

Salesforce Commerce Cloud

Date: 31/08/18

Version 1.0.0

Copyright © Adyen N.V. 2018

Simon Carmiggeltstraat 6 - 50

1011 DJ Amsterdam

The Netherlands

Table of Contents

1	Use cases	5
2	Privacy, payments	5
3	Compatibility	6
4	Support	6
5	Settings in Adyen CA	6
5.1	Create a skin	6
5.2	Configure notifications	6
5.3	Set capture delay	7
5.4	Set a password for web service (ws) user	7
5.5	LIVE account	7
6	Integrating a cartridge	7
6.1	Set up	8
6.1.1	Metadata import	8
6.2	Business Manager configuration	8
6.2.1	Configure payment methods	9
6.2.2	Configure the cartridge	9
6.3	Services	11
6.4	Scheduled jobs	11
6.5	Customize	15
6.5.1	Storefront core changes	15
6.5.2	Storefront controller changes	25
6.6	Testing integration	32
6.6.1	Test credit cards	32
6.6.2	Business Manager configuration	32
6.6.3	Adyen Customer Area	33
7	Data storage	33
8	Set up Apple Pay on the Web	33
8.1	Prerequisites	33
8.2	Steps	33

9	Set up Terminal API	35
9.1	Before you begin	35
9.2	Step 1: Enable POS payments	35
9.3	Step 2: Configure services	35
9.4	Step 3: Configure a terminal	35
9.5	Step 4 (optional): Configure multiple terminals	35
9.5.1	Code sample	36



This document describes version 18.1.0 of the Adyen LINK cartridge. For version 17.1.0, click [here](#).

Adyen offers a [LINK cartridge](#) to integrate with [Salesforce Commerce Cloud](#) for ecommerce platforms. You can integrate your in-store payments through any of our [POS integrations](#) to offer a full omni-channel experience.

Create a merchant account in the [Customer Area](#), to know more read [Get started with Adyen](#).

Adyen offers the following integrations:

- Using the Adyen API, if you want to support cards only.
- Using the Redirect model for cards and local payment methods.

Some items that need to be configured in the [Customer Area](#) for the Adyen Integration to work correctly.

Parameter Name	Description
HMAC256 key	Secret key used to calculate a signature over the payment fields sent to Adyen. This should match what is used in the Adyen configuration settings in your Commerce Cloud site.
SkinCode	Code of the skin you want to use. You can use different skins, each having a different configuration, style, language, etc.
Notification URL and test page	URL where Adyen should send notification messages to. The script at this URL should always reply with [accepted]; otherwise, Adyen keeps sending messages. The Notification URL and its response can be tested from within the CA.
Payment Methods	You can select which payment methods should be displayed by configuring your skin. This can also be influenced with the 'allowedMethods' parameter that can be sent to Adyen for each payment.
Test HPP and skin	The skin and HPP can be tested from within the CA, to check the look and feel, translations, configured payment methods, etc.
Settings for the PAL adapter	To send Credit Card payments directly (without redirecting to the HPP) you can use the PAL adapter. The user, password and location can be configured here.
Notifications User Name	The username used by the Adyen notification system in order to authenticate the calls with Commerce Cloud. Should be the same as the one set in Commerce Cloud system under "Adyen Notifications User" preference. (Found under: Settings > Notifications > User Name)
Notifications Password	The password used by the Adyen notification system in order to authenticate the calls with Commerce Cloud. Should be the same as the one set in Commerce Cloud system under "Adyen Notifications Password" preference. (Found under: Settings > Notifications > Password)

- Successful payment authorizations lead to a **Paid** order status and **Ready for Export** status.
- Refused payment authorizations lead to a 'Not Paid' order payment status, and 'Not Exported' export status.
- Cancels and refunds lead to a 'Not Paid' order payment status (even if the status was 'Paid' before), and 'Not Exported' export status.
- Refused payments have the status 'Refused' in the Adyen CA, but 'AUTHORISATION' Eventcode in Business Manager.
- In the cartridge these API methods are implemented: 'Capture', 'Cancel Before Capture', and 'Cancel Or Refund'. The statuses require custom integration into the end site, they are optional functions and are not described in this document from a position of integration.
- The cartridge makes use of the [Adyen Hosted](#) Client-Side Encryption library, other methods require additional customization.

Use cases

Registered and unregistered shoppers will be able to follow the standard Site Genesis flow with the following changes:

- Adyen redirect, Adyen API and Gift Certificates (GC) will be the only payment options if Adyen redirect is enabled:
 - no entering of credit card details on the Billing page is possible when the user selects this payment method (as this is done on the Adyen HPP) and Directory Lookup feature is disabled
 - the possibility to choose one of the predefined payment methods on the Billing page with further redirection to Adyen HPP, if Directory Lookup is enabled and the user selected Adyen payment method
 - a redirect to Adyen HPP after clicking the 'SUBMIT ORDER' button on the Order Confirmation page (only if no GCs are used and/or they don't cover the whole amount)
 - a return to the Order Summary page after successful payment authorization
 - a return to the Order Confirmation page, if the user cancelled the payment on Adyen HPP
 - a return to the Order Confirmation page with an error message displayed, if the payment was refused
- If Adyen API payments are enabled:
 - credit card details are entered and stored in your Commerce Cloud shop
 - for the other payment methods you can still redirect to the Adyen HPP, but can disable credit cards via the configuration of your skin

Privacy, payments

- If only the redirect method is used, all payment data is entered into the Adyen HPP by the shopper and no Credit Card data will be stored inside Commerce Cloud (except the brand of used card)
- The shoppers email address and shopper ID used in Commerce Cloud are sent to Adyen to allow 'recurring' and/or 'one-click' payment

Compatibility

The integration is based on the Site Genesis demo store provided by Commerce Cloud.

Versions:

- Commerce Cloud 17.1.0
- SiteGenesis 103.1.3


Support

In case of problems with the integration or connection to Adyen, contact the [Support Team](#) or your Adyen account manager.

Supply as much information as possible: your merchant account, skin code, time, order number, PSP reference, etc.

Settings in Adyen CA

Before configuring the settings, make sure that the [Customer Area](#) set up is complete.

 If you don't already have a test account, read [Get started with Adyen](#).

1. Go to the [Customer Area](#) and log in using your Adyen test account credentials.
2. In the main menu, click **Account**.
3. Select the name of your Merchant account.

Create a skin

To use the Redirect model, you have to define a skin in your [Customer Area](#) test environment:

1. Log in to your [Customer Area](#) with your test account credentials.
2. In the main menu, go to **Account > Skins**.
3. If no skin was configured yet, do so by clicking the **New** tab.
4. Enter a description for your skin.
5. Note down the **Skin Code**. You will need it later when [configuring the cartridge](#).
6. Click **Generate new HMAC key** for the Test and Live platform.
Note down these HMAC keys. You will need them later when [configuring the cartridge](#).
7. Leave the **Result URLs** and **Continue-to URLs** empty.

Configure notifications

1. Go to **Account > Server Communication**.
2. Click **Add** next to Standard Notification.

3. Under the **Transport** section:
 - a. Enter your website URL followed by **/Adyen-Notify**
 - b. From the **SSL Version** dropdown, select **TLSv.1.2**.
 - c. Select the **Active** checkbox.
 - d. From the **Method** dropdown, select **HTTP**.
4. Under the **Authentication** section, enter a User Name and Password.
Take a note of this information. You will need it later when [configuring the cartridge](#).
5. Test whether notifications have been configured correctly. Navigate to the Test Notifications section and click **Test Configuration**. If the result is **[Accepted]** your notifications are working correctly.
6. Click **Save Configuration**.

Set capture delay

1. Go to **Account > Merchant Settings**.
The default value for [Capture Delay](#) is Immediate, you may adjust it according to your preferences.
2. Click **Submit**.

Set a password for web service (ws) user

1. Go to **Account > Users**.
2. From the first dropdown in the upper left table cell, select **System**.
3. Click the **ws user** (ws@Company.YourCompanyCode).
If you have not yet set a password for your ws user yet, you can do it by clicking **Generate**.
Take a note of the password. as you need it later on when [configuring the Salesforce Commerce Cloud](#).
4. Click **Save**.

LIVE account

1. After your merchant account is enabled on the Adyen live platform, log in to the live [Customer Area](#) using your Adyen live account credentials .
2. In the main menu, click **Account > Merchant account**.
3. Select your Merchant account.
4. Configure the notifications, merchant account settings, and ws user following the steps described above.

Integrating a cartridge

To integrate with Salesforce Commerce Cloud using the Adyen LINK cartridge, you need to complete the following steps:

- [Set up your Adyen CA](#)
- [Set up the cartridge](#)
- [Configure the Business Manager](#)

- [Configure and enable Scheduled jobs](#)
- [Test integration](#)

Set up

The Adyen LINK Cartridge can be used with either a controller based SiteGenesis or a pipeline.

- Controller-based integration – Install **int_adyen**, **bm_adyen**, and **int_adyen_controllers** cartridges using Commerce Cloud UX-studio.
- **DEPRECATED** Pipeline-based integration – Install **int_adyen**, **bm_adyen**, and **int_adyen_pipelines** cartridges using Commerce Cloud UX-studio.

Metadata import

To add new configuration items, import the predefined metadata:

1. Download the [installation package](#) on our GitHub page.
2. Open the **package/metadata/site_import/sites/** folder.
3. Rename the **yourSiteId** folder to the ID of your site in the Business Manager.
4. Zip the **site_import** folder.
5. In the Business Manager, go to **Administration > Site Development > Site Import & Export** and import the zipped file.

After the import, attributes named **Adyen[attributeName]** are added to:

- **Administration > Site Development > System Object Types > Site Preferences > Attribute Definitions**
- **Administration > Site Development > System Object Types > Order > Attribute Definitions**
- **Administration > Site Development > Custom Object Types**

Also, the following services are added to **Administration > Operations > Services**:

- **AdyenPayment**
- **AdyenPayment3DSecure**
- **AdyenPaymentMethods**
- **AdyenRecurring**
- **AdyenRecurringDisable**

Business Manager configuration

Set up the cartridge path

1. Go to **Administration > Sites > Manage Sites > [yourSite] > Settings** .

2. In the **Cartridges** box, enter the following value, with **[app_storefront]** replaced by your cartridge name:

int_adyen_controllers:int_adyen:[app_storefront]_controllers:[app_storefront]_core

Configure payment methods

Credit cards

To process credit cards:

1. Go to **Merchant tools > Ordering > Payment Methods**.
2. Click **CREDIT_CARD**.
3. In the **Payment Processor** drop-down list, select **ADYEN_CREDIT**.

Mastercard issues cards with either a 2-series or 5-series BIN. By default, the cartridge only supports a 5-series BIN. If you also want to accept payments from a 2-series BIN:

1. Go to **Merchant tools > Ordering > Payment Methods**.
2. Click the **Credit/Debit Cards** button. This opens a window for managing credit/debit cards.
3. Select **Master Card**.
4. In the **Card Number Verification** box, enter **222100-272099, 510000-559999**.

Local payment methods

To process local payment methods:

1. Go to **Merchant tools > Ordering > Payment Methods**.
2. Click **Adyen**.
3. In the **Payment Processor** drop-down list, select, select **Adyen**.

Configure the cartridge

Before you begin, [import the metadata](#), and check that Adyen attributes have been added to **Administration > Site Development > System Object Types > Site Preferences > Attribute Definitions**.

To configure the cartridge:

1. Go to **Merchant tools > Site Preferences > Custom Preferences > Adyen**.
2. Configure the following site preference settings:

Name	Description
merchantCode	Name of the Adyen merchant account for which the payments will be processed.
Test /Production mode	Select (TEST) for test mode or (LIVE) for production mode.

Name	Description
skinCode	Code of the skin that you want to use. You noted this down when you configured your Adyen CA .
HMACkey	The secret HMAC key linked to your skin. Use the key for the test platform if Test /Production mode is set to (TEST) , and the key for the live platform if Test/Production is set to (LIVE) . You noted these keys down when you configured your Adyen CA .
Enable Adyen debug mode	Select Yes to display the Pay button before redirecting to Adyen. This allows you to check the parameters sent in the request.
CSE enabled	Select Yes to enable Client Side Encryption .
CSE library token	Your CSE library token. For more information, see How to get CSE library token .
Switch to directory lookup	Select Yes to enable directory lookup. For more information, see Hosted Payment Pages .
Payment selection	<div>DEPRECATED</div> <p>Only applicable if Switch to Directory Lookup is set to No.</p> <p>Type of the payment flow. For more information, see Hosted Payment Pages.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • one (one) – The payment process takes place within a single page. • multi (multi) – The payment process is split into several pages.
Adyen recurring payments enabled	Select Yes to enable recurring payments.
Recurring type	<p>Type of the recurring contract that you want to use. Possible values:</p> <ul style="list-style-type: none"> • RECURRING – The shopper opts to store their card details for future use. The shopper is present for the subsequent transaction, for cards the security code (CVC/CVV) is required. • ONECLICK – Payment details are stored for future use. For cards, the security code (CVC/CVV) is not required for subsequent payments. This is used for shopper not present transactions. • RECURRING,ONECLICK – Payment details are stored for future use. This allows the use of the stored payment details regardless of whether the shopper is on your site or not.

Name	Description
Adyen notifications user	The user name for notifications that you set in your Adyen CA. You noted this down when you configured your Adyen CA .
Adyen notifications password	The password for notifications that you set in your Adyen CA. You noted this down when you configured your Adyen CA .
Open invoice white list methods	Comma-separated list of open invoice methods that you want to enable. For more information, see Klarna, AfterPay and RatePay open invoice . Possible open invoice methods: <ul style="list-style-type: none"> • klarna • ratepay • afterpay_default • afterpay_directdebit
RatePay Id	The unique RatePAY Id provided by RatePAY integration consultant.

Services

The services contain the authentication for the communication between Salesforce Commerce Cloud and Adyen.

To configure Adyen services in the Business Manager:

1. Go to **Administration > Operations > Services**.
You see the services named **Adyen[serviceName]** that were added when you [imported the metadata](#).
2. Click the **Credentials** tab.
3. In the **Name** column, click the name of the service to open its details page.
4. Set **User** and **Password** to your web service (ws) username and password. You noted these down when you [configured your Adyen CA](#).



Once you are ready to go live, make sure that you switch to the [live endpoints](#), and use your live credentials.

Scheduled jobs

Adyen sends notifications to the Commerce Cloud endpoint updating the payment status. Commerce Cloud stores them as custom object instances queued for processing. Scheduled jobs iterate the received notifications updating the order payment transactions and order status.

Set up a site-level job for each site using the Adyen notifications process and determine the schedule and frequency for each client.

1. For this new job, go to **Administration > Operations > Job Schedules**.
2. Click **New Job**, specify its **ID**, set **Priority** to **Normal**, and then click **Create**.
3. On the **Edit Job** page, switch to the **Step Configurator** tab.

4. Click **Configure a step**, then **ExecuteScriptModule**, then fill in the job details.
For the **Process** step, enter the details as shown below and click **Assign**.

ID*

Process

Description

ExecuteScriptModule.Module*

int_adyen/cartridge/scripts/job/notifications.js

[Global
Parameters](#)

ExecuteScriptModule.FunctionName

processNotifications

[Global
Parameters](#)

☒ ExecuteScriptModule.Transactiona

[Global
Parameters](#)

ExecuteScriptModule.TimeoutInSeconds

3600

[Global
Parameters](#)

☐ Restart Enforced

Custom Parameters

ID*

Value*



Error Handling

On

ERROR




Action

Stop Job



[← Back](#)

[Assign](#)

5. Click the plus icon () to add a new step, then set **ID** to **Clean**, enter the details as shown below, and click **Assign**.

ID*

Clean

Description

ExecuteScriptModule.Module*

int_adyen/cartridge/scripts/job/notifications.js

Global Parameters

ExecuteScriptModule.FunctionName

clearNotifications

Global Parameters

☒ ExecuteScriptModule.Transactional

Global Parameters

ExecuteScriptModule.TimeoutInSeconds

3600

Global Parameters

☐ Restart Enforced

Custom Parameters

ID*

Value*

+

Error Handling

On

ERROR

✎

→

Action

Stop Job

+

← Back

Assign

Customize

To use the Adyen cartridge, you have to add Adyen specific code into the Storefront core code and to the Storefront controller code.

Storefront core changes

Below are the changes that you have to implement in the Storefront core code, for each file. To find these files, go to **app_storefront_core > cartridge**.

forms/default/creditcard.xml

1. Add code under the specified line:

```
<!-- field for credit card owner -->
  <field formid="owner" label="creditcard.ownerlabel" type="string" mandatory="true" max-length="40" binding="creditCardHolder" missing-error="creditcard.ownermissingerror"/>

<!-- ----- ADD THE CODE BELOW ----- -->
  <field formid="encrypteddata" type="string" mandatory="false"/>
```

2. Add code under the specified line:

```
<field formid="saveCard" label="creditcard.savecard" type="boolean" mandatory="false" default-value="true" />

<!-- ----- ADD THE CODE BELOW ----- -->
>
<!-- field for credit card recurring payments / tokenization -->
<field formid="selectedCardID" mandatory="false" type="string"/>
```

js/pages/account.js

1. Require the CSE module:

```
var giftcert = require('../giftcert'),
    tooltip = require('../tooltip'),
    util = require('../util'),
    dialog = require('../dialog'),
    page = require('../page'),
    login = require('../login'),
    validator = require('../validator'),

// ----- ADD THE CODE BELOW -----
adyenCse = require('../adyen-cse');
```


2. Modify the `initializePaymentForm` function:

```
function initializePaymentForm() {
    $('#CreditCardForm').on('click', '.cancel-button', function (e) {
        e.preventDefault();
        dialog.close();
    });

    // ----- ADD THE CODE BELOW -----
    if (SitePreferences.ADYEN_CSE_ENABLED) {
        adyenCse.initAccount();
    }
}
```

`js/pages/checkout/billing.js`

1. Require the CSE module:

```
var ajax = require('../../ajax'),
    formPrepare = require('./formPrepare'),
    giftcard = require('../../giftcard'),
    util = require('../../util'),

    // ----- ADD THE CODE BELOW -----
    adyenCse = require('../../adyen-cse');
```

2. Modify the `setCCFields` function:

```
function setCCFields (data){
    // ...
    $creditCard.find('input[name$="_cvn"]').val('').trigger('change');

    // ----- ADD THE CODE BELOW -----
    $creditCard.find('[name$="creditCard_selectedCardID"]').val(data.selectedCardID).trigger('change');
}
```

3. Add the following functions:

```
/**
 * @function
 * @description Changes the payment type or issuerId of the selected payment
method
 * @param {String, Boolean} value of payment type or issuerId and a test
value to see which one it is, to which the payment type or issuerId should
be changed to
 */
function updatePaymentType(selectedPayType, test) {
  if (!test) {
    $('input[name="brandCode"]').removeAttr('checked');
  } else {
    $('input[name="issuerId"]').removeAttr('checked');
  }
  $('input[value=' + selectedPayType + ']').prop('checked', 'checked');
  formPrepare.validateForm();
}

/**
 * @function
 * @description Adyen - Initializes the visibility of HPP fields
 */
function initializeHPPFields () {
  if ($('[name="brandCode"]:checked').hasClass('openInvoice')) {
    $('.additionalfield').hide().find('input').val('');
    $('.additionalfield.' + $('.checkout-billing').find('select.country').
val()).show();
  } else {
    $('.additionalfield').hide().find('input').val('');
  }
}
```

4. In the exports.init function, add the following variable definitions:

```
exports.init = function () {
  // ...
  var selectedPaymentMethod = $selectPaymentMethod.find(':checked').val();

  // ----- ADD THE CODE BELOW -----
  var $payType = $('[name="brandCode"]');
  var $issuerId = $('[name="issuerId"]');
  var $issuer = $('ul#issuer');
  var selectedPayType = $payType.find(':checked').val();
  var selectedIssuerId = $issuerId.find(':checked').val();
}
```

5. Modify the `exports.init` function:

```
exports.init = function () {
  // ...
  $selectPaymentMethod.on('click', 'input[type="radio"]', function () {
    updatePaymentMethod($(this).val());

    // ----- ADD THE CODE BELOW -----
    if ($(this).val() == 'Adyen' && $payType.length > 0) {
      // set payment type of Adyen to the first one
      updatePaymentType((selectedPayType) ? selectedPayType : $payType[0].
value, false);
    } else {
      $payType.removeAttr('checked');
      $issuerId.removeAttr('checked');
    }
  });

  $issuerId.on('click', function () {
    updatePaymentType($(this).val(), true);
  });
}
```

6. Add code to the `exports.init` function:

```
exports.init = function () {
    // ...
    $giftCertCode.on('keydown', function (e) {
        if (e.which === 13) {
            e.preventDefault();
            $addGiftCert.click();
        }
    });

    // ----- ADD THE CODE BELOW -----
    if (SitePreferences.ADYEN_CSE_ENABLED) {
        adyenCse.initBilling();
    }

    // Adyen - Click event for payment methods
    $payType.on('click', function () {
        updatePaymentType($(this).val(), false);
        //if the payment type contains issuerId fields, expand form with the
values
        if ($(this).siblings('#issuer').length > 0) {
            $issuer.show();
            updatePaymentType((selectedIssuerId) ? selectedIssuerId : $issuerId
[0].value, true);
        } else {
            $issuer.hide();
            $('input[name="issuerId"]').removeAttr('checked');
        }
        initializeHPPFields();
    });

    var currentDate = new Date();
    var currentYear = currentDate.getFullYear();
    var initYear = currentYear - 100;
    $('.openinvoiceInput input[name$="_dob"]').datepicker({
        showOn: 'focus',
        yearRange: initYear + ':' + currentYear,
        changeYear: true
    });
};
```

1. Add Adyen helper:

```
/**
 * Resource helper
 *
 */
//...
var ProductAvailabilityModel = require('dw/catalog/ProductAvailabilityModel')
;

// ----- ADD THE CODE BELOW -----
/* Script Modules */
var AdyenHelper = require ("int_adyen/cartridge/scripts/util/AdyenHelper");
```

2. Add a new validation message:

```
// Validation messages
// ...
VALIDATE_MIN : Resource.msg('validate.min', 'forms', null),

// ----- ADD THE CODE BELOW -----
ADYEN_CC_VALIDATE : Resource.msg('adyen.creditcard', 'adyen', null)
```

3. Modify the ResourceHelper.getPreferences function:

```
ResourceHelper.getPreferences = function(pageContext) {
    var cookieHintAsset = ContentMgr.getContent('cookie_hint');
    return {
        // ...
        CHECK_TLS:Site.getCurrent().getCustomPreferenceValue('checkTLS'),

        // ----- ADD THE CODE BELOW -----
        ADYEN_CSE_ENABLED : AdyenHelper.getAdyenCseEnabled()
```

1. Add code under the specified line:

```
<isscript>
  <!-- ... -->
  var numberAttributes = {
    maxlength: 16,
    size: 17
  };
</isscript>

<!-- ----- ADD THE CODE BELOW ----- -->
<isset name="AdyenHelper" value="{require('int_adyen/cartridge/scripts/util
/AdyenHelper'))}" scope="pdict" />
<isset name="AdyenCseEnabled" value="{pdict.AdyenHelper.
getAdyenCseEnabled()}" scope="page" />
<isif condition="{AdyenCseEnabled}">
<isinclude template="account/payment/adyenpaymentinstrumentdetails"/>
<elseif/>
```

2. Close the tag:

```
<button class="cancel cancel-button simple" type="submit" name="{pdict.
CurrentForms.paymentinstruments.creditcards.cancel.htmlName}" value="{Resour
ce.msg('global.cancel','locale',null)}">
    {Resource.msg('global.cancel','locale',null)}
</button>
</div>

<!-- ----- ADD THE CODE BELOW ----- -->
</isif>
```

Replace the following:

```
<div class="form-row form-row-button">
  <button class="button-fancy-large" type="submit" name="{pdict.CurrentForms.
billing.save.htmlName}" value="{Resource.msg('global.
continueplaceorder','locale',null)}"><span>{Resource.msg('global.
continueplaceorder','locale',null)}</span></button>
</div>
```

with:

```

<div class="form-row form-row-button">
  <isif condition="{(pdict.AdyenHelper && pdict.AdyenHelper.
getAdyenCseEnabled()) || pdict.AdyenCseEnabled === true}">
    <button class="button-fancy-large" type="hidden" id="billing-submit-
hidden" style="display:none" name="{pdict.CurrentForms.billing.save.htmlName}" va
lue="{Resource.msg('global.continueplaceorder','locale',null)}"><span>{Resource.
msg('global.continueplaceorder','locale',null)}</span></button>
    <button class="button-fancy-large" type="submit" id="billing-submit" name
="{pdict.CurrentForms.billing.save.htmlName}" value="{Resource.msg('global.
continueplaceorder','locale',null)}"><span>{Resource.msg('global.
continueplaceorder','locale',null)}</span></button>
  <iselse/>
  <button class="button-fancy-large" type="submit" name="{pdict.
CurrentForms.billing.save.htmlName}" value="{Resource.msg('global.
continueplaceorder','locale',null)}"><span>{Resource.msg('global.
continueplaceorder','locale',null)}</span></button>
</isif>
</div>

```

templates/default/checkout/billing/creditcardjson.isml

Add the following key-value pair:

```

expirationYear:pdict.SelectedCreditCard.creditCardExpirationYear,

<!-- ----- ADD THE CODE BELOW ----- -->
selectedCardID:pdict.SelectedCreditCard.UUID

```

templates/default/checkout/billing/paymentmethods.isml

1. Add code under the specified line:

```

<div class="payment-method <isif condition="{empty(pdict.selectedPaymentID)
|| pdict.selectedPaymentID=='CREDIT_CARD'}">payment-method-expanded</isif> "
data-method="CREDIT_CARD">

<!-- ----- ADD THE CODE BELOW ----- -->
<isset name="AdyenHelper" value="{require('int_adyen/cartridge/scripts/util
/AdyenHelper')}" scope="pdict"/>
<isset name="AdyenCseEnabled" value="{pdict.AdyenHelper.
getAdyenCseEnabled()}" scope="page" />

```


2. Replace the following:

```
<select name="${pdict.CurrentForms.billing.paymentMethods.creditCardList.htmlName}" id="creditCardList" class="input-select">
```

with:

```
<isif condition="${AdyenCseEnabled}">
    <select name="${pdict.CurrentForms.billing.paymentMethods.creditCardList.htmlName}" id="adyenCreditCardList" class="input-select">
<iselse/>
    <select name="${pdict.CurrentForms.billing.paymentMethods.creditCardList.htmlName}" id="creditCardList" class="input-select">
</isif>
```

3. Add code under the specified line:

```
<isprint value="${creditCardInstr.creditCardExpirationMonth}" formatter="00" />
<isprint value="${creditCardInstr.creditCardExpirationYear}" formatter="0000" />
</a>
</isloop>
</iscomment>
</isif>

<!-- ----- ADD THE CODE BELOW ----- -->

<isif condition="${AdyenCseEnabled}">
    <isinclude template="checkout/billing/adyenpaymentmethods"/>
    <isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.selectedCardID}" type="hidden"/>
<iselse/>
```

4. Close the tag:

```
<isinputfield formfield="${pdict.CurrentForms.billing.paymentMethods.creditCard.cvn}" type="input" rowclass="cvn" dynamicname="true" help="${help}" />

<!-- ----- ADD THE CODE BELOW ----- -->
</isif>
```

5. In the Custom processor section, add the Adyen payment method:

```
<iscomment>
    Custom processor
    -----
</iscomment>

<div class="payment-method <isif condition="${!empty(pdict.
selectedPaymentID) && pdict.selectedPaymentID=='PayPal'}">payment-method-
expanded</isif>" data-method="Custom">
    <!-- Your custom payment method implementation goes here. -->
    ${Resource.msg('billing.custompaymentmethod','checkout',null)}
</div>

<!-- ----- ADD THE CODE BELOW ----- -->
<div class="payment-method <isif condition="${!empty(pdict.
selectedPaymentID) && pdict.selectedPaymentID=='Adyen'}">payment-method-
expanded</isif>" data-method="Adyen">
    <isinclude template="hpp"/>
</div>
```

6. To accept payments from POS devices, add code under the specified line:

```
<div class="payment-method <isif condition="${!empty(pdict.
selectedPaymentID) && pdict.selectedPaymentID=='Adyen'}">payment-method-
expanded</isif>" data-method="Adyen">
    <isinclude template="hpp"/>
</div>

<!-- ----- ADD THE CODE BELOW ----- -->
<div class="payment-method <isif condition="${!empty(pdict.
selectedPaymentID) && pdict.selectedPaymentID=='AdyenPOS'}">payment-method-
expanded</isif>" data-method="AdyenPOS">
    <isinclude template="pos"/>
</div>
```

[templates/default/checkout/summary/summary.isml](#)

Add code under the specified line:

```

<input type="hidden" name="{dw.web.CSRFProtection.getTokenName()}" value="{dw.web.CSRFProtection.generateToken()}" />

<!-- ----- ADD THE CODE BELOW ----- -->
<iscomment>
Set the brandcode and issuerId in session for when user hits the back button on
Adyen hpp
</iscomment>
<isif condition="{!empty(pdct.CurrentHttpParameterMap.brandCode.value) || !empty(session.custom.brandCode)}">
    <isset name="brandCode" value="{!empty(pdct.CurrentHttpParameterMap.brandCode.value) ? pdct.CurrentHttpParameterMap.brandCode.value : session.custom.brandCode}" scope="page"/>
    <input type="hidden" name="brandCode" value="{brandCode}" />
    <isif condition="{!empty(pdct.CurrentHttpParameterMap.issuerId.value) || !empty(session.custom.issuerId)}">
        <isset name="issuerId" value="{!empty(pdct.CurrentHttpParameterMap.issuerId.value) ? pdct.CurrentHttpParameterMap.issuerId.value : session.custom.issuerId}" scope="session"/>
        <input type="hidden" name="issuerId" value="{session.custom.issuerId}" />
    </isif>
    <isset name="brandCode" value="{!empty(pdct.CurrentHttpParameterMap.brandCode.value) ? pdct.CurrentHttpParameterMap.brandCode.value : session.custom.brandCode}" scope="session"/>
    <isset name="dob" value="{!empty(pdct.CurrentHttpParameterMap.dob.value) ? pdct.CurrentHttpParameterMap.dob.value : ''}" scope="session"/>
    <isset name="gender" value="{!empty(pdct.CurrentHttpParameterMap.gender.value) ? pdct.CurrentHttpParameterMap.dob.value : ''}" scope="session"/>
</isif>

```

templates/default/components/header/htmlhead.isml

Add code under the specified line:

```

<!-- UI -->
<link rel="stylesheet" href="{URLUtils.staticURL('/css/style.css')}" />

<!-- ----- ADD THE CODE BELOW ----- -->
<link rel="stylesheet" href="{URLUtils.staticURL('/css/checkout.css')}" />

```

Storefront controller changes

If you integrate via the controller method, you have to implement changes in the Storefront controller code. Below are the changes that you have to implement, for each file. To find these files, go to **app_storefront_controllers** > **cartridge**.



If you integrate via the pipeline method, you have to implement changes in the Storefront pipeline. For a description of these changes, [Support Team](#).

1. Add Adyen helpers:

```
/* Script Modules */
var app = require('~cartridge/scripts/app');
var guard = require('~cartridge/scripts/guard');

// ----- ADD THE CODE BELOW -----
var AdyenController = require("int_adyen_controllers/cartridge/controllers/Adyen");
var AdyenHelper = require("int_adyen/cartridge/scripts/util/AdyenHelper");
```

2. In the returnToForm function, add the Adyen helper as a key to app.getView:

```
if (params) {
    app.getView(require('~cartridge/scripts/object').extend(params, {
        Basket: cart.object,
// ----- ADD THE FOLLOWING LINE -----
        AdyenHelper : AdyenHelper,
        ContinueURL: URLUtils.https('COBilling-Billing')
    })).render('checkout/billing/billing');
} else {
    app.getView({
        Basket: cart.object,
// ----- ADD THE FOLLOWING LINE -----
        AdyenHelper : AdyenHelper,
        ContinueURL: URLUtils.https('COBilling-Billing')
    }).render('checkout/billing/billing');
}
```

3. In the publicStart function, remove the following line:

```
start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards});
```

If you are not using POS, replace it with:

```
var AdyenHppPaymentMethods = AdyenController.GetPaymentMethods(cart);

start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards,
    AdyenHppPaymentMethods : AdyenHppPaymentMethods } );
```

If you are using POS, replace it with:

```
var AdyenHppPaymentMethods = AdyenController.GetPaymentMethods(cart);
var AdyenPosTerminals = AdyenController.GetTerminals();
start(cart, {ApplicableCreditCards: creditCardList.ApplicableCreditCards,
    AdyenHppPaymentMethods : AdyenHppPaymentMethods, AdyenPosTerminals :
    AdyenPosTerminals});
```

4. Modify the `validatePayment` function:

```
function validatePayment(cart) {  
  var paymentAmount, countryCode, invalidPaymentInstruments, result;  
  
  // ----- ADD THE CODE BELOW -----  
  if (AdyenHelper.getAdyenCseEnabled()) {  
    result = true;  
    return result;  
  }  
}
```

5. In the `saveCreditCard` function, add code under the specified line:

```
function saveCreditCard() {  
  
  // ----- ADD THE CODE BELOW -----  
  if (AdyenHelper.getAdyenRecurringPaymentsEnabled()) {  
    //saved credit cards are handling in COPlaceOrder and Login for Adyen -  
    saved cards are synced with Adyen ListRecurringDetails API call  
    return true;  
  } else {  
  

```

6. Close the else condition:

```
customer.getProfile().getWallet().removePaymentInstrument  
(creditcard);  
  }  
  }  
});  
}  
return true  
  
// ----- ADD THE CODE BELOW -----  
}
```

controllers/COPlaceOrder.js

1. In the `handlePayments` function, add a new variable definition:

```
function handlePayments(order) {  
  // ...  
  var handlePaymentTransaction = function () {  
    paymentInstrument.getPaymentTransaction().setTransactionID(order.  
getOrderNo());  
  };  
  
  // ----- ADD THE CODE BELOW -----  
  var show3dSecureForm : Boolean = false;  

```

2. In the `handlePayments` function, add code under the specified line:

```
function handlePayments(order) {
    // ...
    if (authorizationResult.not_supported || authorizationResult.error) {
        return {
            error: true
        };
    }

    // ----- ADD THE CODE BELOW -----
    if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).
getPaymentProcessor().ID === 'ADYEN_CREDIT'
        && authorizationResult.authorized3d === true) {
        var show3dSecureForm : Boolean = true;
        var view : Object = authorizationResult.view;
    }
}
```

3. In the `handlePayments` function, add code after the specified line:

```
for (var i = 0; i < paymentInstruments.length; i++) {
    var paymentInstrument = paymentInstruments[i];
    if (PaymentMgr.getPaymentMethod(paymentInstrument.getPaymentMethod()).
getPaymentProcessor() === null) {
        // ...
    } else {
        // ...
    }
}

// ----- ADD THE CODE BELOW -----
if(shown3dSecureForm) {
    return{view : view};
}
```

4. Replace `var handlePaymentsResult = handlePayments(order)` with the following:

```
var skipSubmitOrder : Boolean = false;
var handlePaymentsResult = handlePayments(order);
if (!empty(handlePaymentsResult.view)){
    skipSubmitOrder = true;
}
```

5. Replace `var orderPlacementStatus = Order.submit(order);` with

```
if (order.paymentInstrument.paymentMethod == "Adyen") {
    return {
        Order: order,
        order_created: true
    };
} else {
    if (skipSubmitOrder) {
        return {
            Order: order,
            order_created: true,
            view : handlePaymentsResult.view,
            skipSubmitOrder : skipSubmitOrder
        };
    } else {
        var orderPlacementStatus = Order.submit(order);
    }
}
```

controllers/COSummary.js:

1. At the top of the file, add the following code:

```
var AdyenController = require("int_adyen_controllers/cartridge/controllers/Adyen");
```

2. In the submit function, replace `showConfirmation(placeOrderResult.Order);` with:

```
if (placeOrderResult.Order.paymentInstrument.paymentMethod == "Adyen"){
    AdyenController.Redirect(placeOrderResult.Order);
} else {
    if (placeOrderResult.skipSubmitOrder === true) {
        placeOrderResult.view.render('adyenform');
    } else {
        showConfirmation(placeOrderResult.Order);
    }
}
```

controllers/PaymentInstruments.js

1. At the top of the file, add the following code:

```
/*API includes*/
var PaymentInstrument = require('dw/order/PaymentInstrument');
var Logger = require('dw/system/Logger');
```


2. At the top of the file, add the following code:

```
/*Script Modules*/  
var AdyenHelper = require('int_adyen/cartridge/scripts/util/AdyenHelper');
```

3. Modify the list function:

```
function list() {  
    // ----- ADD THE CODE BELOW -----  
    // Get the Saved Cards from Adyen to get latest saved cards  
    require('int_adyen/cartridge/scripts/UpdateSavedCards').updateSavedCards  
    ({CurrentCustomer : customer});  
}
```

4. Modify the create function:

```
var paymentInstruments = wallet.getPaymentInstruments(dw.order.  
PaymentInstrument.METHOD_CREDIT_CARD);  
  
// ----- ADD THE CODE BELOW -----  
if (AdyenHelper.getAdyenRecurringPaymentsEnabled()) {  
    var createRecurringPaymentAccountResult = AdyenHelper.  
createRecurringPaymentAccount({  
        Customer: customer  
    });  
  
    if (createRecurringPaymentAccountResult.error) {  
        return false;  
    }  
    pspReference = 'PspReference' in createRecurringPaymentAccountResult  
&& !empty(createRecurringPaymentAccountResult.PspReference) ?  
createRecurringPaymentAccountResult.PspReference : '';  
    tokenID = 'TokenID' in createRecurringPaymentAccountResult && !empty  
(createRecurringPaymentAccountResult.TokenID) ?  
createRecurringPaymentAccountResult.TokenID : '';  
    /*if (empty(TokenID) || empty(PspReference)) {  
        return false;  
    }*/  
    try {  
        Transaction.wrap(function() {  
            /* var newCreditCard = customer.getProfile().getWallet().  
createPaymentInstrument(PaymentInstrument.  
METHOD_CREDIT_CARD);  
            * // copy the credit card details to the  
payment instrument  
            * newCreditCard.setCreditCardHolder(  
                newCreditCard.setCreditCardNumber(  
                newCreditCard.setCreditCardType  
(  
                    newCreditCard.setCreditCardToken  
(tokenID);  
                    newCreditCard.custom.AdyenPspReference =  
pspReference; */  
            require('int_adyen/cartridge/scripts/UpdateSavedCards').  
updateSavedCards({  
                CurrentCustomer: customer,  
                PaymentsMap: createRecurringPaymentAccountResult.  
PaymentsMap  
            });  
        });  
    } catch (e) {  
        Logger.error('{0}: {1}', e, e.stack);  
        return false;  
    }  
    return true;  
}
```

5. In the Delete function, replace `wallet.removePaymentInstrument(action.object);` with:

```
var paymentInstrument = action.object;
if (!empty(paymentInstrument)) {
    if (AdyenHelper.getAdyenRecurringPaymentsEnabled() && !empty
(paymentInstrument.getCreditCardToken())) {
        var result = require('int_adyen/cartridge/scripts
/adyenDeleteRecurringPayment').deleteRecurringPayment({
            Customer: customer,
            RecurringDetailReference: paymentInstrument.getCreditCardToken()
        });
        if (result == PIPELET_NEXT) {
            wallet.removePaymentInstrument(paymentInstrument);
        }
    } else {
        wallet.removePaymentInstrument(paymentInstrument);
    }
}
```

6. Modify the `verifyCreditCard` function:

```
function verifyCreditCard() {
    var newCreditCardForm = app.getForm('paymentinstruments.creditcards.
newcreditcard');

    // ----- ADD THE CODE BELOW -----
    if (AdyenHelper.getAdyenCseEnabled()) {
        return true;
    }
}
```

Testing integration

Make sure you have an [Adyen test account](#). Configure your account and then configure the Adyen settings in the [Business Manager](#).

Test credit cards

In case you want to test the Credit Cards payment method, Adyen provides test numbers you can use. For more information, refer to [Test card numbers](#).

Business Manager configuration

Read the [Business Manager configuration](#) section for information about configuring BM.

The **Debug Setting** set to **yes** shows the **Pay** button before performing a redirect to Adyen. This allows you to check the parameters sent to the HPP. Don't change the values of these parameters before sending them, as this will result in an invalid merchant signature for this request.

Adyen Customer Area

1. When you created a new order, it is placed into the Payments list in the [Customer Area](#).
2. When you perform some operations on a payment, the payment status is changed correspondingly. Note that it may take some time for the change to be synchronized from Business Manager to the Adyen CA.

Data storage

For each order that results in a payment attempt on the Adyen HPP, the payment results are available in the following fields:

- PSP reference – a unique Adyen ID of this payment. It is a link between Commerce Cloud order and Adyen payment.
- Payment Method.
- Eventcode – AUTHORISED, CANCELLATION, REFUND, etc.
- Amount – the amount paid.

They are handled by the `Adyen.notify` controller method, which receives the notification messages from Adyen. In one case, when a payment is refused on Adyen HPP, these fields are updated in `Adyen.showConfirmation` controller.

Notifications are queued in Commerce Cloud for a short duration: either until Adyen provides a final status for the order, or the custom object retention period is reached.

Commerce Cloud stores a custom object briefly until a notification has been received from Adyen. As the final status is obtained, the custom object is deleted. If Adyen don't respond to the order with a notification in 72 hours, Commerce Cloud will automatically remove the custom object instance.

All the communication to/from Adyen is logged in the log file directory available via webDAV. Check this log file in case you have any problems.

Also, all the notifications sent by Adyen are logged on the order level and can be found on Notifications tab of Order Details page in BM.

Set up Apple Pay on the Web

Salesforce Commerce Cloud provides integration with [Apple Pay on the Web](#), where Adyen is used as one of the payment facilitators to accept Apple Pay tokens and process them.

Prerequisites

Before proceeding with the steps below, make sure that you registered an Apple Merchant ID and sent it to Adyen.

For more information, see [Enable Apple Pay](#).

Steps

To configure Apple Pay on the Web in Salesforce Commerce Cloud:

1. Go to **Merchant Tools > Site Preferences > Apple Pay**.
2. In the **Payment integration** section, fill in the following items:

Parameter Name	Description
Apple Pay Enabled?	Select the check box.
Apple Merchant ID	Enter your Apple Merchant ID (configurable in https://developer.apple.com/).
Apple Merchant Name	Enter your Apple Merchant Name (configurable in https://developer.apple.com/).
Country Code	Enter your Country Code.
Merchant Capabilities	Select 3DS.
Supported Networks	Select the supported networks.
Redirect Pages to HTTPS?	Select the check box.
Use Commerce Cloud Apple Pay Payment API?	Select the check box.
Payment Provider URL	Use https://pal-test.adyen.com/pal/adapter/Demandware/authorise for Test or https://pal-live.adyen.com/pal/adapter/Demandware/authorise for Live.
Payment Provider Merchant ID	Enter your Adyen Merchant Account name.
API Version	Select the latest.
Use Basic Authentication?	Select the check box.
Payment Provider User	Your Adyen web service (WS) user name.
Payment Provider Password	Your Adyen web service (WS) user password.
Use JWS?	Leave it unselected.
JWS Private Key Alias	Leave it empty.

3. Register Apple Sandbox/Production domain on the **Domain Registration** section of Salesforce Commerce Cloud.
4. Go to **Merchant Tools > Ordering > Payment Methods > DW_APPLE_PAY** and set the **Payment Processor** to ADYEN_CREDIT.

For any questions regarding this integration, contact Salesforce Commerce Cloud directly.

Set up Terminal API

Adyen's Terminal API allows your Salesforce Commerce Cloud stores to process payments via point-of-sale devices.

Before you begin

- Make sure that you have [imported the metadata](#).
- Log in to the Business Manager.

Step 1: Enable POS payments

1. Go to **Merchant Tools > Ordering > Payment methods**.
2. Click **AdyenPOS**.
3. Select **Yes** in the **Enabled** column.
4. In the **AdyenPOS Details** section, from the **Payment Processor** drop-down, select **Adyen_POS**.

Step 2: Configure services

1. Go to **Administration > Operations > Services**.
2. In the **Profile** column next to **AdyenPosPayment**, click **Adyen**.
3. In the **Name** box, enter **Adyen**.
4. In the **Connection Timeout (ms)** box, enter **100.000**.
5. Ensure that the **Enable Circuit Breaker** check box is not selected.

Step 3: Configure a terminal

1. Go to **Merchant Tools > Site Preferences > Custom Preferences**.
2. Click **Adyen**.
3. In the **Adyen POS unique terminal ID** box, enter the ID of your Adyen POS terminal. You can find this in the Adyen [Customer Area](#), under **Point of Sale > Terminal Fleet Manager**.
4. In the **X-API-KEY of Web service** box, enter your Checkout API key. You can find this in the Adyen [Customer Area](#), under **Settings > Users > [your web service user] > Checkout API Key**.

Step 4 (optional): Configure multiple terminals

To configure multiple terminals:

1. Go to **Merchant Tools > Site Preferences > Custom Preferences**.
2. Click **Adyen**.
3. In the **Terminal configuration JSON** box, enter JSON objects for the different terminals. Use the following key-value pairs:

Key	Value
POIID	The unique ID of the terminal. You can find this in the Adyen Customer Area , under Point of Sale > Terminal Fleet Manager > Unique Terminal Id .
Description	Description of the terminal.
Store	The name of the store for this terminal. You can specify your own store names.



Make sure that you only change the values, and keep the keys (POIID, Description, Store) as they are.

Code sample

```
[
  {
    "POIID" : "P400-1234",
    "Description" : "Terminal description",
    "Store" : "YourStore1"
  },
  {
    "POIID" : "P400-5678",
    "Description" : "Terminal description",
    "Store" : "YourStore2"
  }
]
```