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

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

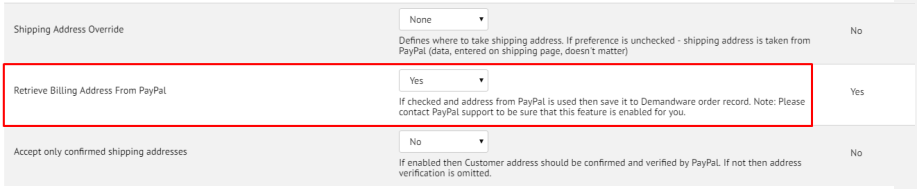
Sample Paypal Express Checkout Flow

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.

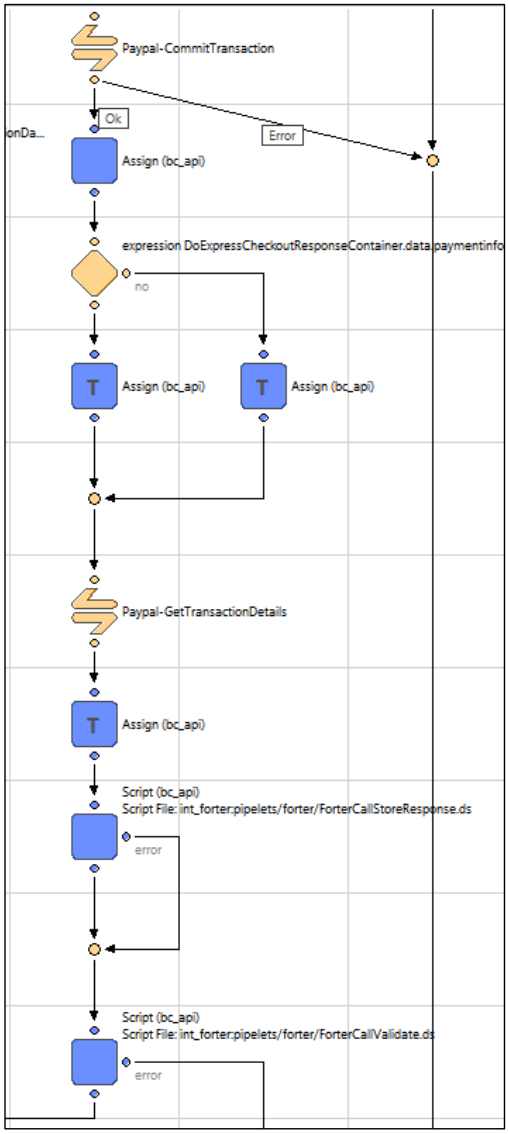
a. The Express Checkout should include Authorization before Forter is called. This can be done by setting the Payment Action to "Authorization" or by setting Run Authorization in case of Order to Yes.



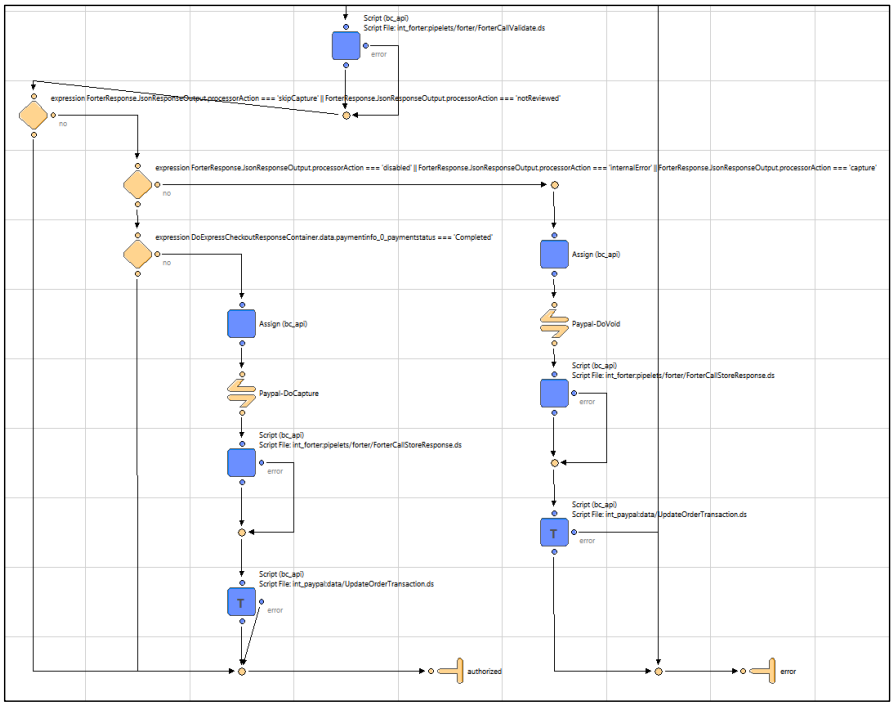
b. **You must make sure the Paypal cartridge is set to send to Salesforce Commerce Cloud the billing information. Please note this configuration may need external permissions added by Paypal support to your Paypal account**.



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.

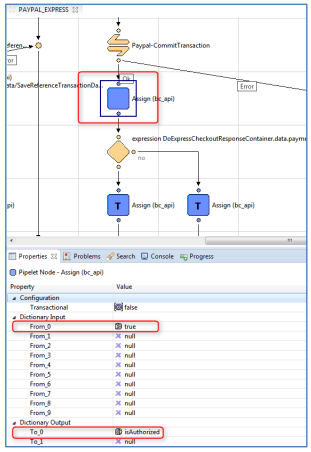

Paypal-CommitTransaction
Ok
Error
onDa...
Assign (bc_api)
expression DoExpressCheckoutResponseContainer.data.paymentinfo
no
Assign (bc_api)
Assign (bc_api)
Paypal-GetTransactionDetails
Assign (bc_api)
Script (bc_api)
Script File: int_forter:pipelets/forter/ForterCallStoreResponse.ds
error
Script (bc_api)
Script File: int_forter:pipelets/forter/ForterCallValidate.ds
error

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.

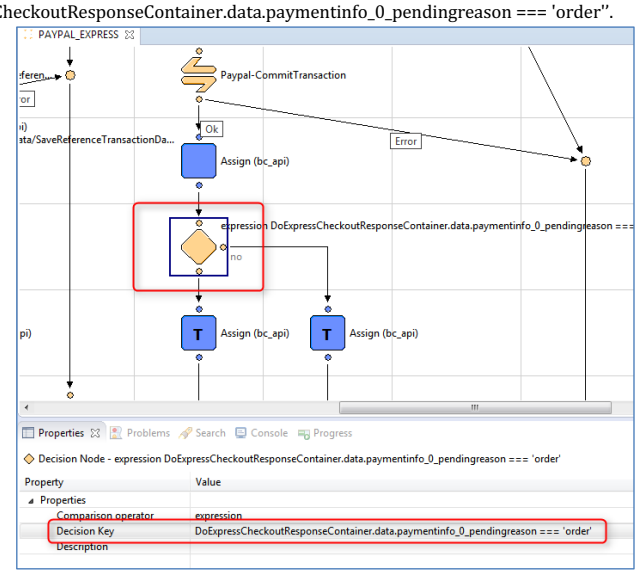


Below is detailed information about modifications from the previous images:

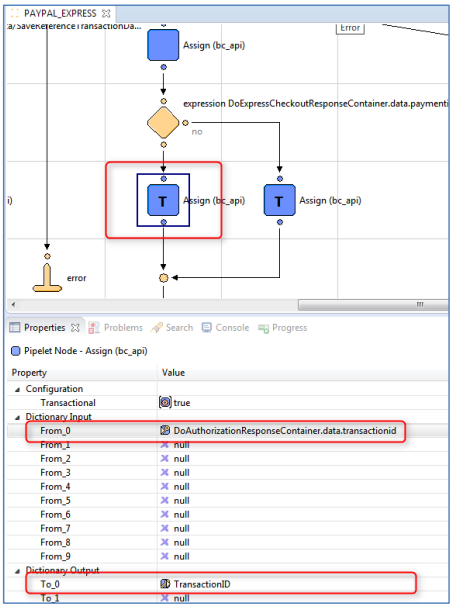
1. Add an Assign node. Set 'From_0' to 'true'; 'To_0' to 'isAuthorized'.



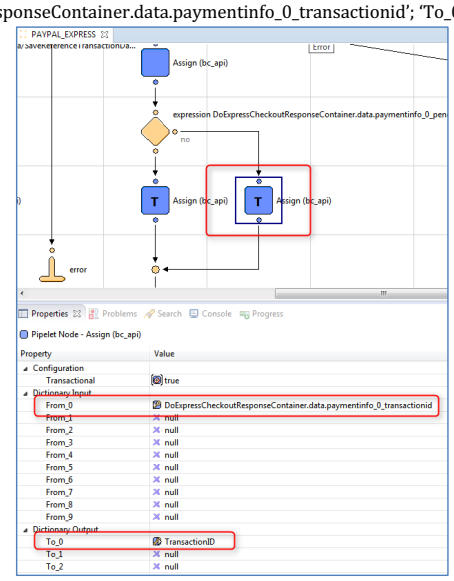
2. Add a Decision node. Set 'Decision Key' to 'DoExpressCheckoutResponseContainer.data.paymentinfo_0_pendingreason === 'order''.



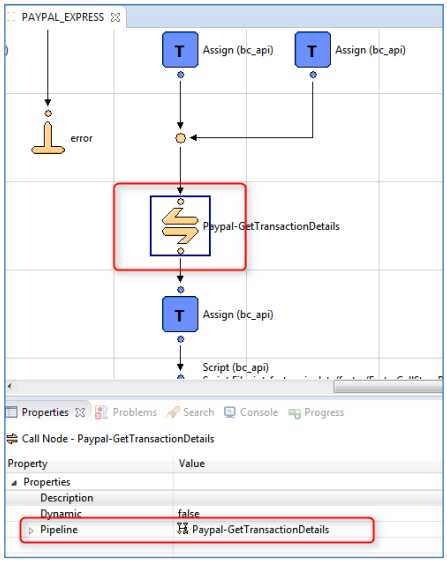
3. Add an Assign node. Set 'From_0' to 'DoAuthorizationResponseContainer.data.transactionid'; 'To_0' to 'TransactionID'.



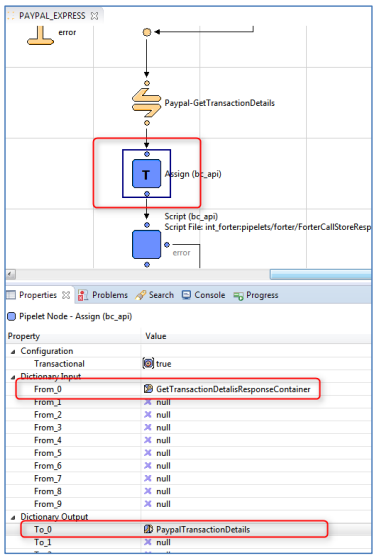
4. Add an Assign node. Set ‘From_0’ to ‘DoExpressCheckoutResponseContainer.data.paymentinfo_0_transactionid’; ‘To_0’ to ‘TransactionID’.



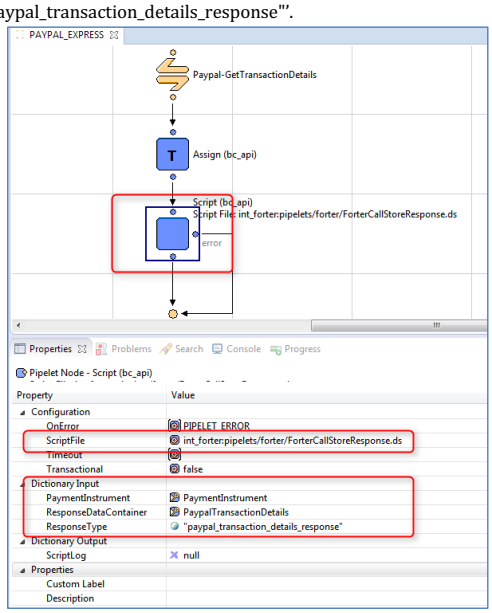
5. Add a Call node. Set ‘Pipeline’ to ‘Paypal-GetTransactionDetails’.



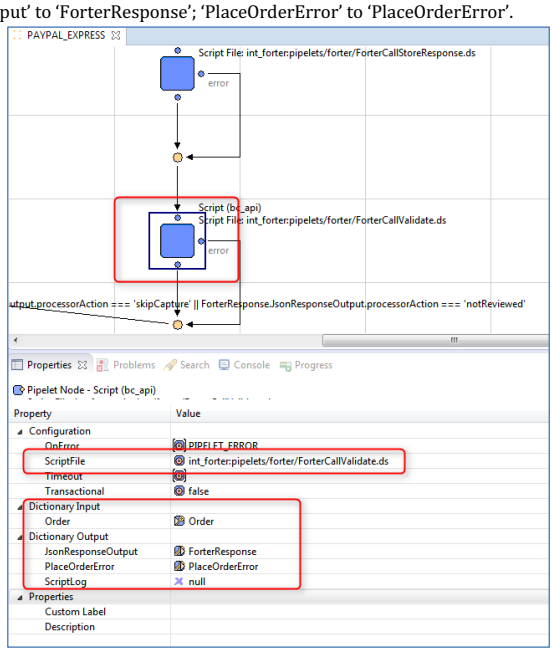
6. Add an Assign node. Set ‘From_0’ to ‘GetTransactionDetalisResponseContainer’; ‘To_0’ to ‘PaypalTransactionDetails’.



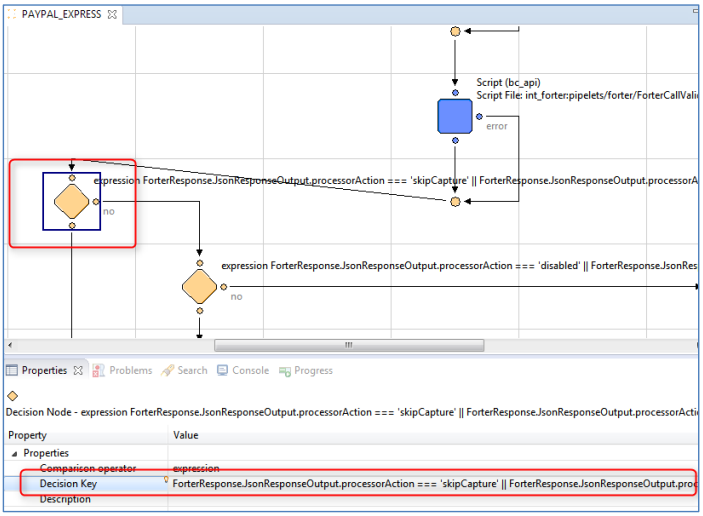
7. Add a Script node. Set ‘ScriptFile’ to ‘int_forter:pipelets/forter/ForterCallStoreResponse.ds’; ‘PaymentInstrument’ to ‘PaymentInstrument’; ‘ResponseDataContainer’ to ‘PaypalTransactionDetails’; ‘ResponseType’ to ‘"paypal_transaction_details_response"’.



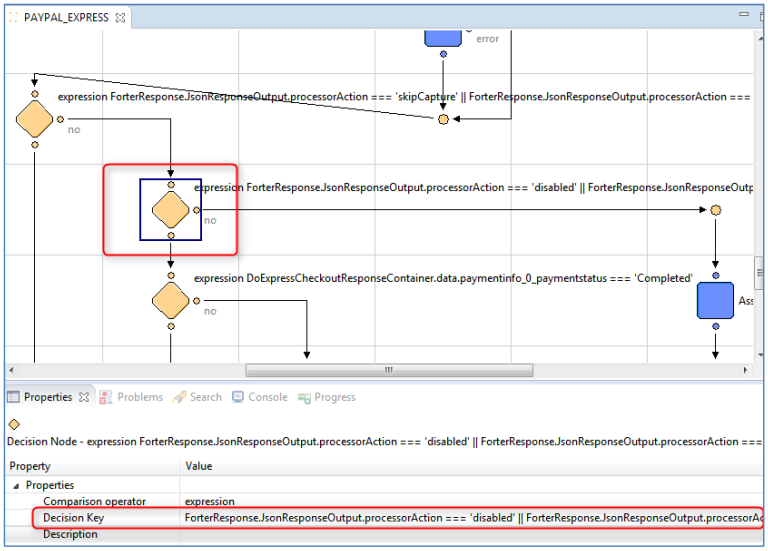
8. Add a Script node. Set ‘ScriptFile’ to ‘int_forter:pipelets/forter/ForterCallValidate.ds’; ‘Order’ to ‘Order’; ‘JsonResponseOutput’ to ‘ForterResponse’; ‘PlaceOrderError’ to ‘PlaceOrderError’.



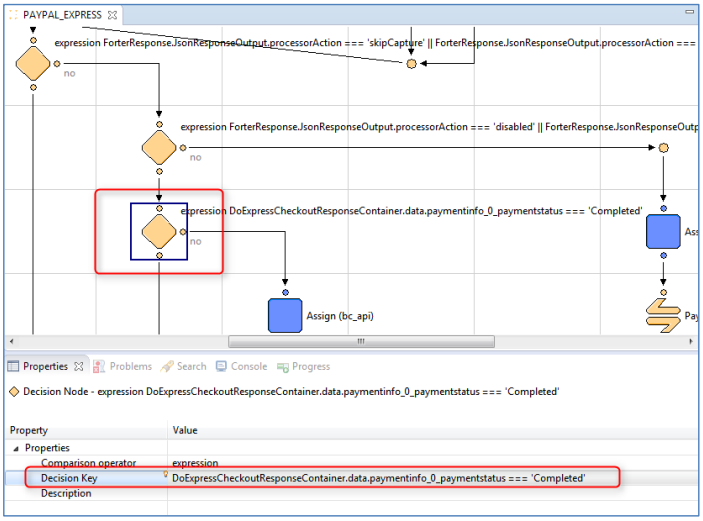
9. Add a Decision node. Set ‘Decision Key’ to ‘ForterResponse.JsonResponseOutput.processorAction === 'skipCapture' || ForterResponse.JsonResponseOutput.processorAction === 'notReviewed''.



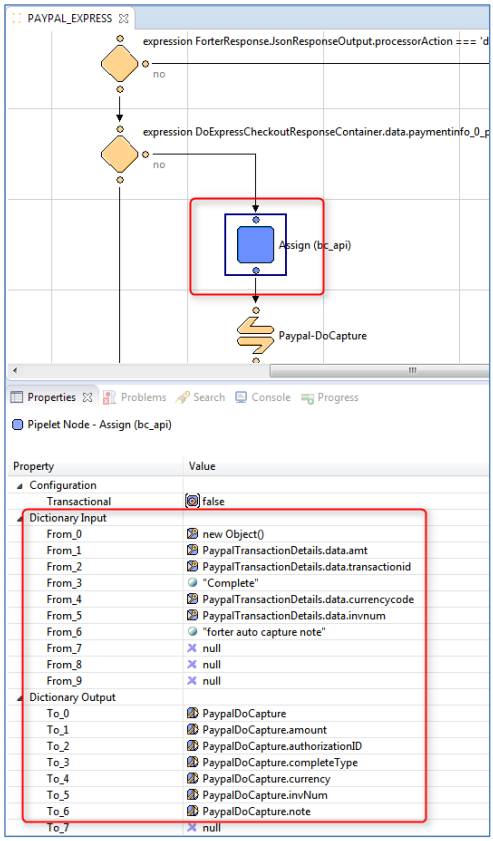
10. Add a Decision node. Set ‘Decision Key’ to ‘ForterResponse.JsonResponseOutput.processorAction === 'disabled' || ForterResponse.JsonResponseOutput.processorAction === 'internalError' || ForterResponse.JsonResponseOutput.processorAction === 'capture''.



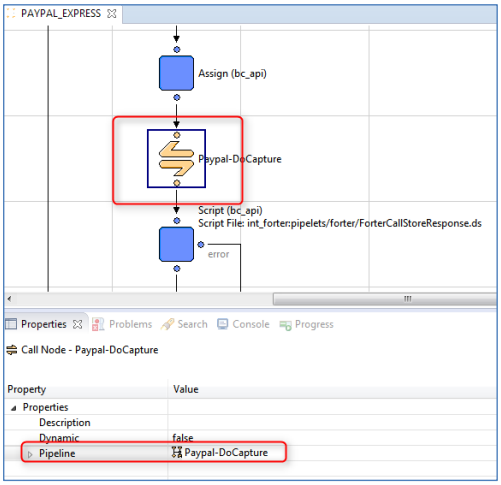
11. Add a Decision node. Set ‘Decision Key’ to ‘DoExpressCheckoutResponseContainer.data.paymentinfo_0_paymentstatus === 'Completed''.



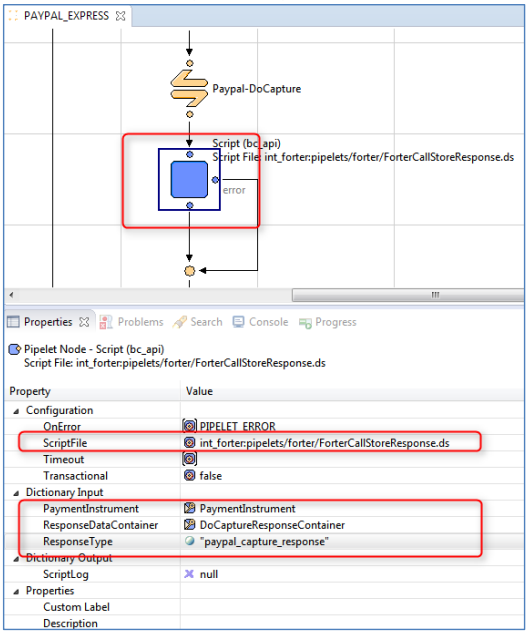
12. Add an Assign node. Set
    'From_0' to 'new Object()';
    'From_1' to 'PaypalTransactionDetails.data.amt';
    'From_2' to 'PaypalTransactionDetails.data.transactionid';
    'From_3' to '"Complete"';
    'From_4' to 'PaypalTransactionDetails.data.currencycode';
    'From_5' to 'PaypalTransactionDetails.data.invnum';
    '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';



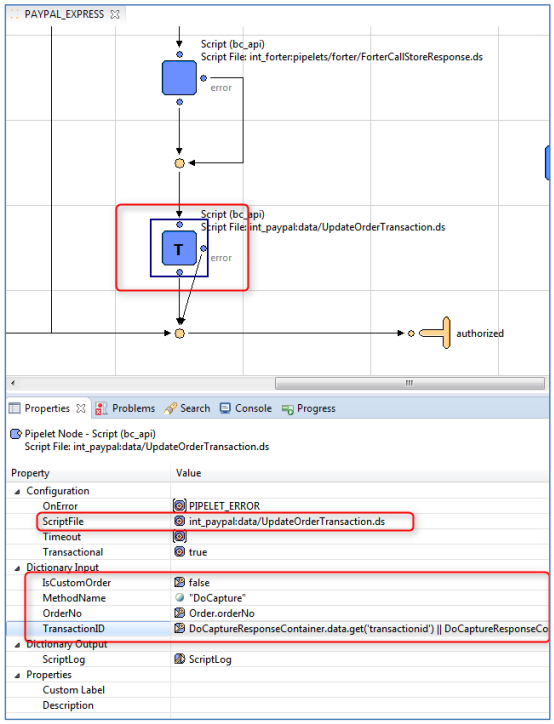
13. Add a Call node. Set ‘Pipeline’ to ‘Paypal-DoCapture’.



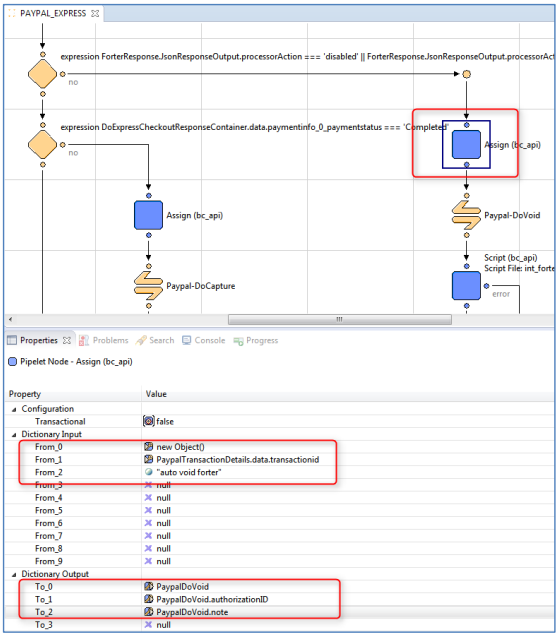
14. Add a Script node. Set ‘ScriptFile’ to ‘int_forter:pipelets/forter/ForterCallStoreResponse.ds’; ‘PaymentInstrument’ to ‘PaymentInstrument’; ‘ResponseDataContainer’ to ‘DoCaptureResponseContainer’; ‘ResponseType’ to ‘"paypal_capture_response"’.



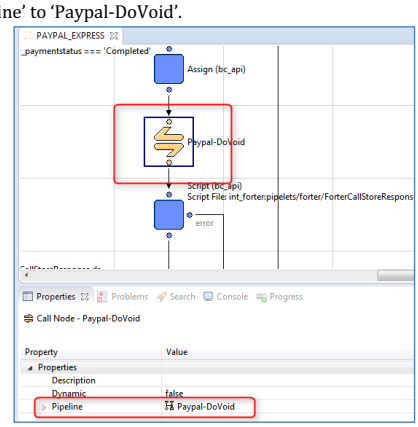
15. Add a Script node. Set
'ScriptFile' to 'int_paypal:data/UpdateOrderTransaction.ds';
'IsCustomOrder' to 'false';
'MethodName' to '"DoCapture"';
'OrderNo' to 'Order.orderNo';
'TransactionID' to 'DoCaptureResponseContainer.data.get('transactionid') ||
DoCaptureResponseContainer.data.get('authorizationid')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



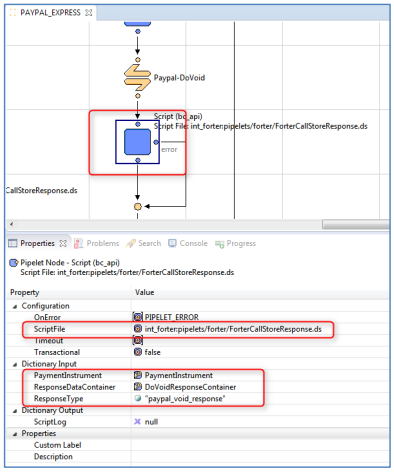
16. Add an Assign node. Set
    'From_0' to 'new Object()';
    'From_1' to 'PaypalTransactionDetails.data.transactionid';
    'From_2' to '"auto void forter"';
    'To_0' to 'PaypalDoVoid';
    'To_1' to 'PaypalDoVoid.authorizationID';
    'To_2' to 'PaypalDoVoid.note';



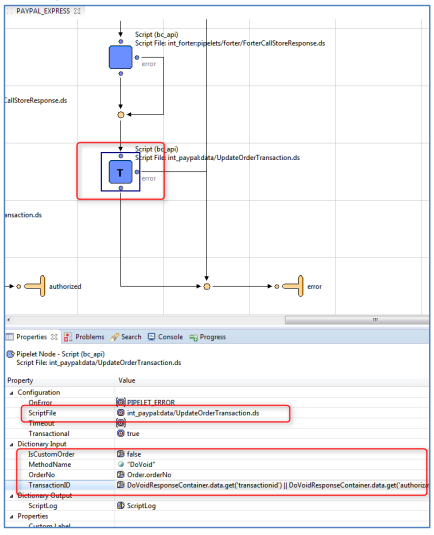
17. Add a Call node. Set 'Pipeline' to 'Paypal-DoVoid'.



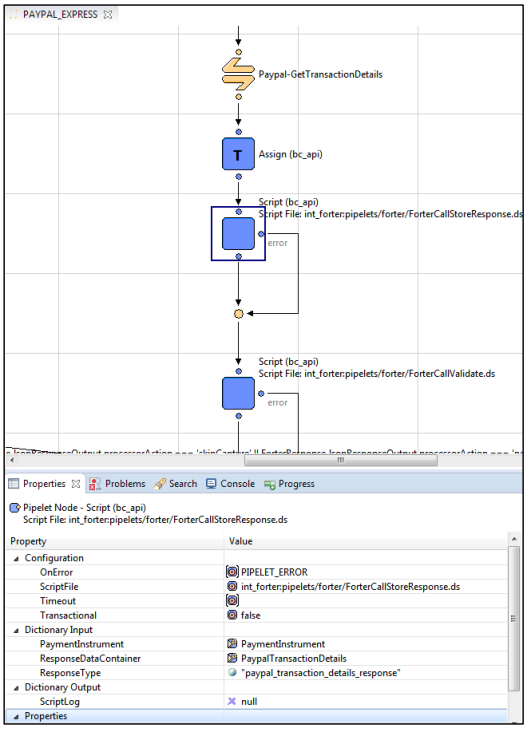
18. Add a Script node. Set ‘ScriptFile’ to ‘int_forter:pipelets/forter/ForterCallStoreResponse.ds’; ‘PaymentInstrument’ to ‘PaymentInstrument’; ‘ResponseDataContainer’ to ‘DoVoidResponseContainer’; ‘ResponseType’ to ‘"paypal_void_response"’.



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 PaypPal-GetTransacrionDetails and passed into the int_forter:pipelets/forter/ForterCallStoreResponse.ds script.
In the script Dictionary Input tje "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 massage 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)){}`:

PAYPAL_EXPRESS.js

```
'use strict';

/* API Includes */
var Pipeline = require('dw/system/Pipeline');
var OrderMgr = require('dw/order/OrderMgr');
var URLUtils = require('dw/web/URLUtils');

function Handle(args) {
    var pdict = Pipeline.execute('PAYPAL_EXPRESS-Handle', {
        Basket: args.Basket,
        ContinueURL: URLUtils.https('Paypal-ContinueExpressCheckout')
    });
    if(pdict.isSuccess) {
        return {success: true};
    } else {
        return {error: true};
    }
}

function Authorize(args) {
    var pdict = Pipeline.execute('PAYPAL_EXPRESS-Authorize', {
        Order: OrderMgr.getOrder(args.OrderNo),
        PaymentInstrument: args.PaymentInstrument
    });
    if(pdict.isAuthorized) {
        if (!empty(pdict.ForterResponse.PlaceOrderError)) {
            return {error: true, forterErrorCode : pdict.ForterResponse.PlaceOrderError};
        } else if (
                    (!empty(pdict.ForterResponse.JsonResponseOutput.actionEnum) && pdict.ForterResponse.JsonResponseOutput.actionEnum == 'DECLINED')
                    &&
                    (!empty(pdict.ForterResponse.JsonResponseOutput.processorAction) && pdict.ForterResponse.JsonResponseOutput.processorAction != 'skipCapture')
                ) {
            return {error: true};
        } else {
            return {authorized: true};
        }
    } else {
        return {error: true};
    }
}
/*
 * Module exports
 */
/*
 * Local methods
 */
exports.Handle = Handle;
exports.Authorize = Authorize;
```

In order to handle the customized error massage 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              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
81      return {};
82  }
83
```

- inside the start() function

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

COPlaceOrder.js

```
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
169     } else if (handlePaymentsResult.missingPaymentInfo) {
170         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     }
```

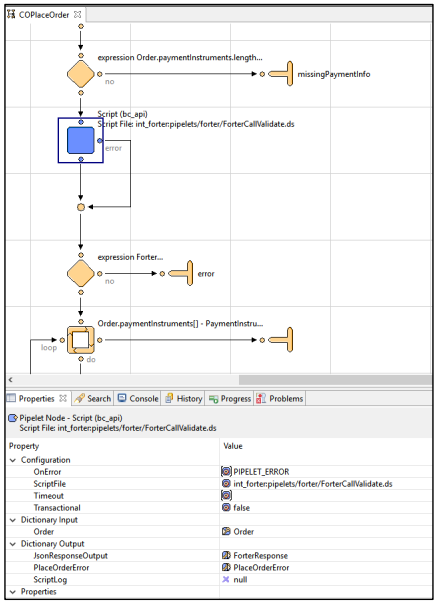
The verificationResults and paymentGatewayData object in the ForterCreditCard function must be adjusted according to the payment gateway used

Pre-Authorization Flow

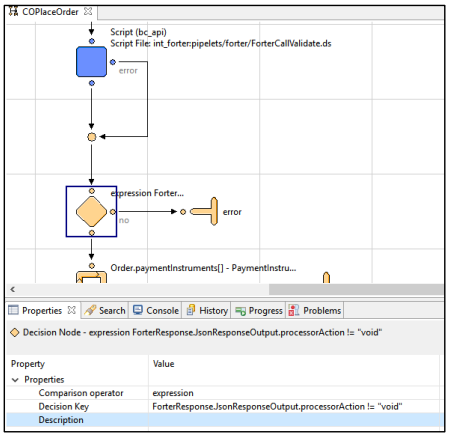
This is not part of the default integration with Forter. This section should be implemented only on specific use cases. Please consult your account manager/integration engineer in Forter before implementing.

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:

1. 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,
      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

```
// these lines must be uncommented in case if you want to activate the pre-authorization flow or being included in a top-level cartridge
var orderNumber = order.getCurrentOrderNo();
var argOrderValidate = {
    Order                    : order,
    originalOrderNo          : orderNumber,
    orderValidateAttemptInput : 1,
    authorizationStep        : "PRE_AUTHORIZATION"
};
var forterCall       = require('*/cartridge/controllers/ForterValidate');
var forterDecision   = forterCall.ValidateOrder(argOrderValidate);

in case if no response from Forter, try to call one more time
if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
    argOrderValidate = {
        Order                    : order,
        originalOrderNo          : orderNumber,
        orderValidateAttemptInput : 2,
        authorizationStep        : "PRE_AUTHORIZATION"
    };
    forterCall       = require('*/cartridge/controllers/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)) {
        return {
            error       : true,
            errorMessage: forterDecision.PlaceOrderError.code
        };
    } else {
        return {
            error       : true,
            errorMessage: Resource.msg('error.technical', 'checkout', null)
        };
    }
}
```

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

```
        app.getController('Cart').Show();

        return {};
    }
    var handlePaymentsResult = handlePayments(order);

    if (handlePaymentsResult.error) {
        return Transaction.wrap(function () {
            OrderMgr.failOrder(order);

            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')
                };
            }
        });

    } else if (handlePaymentsResult.missingPaymentInfo) {
        return Transaction.wrap(function () {
            OrderMgr.failOrder(order);
            return {
                error: true,
                PlaceOrderError: new Status(Status.ERROR, 'confirm.error.technical')
            };
        });
    }

    var orderPlacementStatus = Order.submit(order);
    if (!orderPlacementStatus.error) {
        clearForms();
    }
    return orderPlacementStatus;
```

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/pipelets/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/pipelets/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

```
// these lines must be uncommented in case if you want to activate the pre-authorization flow or being included in a top-level cartridge
var orderNumber = order.getCurrentOrderNo();
var argOrderValidate = {
    orderNumber             : orderNumber,
    orderValidateAttemptInput : 1,
    authorizationStep       : "PRE_AUTHORIZATION"
};
var forterCall      = require('*/cartridge/scripts/pipelets/forter/forterValidate');
var forterDecision   = forterCall.validateOrder(argOrderValidate);

in case if no response from Forter, try to call one more time
if (forterDecision.result === false && forterDecision.orderValidateAttemptInput == 2) {
    argOrderValidate = {
        orderNumber             : orderNumber,
        orderValidateAttemptInput : 2,
        authorizationStep       : "PRE_AUTHORIZATION"
    };
    forterCall      = require('*/cartridge/scripts/pipelets/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;
}
```

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/pipelets/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 an
var argOrderUpdate = {
    orderNumber: orderNumber,
    updateAttempt: 1
},
forterCall = require('*/cartridge/scripts/pipelets/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*

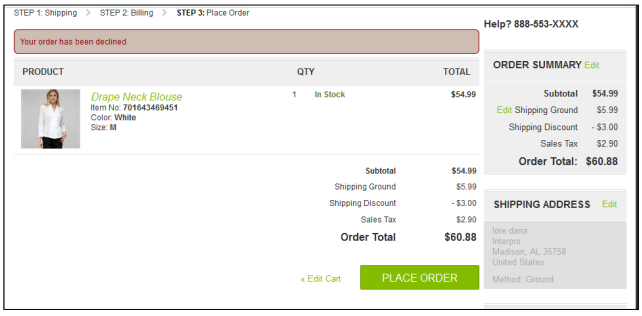
Please note that Forter's custom JavaScript snippet, which captures vital behavioral data, has been added to the SiteGenesis footer. SiteGenesis footer.isml has been modified in order to add the mentioned functionality.

```
<!-- ======================================================================== -->
<!-- ==                                                                    == -->
<!-- ==  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 massage 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

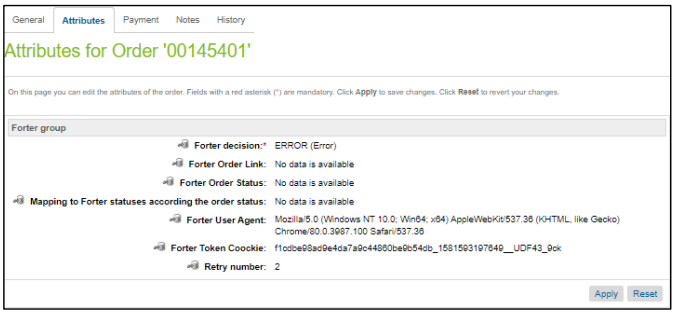
4.1 Data Storage

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.



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.

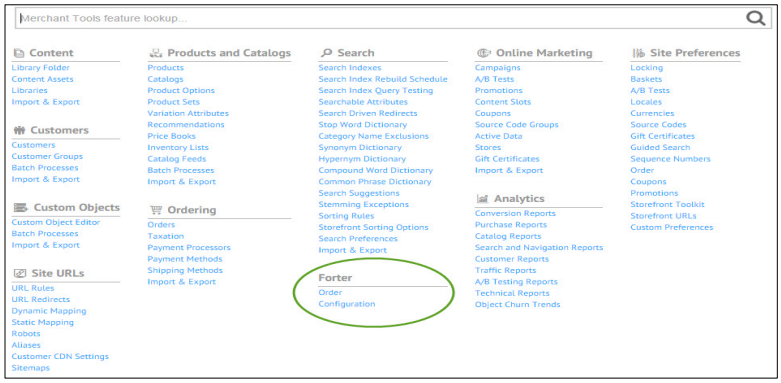
5.2 Business Manager

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*.

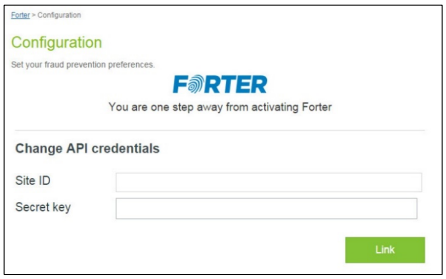


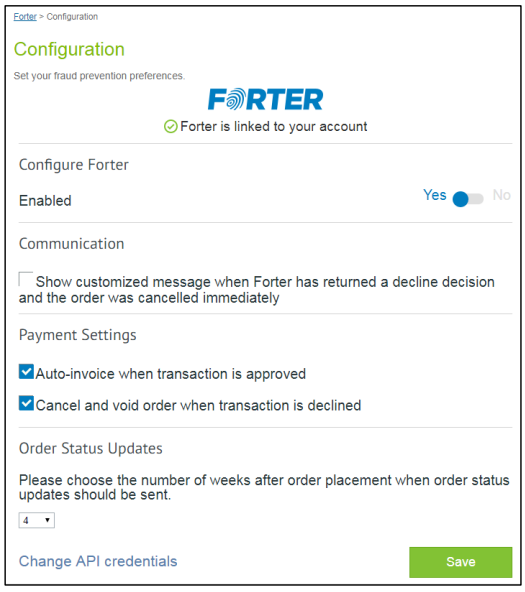
5.2.2 *Forter section*

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.

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.





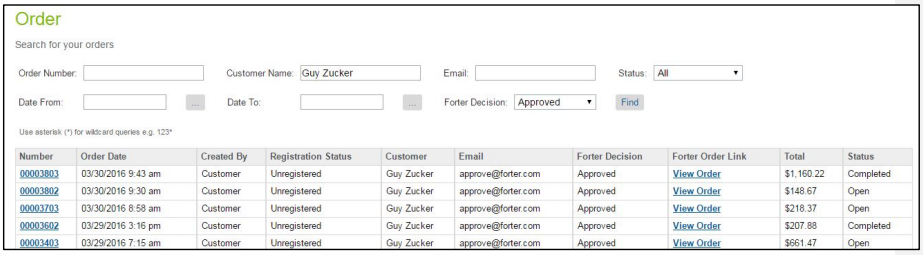
Forter configurations are saved in Salesforce Commerce Cloud site preferences. They can be manipulated through the Forter -> Configuration extension screen. Please do not manipulate them directly through site preferences because this way no Forter call for verification is made.

5.2.4 *Forter Orders*

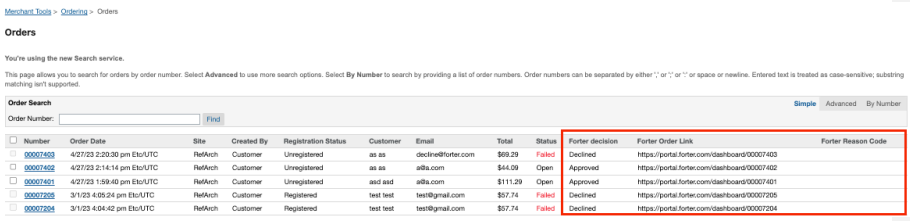
A Forter orders page is added. Thus the merchant can easily search for orders based on the Forter decision. The Forter Decision will influence the Order Status.

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.

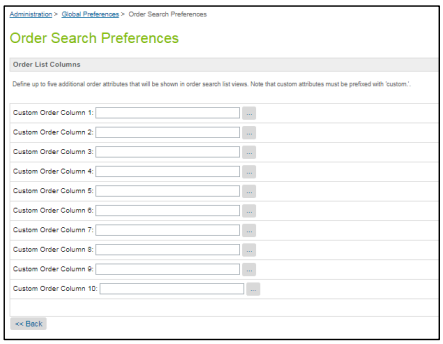


In addition, you may want to add the Forter Decision, the Forter Order Link and the Forter Reason Code to the default order search view.

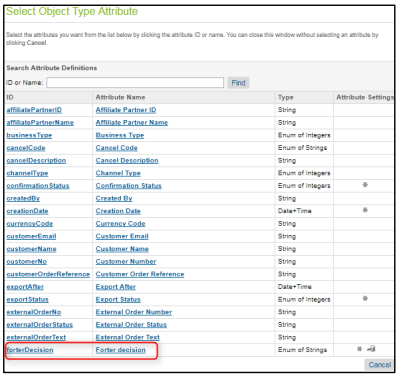


Steps to add custom fields described below:

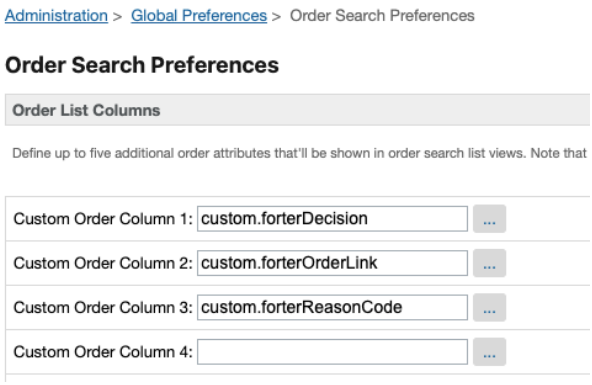
1. Go to Administration > Global Preferences > Order Search.



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

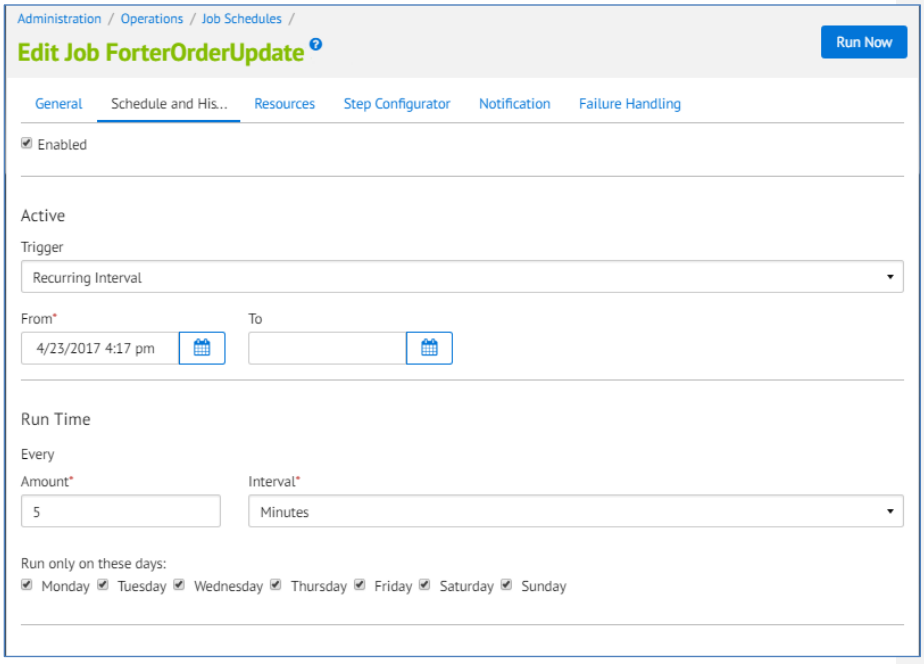


3. Repeat the steps noted above for the 'forterOrderLink' and 'forterReasonCode' attributes.



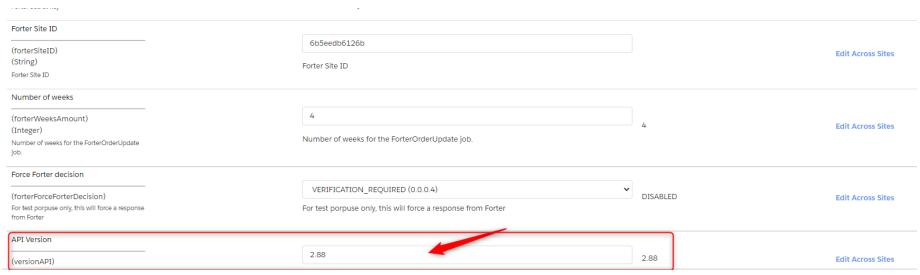
5.2.5 *Order Update job*

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.



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 forter preference and search for API Version attribute. The default value is 2.88 (the current version for forter API). You can change from there when a version shows up.



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: "${dw.web.URLUtils.url('ForterValidate-UpdateForterInfo').toString()}",
                    data: postInfo,
                    success: function(data) {}
                });
            }
        });
    })()
</script>
<script type="text/javascript" id="<isprint value="${dw.system.Site.getCurrent().getCustomPreferenceValue('forterSiteID')}" />">
(function () {
    var merchantConfig = {
        csp: false
    };
    You, now • Uncommitted changes
    var siteId = "<isprint 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(/%SN%/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