Authorize.Net
a CyberSource solution

Direct Post Method (DPM) Developer Guide

Card Not Present Transactions

Authorize.Net Developer Support
http://developer.authorize.net

Authorize.Net LLC 3/30/11 Ver. Ver 1.1

Authorize.Net Trademarks

Revision History

PUBLISH DATE	UPDATES
10/22/2010	Release of Ver 1.0 Direct Post Method (DPM) Developer Guide
10/29/2010	Minor formatting changes and additional clarifying content in Sections 1 and 5.
November 2010	Corrected sample code for checkout_form.jsp Minor edits to language
January 2011	Code corrections

Table of Contents

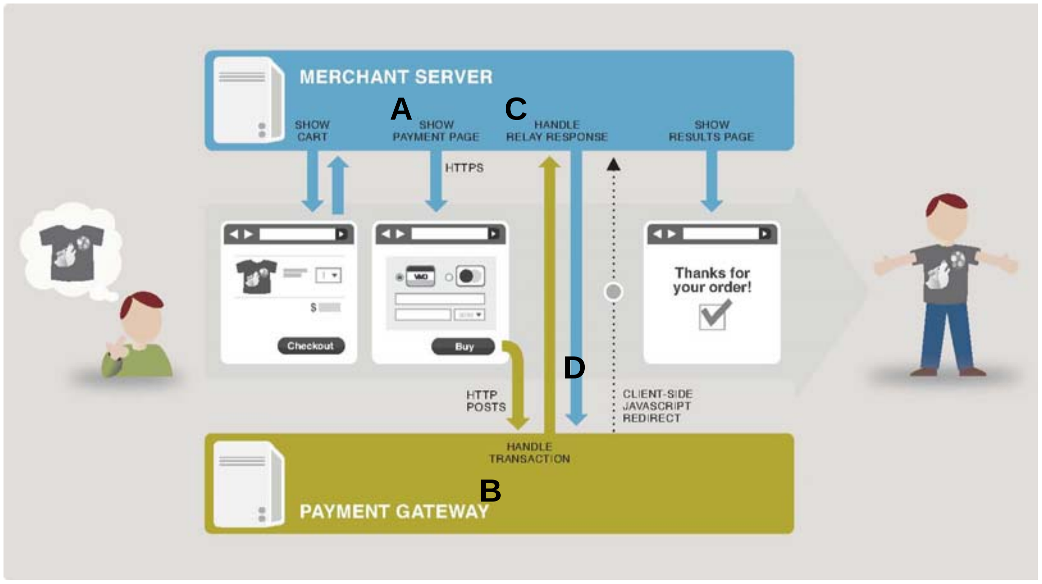
Chapter 1

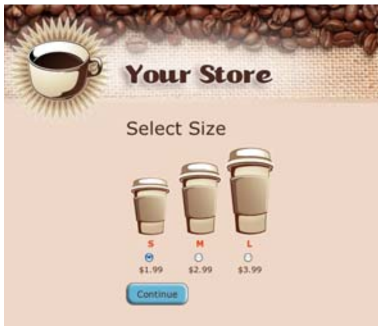
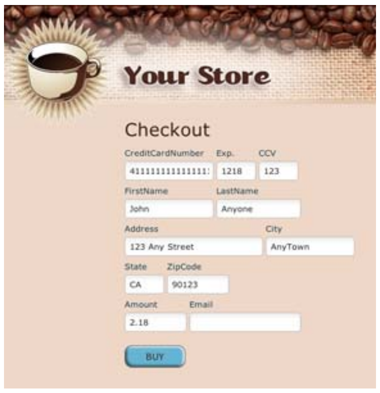
Conceptual Overview

Using the Direct Post Method (DPM), merchants post transactions directly to Authorize.Net, bypassing the merchant server. Upon authorization, Authorize.Net returns coded values to the merchant server for verification, then generates a receipt for display within the merchant page. Customer billing data posts directly to Authorize.Net without touching the merchant server, while the merchant still retains control over the checkout process.

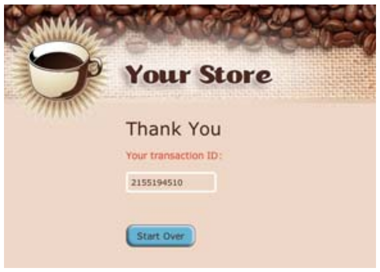
The `relay URL` in DPM is the location where Authorize.Net sends a POST of the transaction result. Instead of returning content for display in the client browser, DPM works by returning a snipped generated by the merchant's relay URL to the merchant server, which redirects the client browser to the merchant's server. This way, the URL in the client browser is pointed to the merchant's server. This uses JavaScript, if available on the client browser, and a `meta refresh` tag if it is not.

The following diagram (and the table that follows) provide an annotated illustration of the architecture and flow of control of a sample DPM transaction:



LABEL	USER EXPERIENCE	DESCRIPTION
A	 Your Store Select Size S M L $1.99 $2.99 $3.99 Continue	When a customer submits a shopping cart, the merchant server displays the form (for example, `checkout_form.jsp`) that sets the Direct Post Method into motion.
B	 Your Store Checkout CreditCardNumber: 41111111111111111 Exp.: 1218 CCV: 123 FirstName: John LastName: Anyone Address: 123 Any Street City: AnyTown State: CA ZipCode: 90123 Amount: 2.18 Email: BUY	On **Submit,** customer billing data posts directly to Authorize.Net*,* bypassing the merchant server. This post can include both hidden (merchant-supplied) and customer data fields. The developer can use any of the DPM Form Fields*,* and can create Merchant-Defined Data Fields*.* DPM Form and Merchant-Defined Data Fields on page 21 identifies and defines common form fields and specifies the constraints on merchant-defined fields*.* The posted form includes an *`x_relay_url`* value (containing the URL to which Authorize.Net will post transaction results upon completion). This is not a typical relay URL*;* that is, it contains no content for display in the client browser. Instead, it returns a snippet that redirects the client browser to a URL on the merchant's server. This maintains the merchant server's URL in the client browser address bar throughout the transaction.

LABEL	USER EXPERIENCE	DESCRIPTION

C/D	 Your Store
	Thank You
	Your transaction ID:
	2155194510
	Start Over
		When the merchant server receives the HTTP POST from Authorize.Net, it runs validation (comparing hash values) and logs the order.
		
		Authorize.Net expects a return from this HTTP POST and displays the result:
		• on failure (an HTTP response code other than `200-OK`), an error message
		• on success, the Authorize.Net server returns the snippet generated by the merchant's relay url that forces the redirection of the client browser to the merchant server. This uses JavaScript if available on the client browser, and a *meta refresh* tag if it is not. For example:
		`<html>`
		`  <head>`
		`     <script type='text/javascript' charset='utf-8'>`
		`      window.location='http://`**`YOUR_SERVER`**`.COM/receipt.jsp';`
		`      </script>`
		`      <noscript>`
		`      <meta http-equiv='refresh' content='1;url=http://YOURSERVER.COM/receipt.jsp'>`
		`      </noscript>`
		`   </head>`
		`   <body></body>`
		`</html>`
		
		This keeps the URL in the client browser pointed to the merchant's server. (The redirect should also contain enough information about the transaction so that the merchant's server can display something sensible to the user.)

Please note that the fields documented in the following sections describe only the minimum required fields. If you wish to submit additional fields (for example, such as billing address), refer to the documentation for the Server Integration Method (SIM).

Chapter 2

Integrating with Java

The example Java integration of the Direct Post Method implementation described in this Section uses three code snippets:

- checkout_form.jsp,
- `relay_response.jsp`, and
- order_receipt.jsp

These snippets perform the sequence of functions described in **Section 1, "Conceptual Overview"**.

Note The following steps assume that your web server is on the public internet and can be accessed by means of a domain name or IP address.

The following steps create the three snippets.

1 Obtain an API Login ID and Transaction Key.

These keys authenticate requests to the payment gateway. To obtain them, sign up for a test account.

2 Install the Authorize.Net Java SDK.

This gives you access to the full suite of APIs.

3 Download the Java SDK from the Developer Center:

`developer.authorize.net/downloads`

Place `authnet-sdk-java.jar` (found in `target`) and the `jar` files found in `/lib` into your *classpath.*

4 Create `checkout_form.jsp.`

The following provides an example of fully functional `checkout_form.jsp` code. If you use this code as a template, be sure to populate the three variables `API_LOGIN_ID`, `TRANSACTION_KEY`, and `MERCHANT_HOST`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html lang='en'>
<head>
<title>testing</title>
</head>
<body>

<%@ page import="net.authorize.sim.*"%>
<%
String apiLoginId = "API_LOGIN_ID";
String transactionKey = "TRANSACTION_KEY";
String relayResponseUrl = "http://MERCHANT_HOST/relay_response.jsp";
String amount = "1.99";
Fingerprint fingerprint = Fingerprint.createFingerprint(apiLoginId,
transactionKey,
1234567890, // random sequence used for the fingerprint amount);
long x_fp_sequence = fingerprint.getSequence();
long x_fp_timestamp = fingerprint.getTimeStamp();
String x_fp_hash = fingerprint.getFingerprintHash();
%>

<form id='secure_redirect_form_id' action='https://test.authorize.net/
gateway/transact.dll' method='POST'>
<label for='x_card_num'>Credit Card Number</label>
<input type='text' class='text' id='x_card_num' name='x_card_num'
size='20' maxlength='16' />
<br />
<label for='x_exp_date'>Expiration Date</label>
<input type='text' class='text' id='x_exp_date' name='x_exp_date' size='6'
maxlength='6'/>
<br />
<label for='x_amount'>Amount</label>
<input type='text' class='text' id='x_amount' name='x_amount' size='10'
maxlength='10' readonly='readonly' value='<%=amount%>' />
<br />
<input type='hidden' name='x_invoice_num'
value='<%=System.currentTimeMillis()%>' />
<input type='hidden' name='x_relay_url' value='<%=relayResponseUrl%>' />
<input type='hidden' name='x_login' value='<%=apiLoginId%>' />
<input type='hidden' name='x_fp_sequence' value='<%=x_fp_sequence%>' />
<input type='hidden' name='x_fp_timestamp' value='<%=x_fp_timestamp%>' />
<input type='hidden' name='x_fp_hash' value='<%=x_fp_hash%>' />
<input type='hidden' name='x_version' value='3.1' />
```

```
<input type='hidden' name='x_method' value='CC' />
<input type='hidden' name='x_type' value='AUTH_CAPTURE' />
<input type='hidden' name='x_amount' value='<%=amount%>' />
<input type='hidden' name='x_test_request' value='FALSE' />
<input type='hidden' name='notes' value='extra hot please' />
<input type='submit' name='buy_button' value='BUY' />
</form>

</body>
</html>
```

Note To place the `jsp` in a separate webapp container of your choosing, modify `relayResponseUrl` accordingly.

5 Create `relay_response.jsp.`

The following provides an example of fully functional `relay_response.jsp` code. If you use this code as a template, be sure to populate the two variables (indicated in **BOLD CAPS**):

Note Unless you have explicitly set MD5-Hash in the merchant interface (using **Account** > **Settings** > **Security Settings** > **MD5-Hash**), leave this as an empty string.

```
<%@ page import="java.util.Map"%>
<%@ page import="net.authorize.*"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
var referrer = document.referrer;
if (referrer.substr(0,7)=="http://") referrer = referrer.substr(7);
if (referrer.substr(0,8)=="https://") referrer = referrer.substr(8);
if(referrer && referrer.indexOf(document.location.hostname) != 0) {
<%
  String apiLoginId = "API_LOGIN_ID";
  String receiptPageUrl = "http://MERCHANT_HOST/order_receipt.jsp";
  /*
   * Leave the MD5HashKey as is - empty string, unless you have explicitly
   *  set it in the merchant interface:
   * Account > Settings > Security Settings > MD5-Hash
   */
  String MD5HashKey = "";
  net.authorize.sim.Result result =
    net.authorize.sim.Result.createResult(apiLoginId, MD5HashKey,
      request.getParameterMap());
  // perform Java server side processing...
  // ...
  // build receipt url buffer
StringBuffer receiptUrlBuffer = new StringBuffer(receiptPageUrl);
if(result != null) {
  receiptUrlBuffer.append("?");
  receiptUrlBuffer.append(
    ResponseField.RESPONSE_CODE.getFieldName()).append("=");
  receiptUrlBuffer.append(result.getResponseCode().getCode());
  receiptUrlBuffer.append("&");
  receiptUrlBuffer.append(
    ResponseField.RESPONSE_REASON_CODE.getFieldName()).append("=");
  receiptUrlBuffer.append(
    result.getReasonResponseCode().getResponseReasonCode());
  receiptUrlBuffer.append("&");
  receiptUrlBuffer.append(
    ResponseField.RESPONSE_REASON_TEXT.getFieldName()).append("=");
  receiptUrlBuffer.append(
  result.getResponseMap().get(
    ResponseField.RESPONSE_REASON_TEXT.getFieldName()));

  if(result.isApproved()) {
    receiptUrlBuffer.append("&").append(
  ResponseField.TRANSACTION_ID.getFieldName()).append("=");
    receiptUrlBuffer.append(result.getResponseMap().get(
  ResponseField.TRANSACTION_ID.getFieldName()));
  }
}
%>
// Use Javascript to redirect the page to the receipt redirect url.
// If Javascript is not available, then the <meta> refresh tag
// will handle the redirect.
document.location = "<%=receiptUrlBuffer.toString()%>";
}
//-->
</script>
<noscript><meta http-equiv="refresh"
content="0;url=<%=receiptUrlBuffer.toString()%>"></noscript>
</body>
</html>
```

6 Create `order_receipt.jsp.`

This file contains receipt information that will display to the customer. It should be accessible at the ***`receipt_page_url`*** location specified in the previous step. The following provides an example of fully functional `order_receipt.jsp` code:

```
<%@ page import="java.util.Map" %>
<%@ page import="net.authorize.*" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
</head>
<body>
<h1>Your Receipt Page</h1></br>
<%
// Show the confirmation data
Map<String, String[]> requestParameterMap = request.getParameterMap();
if(requestParameterMap != null &&  requestParameterMap.containsKey(
  ResponseField.RESPONSE_CODE.getFieldName())) {
  String transactionId = "";
  if(requestParameterMap.containsKey(
    ResponseField.TRANSACTION_ID.getFieldName())) {

    transactionId = requestParameterMap.get(
      ResponseField.TRANSACTION_ID.getFieldName())[0];
}
// 1 means we have a successful transaction
if("1".equals(requestParameterMap.get(
  ResponseField.RESPONSE_CODE.getFieldName())[0])) {
%>
<h2>Success!</h2>
<h3>Your transaction ID:</h3>
<div><%=net.authorize.util.StringUtils.sanitizeString(transactionId)%>
</div>
<%
} else {
%>
<h2>Error!</h2>
<h3><%=net.authorize.util.StringUtils.sanitizeString(
requestParameterMap.get(
  ResponseField.RESPONSE_REASON_TEXT.getFieldName())[0])%>
</h3>
<table>
<tr>
<td>response code</td>
<td><%=net.authorize.util.StringUtils.sanitizeString(
requestParameterMap.get(ResponseField.RESPONSE_CODE.getFieldName())[0])%>
</td>
</tr>
<tr>
<td>response reason code</td>
<td><%=net.authorize.util.StringUtils.sanitizeString(
requestParameterMap.get(ResponseField.RESPONSE_REASON_
CODE.getFieldName())[0])%>
</td>
</tr>
</table>
</div>
<%
  }
}
%>
</body>
</html>
```

7 Check your work.

`checkout_form.jsp, relay_response.jsp,` and `receipt_page.jsp` must be publicly accessible on your web server and all paths contained in such variables match actual file locations.

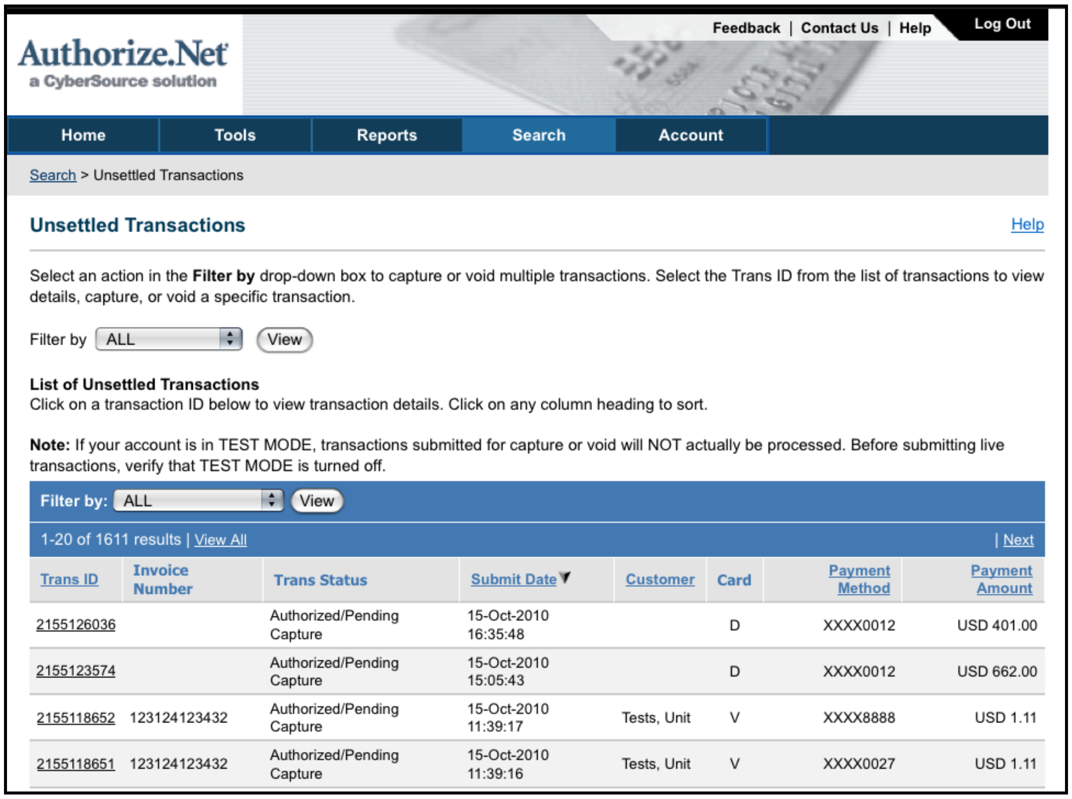
8 Verify your Direct Post Method implementation.

From your test environment,

```
http://MERCHANT_HOST/checkout_form.jsp
... enter credit card 4111111111111111, any future expiration
date (in the form MMDD; for example, 1120), then Submit.
```

Note Authorize.Net requires port 80 for comunication on the relay_response endpoint.

You should receive a success (receipt) or error message. You can also verify payment transaction success on your transaction report:



Search > Unsettled Transactions

Unsettled Transactions

Help

Select an action in the **Filter by** drop-down box to capture or void multiple transactions. Select the Trans ID from the list of transactions to view details, capture, or void a specific transaction.

Filter by ALL View

List of Unsettled Transactions
Click on a transaction ID below to view transaction details. Click on any column heading to sort.

Note: If your account is in TEST MODE, transactions submitted for capture or void will NOT actually be processed. Before submitting live transactions, verify that TEST MODE is turned off.

Filter by: ALL View

1-20 of 1611 results | View All | Next

Trans ID	Invoice Number	Trans Status	Submit Date	Customer	Card	Payment Method	Payment Amount
2155126036		Authorized/Pending Capture	15-Oct-2010 16:35:48		D	XXXX0012	USD 401.00
2155123574		Authorized/Pending Capture	15-Oct-2010 15:05:43		D	XXXX0012	USD 662.00
2155118652	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:17	Tests, Unit	V	XXXX8888	USD 1.11
2155118651	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:16	Tests, Unit	V	XXXX0027	USD 1.11

If you receive an error, check the syntax and accuracy of the variables you entered into the code snippets during steps 3-5, and also verify that all paths contained in such variables match actual file locations.

Chapter 3

Integrating with PHP

The example PHP integration of the Direct Post Method implementation described in this Section uses three code snippets:

- checkout_form.php,
- `relay_response.php`, and
- order_receipt.php

These snippets perform the sequence of functions described in **Section 1, "Conceptual Overview"**.

Note The following steps assume that your web server is on the public internet and can be accessed by means of a domain name or IP address.

The following steps create the three snippets.

1 Obtain an API Login ID and Transaction Key.

These keys authenticate requests to the payment gateway. To obtain them, sign up for a test account.

2 Download the Authorize.Net PHP SDK and include it in your project.

This gives you access to the full suite of APIs.

3 Download the PHP SDK from the Developer Center.

`developer.authorize.net/downloads`

4 Save the SDK into a folder that your web server can access (for example, `/var/www` or `/htdocs`).

5 Include `anet_php_sdk/AuthorizeNet.php` in your project.

6 Create `checkout_form.php.`

The following provides an example of fully functional `checkout_form.php` code. If you use this code as a template, be sure to populate the three variables (indicated in **BOLD CAPS**):

```
<?php require_once 'anet_php_sdk/AuthorizeNet.php'; // The SDK
$relay_response_url = "http://YOUR_DOMAIN.com/relay_
response.php"; // You will create this file in step 7.
$api_login_id = 'YOUR_API_LOGIN_ID';
$transaction_key = 'YOUR_TRANSACTION_KEY';
$amount = "5.99";
$fp_sequence = "123"; // Any sequential number like an invoice
number.
echo AuthorizeNetDPM::getCreditCardForm($amount, $fp_sequence,
$relay_response_url,$api_login_id, $transaction_key);?>
```

Note To place the php in a separate webapp container of your choosing, modify ***relayResponseUrl*** accordingly.

7 Create `relay_response.php.`

The following provides an example of fully functional `relay_response.php` code. If you use this code as a template, be sure to populate the two variables (indicated in **BOLD CAPS**):

Note Unless you have explicitly set MD5-Hash in the merchant interface (using **Account** > **Settings** > **Security Settings** > **MD5-Hash**), leave this as an empty string.

```
<?php require_once 'anet_php_sdk/AuthorizeNet.php'; // The SDK
$redirect_url = "http://YOUR_DOMAIN.com/receipt_page.php"; //
Where the user
will end up.
$api_login_id = 'YOUR_API_LOGIN_ID';
$md5_setting = ""; // Your MD5 Setting
$response = new AuthorizeNetSIM($api_login_id, $md5_setting);
if ($response->isAuthorizeNet())
{
if ($response->approved)
   {
       // Do your processing here.
       $redirect_url .= '?response_code=1&transaction_id=' .
       $response->transaction_id;
   }
   else
   {
       $redirect_url .= '?response_code='.$response->response_
code .
'&response_reason_text=' . $response->response_reason_text;
   }
   // Send the Javascript back to AuthorizeNet, which will
redirect user back to
your site.
   echo AuthorizeNetDPM::getRelayResponseSnippet($redirect_
url);
}
else
{
echo "Error. Check your MD5 Setting.";
}?>
```

8 Create `order_receipt.php.`

This file contains receipt information that will display to the customer. It should be accessible at the ***`redirect_url`*** location specified in the previous step.

The following provides an example of fully functional `order_receipt.php` code:

```
<?php
if ($_GET['response_code'] == 1)
{
echo "Thank you for your purchase! Transaction id: "
  .htmlentities($_GET['transaction_id']);}
else
{
echo "Sorry, an error occurred: " . htmlentities($_
GET['response_reason_text']);
}?>
```

9 Check your work.

`checkout_form.php, relay_response.php,` and `receipt_page.php` must be publicly accessible on your web server and all paths contained in such variables match actual file locations.

10 Verify your Direct Post Method implementation.

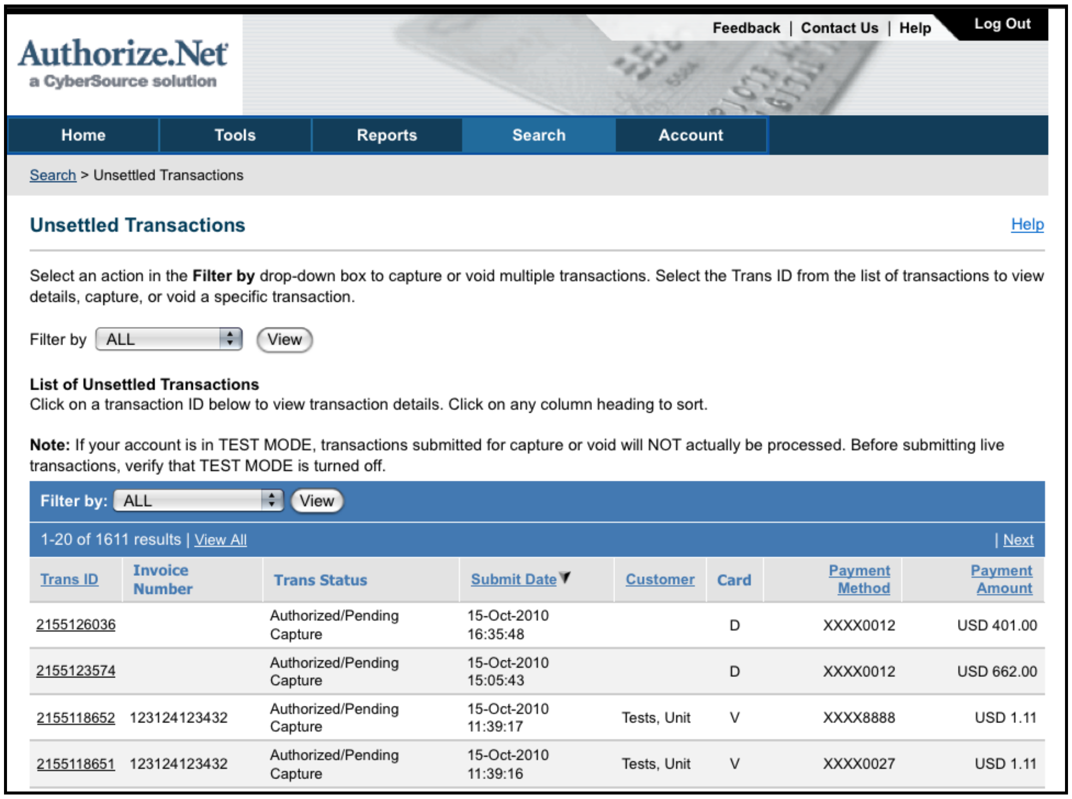
From your test environment:

`http://`**`MERCHANT_HOST`**`/checkout_form.php`

Enter credit card *4111111111111111*, any future expiration date (in the form `MMDD;` for example, *1120*), then **Submit.**

Note Authorize.Net requires port 80 for comunication on the relay_response endpoint.

You should receive a success (receipt) or error message. You can also verify payment transaction success on your transaction report:



Search > Unsettled Transactions

Unsettled Transactions Help

Select an action in the **Filter by** drop-down box to capture or void multiple transactions. Select the Trans ID from the list of transactions to view details, capture, or void a specific transaction.

Filter by ALL View

List of Unsettled Transactions
Click on a transaction ID below to view transaction details. Click on any column heading to sort.

Note: If your account is in TEST MODE, transactions submitted for capture or void will NOT actually be processed. Before submitting live transactions, verify that TEST MODE is turned off.

Filter by: ALL View

1-20 of 1611 results | View All | Next

Trans ID	Invoice Number	Trans Status	Submit Date	Customer	Card	Payment Method	Payment Amount
2155126036		Authorized/Pending Capture	15-Oct-2010 16:35:48		D	XXXX0012	USD 401.00
2155123574		Authorized/Pending Capture	15-Oct-2010 15:05:43		D	XXXX0012	USD 662.00
2155118652	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:17	Tests, Unit	V	XXXX8888	USD 1.11
2155118651	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:16	Tests, Unit	V	XXXX0027	USD 1.11

Note If you receive an error, check the syntax and accuracy of the variables you entered into the code snippets during steps 3-5, and also verify that all paths contained in such variables match actual file locations.

Chapter 4

Integrating with Ruby

The example Ruby integration of the Direct Post Method implementation described in this Section uses three code snippets:

- payment.erb,
- payments_controller.rb, and
- receipt.erb

These snippets perform the sequence of functions described in **Section 1, "Conceptual Overview"**.

Note The following steps assume that your web server is on the public internet and can be accessed by means of a domain name or IP address.

The following steps create the three snippets.

1. Obtain an API Login ID and Transaction Key.

   These keys authenticate requests to the payment gateway. To obtain them, sign up for a test account.
2. Install the Authorize.Net SDK.

   This gives you access to the full suite of APIs.

   Enter the following from the command line; substitute the appropriate version number for the x's:

   ```
   sudo gem install authorize-net-1.x.x.gem
   ```
3. For Ruby on Rails version 2.x, run the script from the command line. For Ruby on Rails version 3, skip to step 4.

   From the command line, with the items in **BOLD CAPS** replaced with their corresponding values from the local merchant environment, run:

```
sudo gem install rails -v '~> 2.1'
rails my_direct_post_app
```

```
cd my_direct_post_app
script/generate authorize_net_direct_post payments \
    YOUR_API_LOGIN_ID YOUR_TRANSACTION_KEY YOUR_API_LOGIN_ID
script/server
```

This process generates examples of the three required files, populated with local merchant values. If you use this code as a template, be sure to populate the variables (indicated in **BOLD CAPS**).

4 For Ruby on Rails version 2.x, skip to step 5. For Ruby on Rails version 3, do the following:

From the command line run:

```
sudo gem install rails
rails new my_direct_post_app
cd my_direct_post_app
```

Open the file "Gemfile" in the root directory of you new ruby app and add the following line to the end of the file:

```
gem 'authorize-net'
```

Back at the command line, run:

```
rails generate authorize_net:direct_post payments YOUR_API_ LOGIN_
ID YOUR_TRANSACTION_KEY YOUR_API_LOGIN_ID
rails server
```

5 Verify your Direct Post Method implementation.

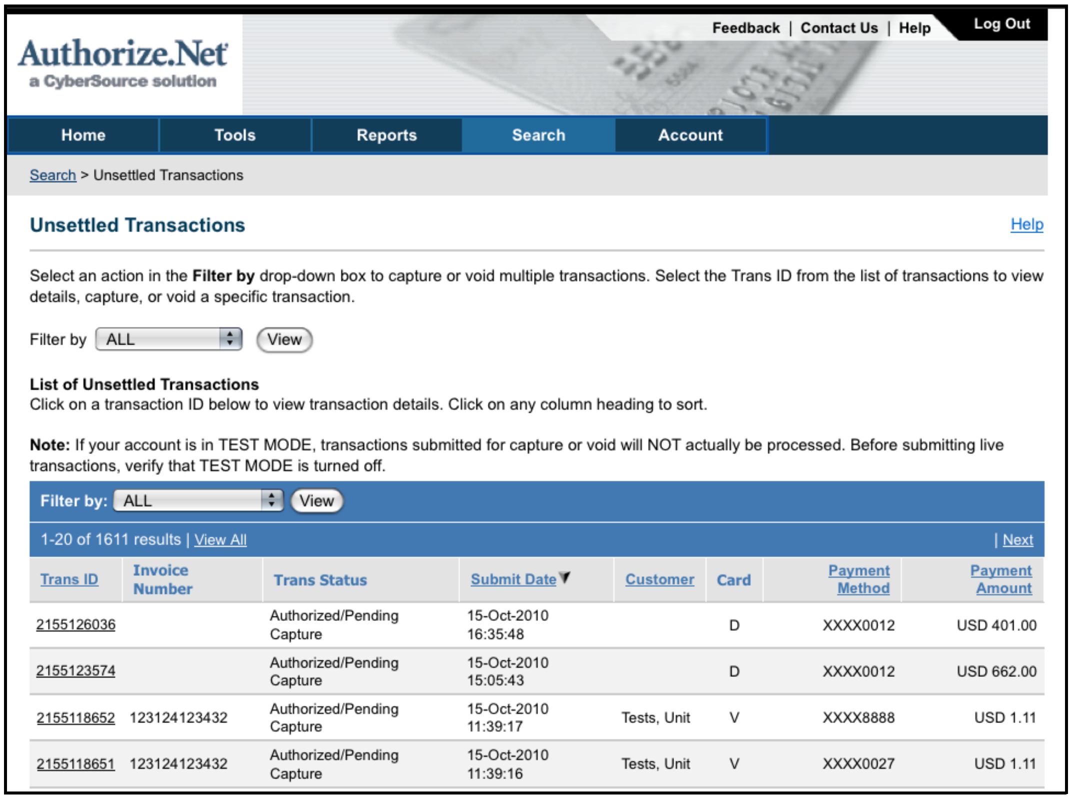
From your test environment:

`http://`**`MERCHANT_HOST`**`/payments/payment`

Enter credit card *4111111111111111*, any future expiration date (in the form `MMYY;` for example, *1112*), then **Submit.**

Note Authorize.Net requires port 80 for comunication on the relay_response endpoint.

You should receive a success (receipt) or error message. You can also verify payment transaction success on your transaction report:



Search > Unsettled Transactions

Unsettled Transactions

Help

Select an action in the **Filter by** drop-down box to capture or void multiple transactions. Select the Trans ID from the list of transactions to view details, capture, or void a specific transaction.

Filter by ALL View

List of Unsettled Transactions
Click on a transaction ID below to view transaction details. Click on any column heading to sort.

Note: If your account is in TEST MODE, transactions submitted for capture or void will NOT actually be processed. Before submitting live transactions, verify that TEST MODE is turned off.

Filter by: ALL View

1-20 of 1611 results | View All | Next

Trans ID	Invoice Number	Trans Status	Submit Date	Customer	Card	Payment Method	Payment Amount
2155126036		Authorized/Pending Capture	15-Oct-2010 16:35:48		D	XXXX0012	USD 401.00
2155123574		Authorized/Pending Capture	15-Oct-2010 15:05:43		D	XXXX0012	USD 662.00
2155118652	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:17	Tests, Unit	V	XXXX8888	USD 1.11
2155118651	123124123432	Authorized/Pending Capture	15-Oct-2010 11:39:16	Tests, Unit	V	XXXX0027	USD 1.11

Note If you receive an error, check the syntax and accuracy of the variables you entered into the code snippets during step 3, and also verify that all paths contained in such variables match actual file locations.

Chapter 5

Integrating with .NET

The example .NET integration of the Direct Post Method implementation assumes that your web server is on the public internet and can be accessed by means of a domain name or IP address:

Note This SDK supports .NET 3.5, not .NET 4.0.

1 Download the Authorize.Net C# SDK and include it in your project.

2 Download the SDK and unzip to your hard drive.

3 Create a new ASP.NET MVC application and add a reference to the `AuthorizeNET.dll` from the SDK.

4 Add the `AuthorizeNET.Helpers` namespace to `web.config.`

Under `system.web/pages`:

```
<namespaces>
        <add namespace="System.Web.Mvc"/>
    ...
        <add namespace="AuthorizeNet.Helpers"/>
</namespaces>
```

5 Create a checkout form that posts directly to *Authorize.Net.*

You can add the following code to the `Home/Index View` to create a form that submits a test transaction to Authorize.Net (with hardcoded values). This code uses the SDK's **CheckoutFormBuilder** to create the form for you; just include ***YOUR_SERVER***, ***YOUR_API_LOGIN*** and ***YOUR_TRANSACTION_KEY***:

```
<h1><%=ViewData["message"] %></h1>
    <%using (Html.BeginSIMForm("http://YOUR_SERVER.com/home/sim",
1.99M,"YOUR_API_LOGIN","YOUR_TRANSACTION_KEY",true)){%>
    <%=Html.CheckoutFormInputs(true)%>
    <%=Html.Hidden("order_id","1234") %>
    <input type = "submit" value = "Pay" />
    <%}%>
```

Note In this example code, the CheckoutFormBuilder creates the form that will post to Authorize.Net, setting the *`TestMode`* to *`true`*. It also passes an ***`order id`*** (not required by the API, but enables order identification on receipt of the response).

6 Create an action to handle the response from Authorize.Net.

This example uses ASP.NET MVC*;* if using ASP.NET WebForms, you can put this code on `PageLoad().`

The above code passed in the URL `http://YOUR_SERVER.com/home/sim,` so an Action called *Sim* must be created in the application's HomeController. The form POST from Authorize.net will include all of the transaction information, including an MD5 hash for validation of the post’s origin.

By appending your ***`MERCHANT_HASH_CODE`*** and ***`YOUR_API_LOGIN`*** to the following code and pasting it into the HomeController, you can create an example of an action to handle the response:

```
[AcceptVerbs(HttpVerbs.Post)]
 public ActionResult Sim(FormCollection post) {
 var response = new AuthorizeNET.SIMResponse(post);
  //first order of business - validate that it was Auth.net that
posted this using
  //the MD5 hash that was passed back to us
  var isValid = response.Validate("YOUR_MERCHANT_HASH_CODE",
"YOUR_API_LOGIN");
  //if it's not valid - just send them to the home page. Don't
throw - that's how
  //hackers figure out what's wrong :)
  if (!isValid)
   return Redirect("/");
   //the URL to redirect to- this MUST be absolute
   var returnUrl = "http://YOUR_SERVER.com/
?m="+response.Message;
   return
Content(AuthorizeNET.Helpers.CheckoutFormBuilders.Redirecter(re
turnUrl));
  }
```

Note The example code uses SDK classes to parse the response and validate its origin using the MD5 hash. (You can find your ***`MERCHANT_HASH_CODE`*** in your merchant profile on Authorize.Net*;* if you created your developer account on the new developer center, your ***`MERCHANT_HASH_CODE`*** is initially blank.)

Instead of returning content for display in the client browser, DPM works by returning a snippet generated by the merchant's relay URL to the merchant server. This snippet redirects the client browser to the merchant's server. This uses JavaScript, if available on the client browser, and a `meta refresh` tag if it is not. In the last line of the above code, the SDK generates this JavaScript. This keeps the URL in the client browser pointed to the merchant's server.

The `redirecter` call in the example code calls SDK helper code that creates the JavaScript snippet that does the redirect. A real implementation will contain additional parameters specifying what to display to the user on the final page. (At a minimum, by passing the ***`TRANSACTION_ID`*** and the ***`TRANSACTION_CODE`***, you can send a message to the user with information about the order and whether the transaction went through.)

For simplicity, the example code redirects back to the home page of the application and displays the result. The whole process occurs out of session*;* that is, the Authorize.Net server notifies the merchant's server of the transaction, without awareness of the current user's session.

Appendix A

DPM Form and Merchant-Defined Data Fields

This appendix describes common DPM form fields and also provides guidance regarding restrictions on Merchant-Defined Data Fields.

The following table describes common DPM form fields:

field	description
x_invoice_num	Merchant-assigned invoice number, for merchant reference.
x_relay_url	Not a typical relay URL; that is, the user will not be redirected to this URL. It exists only to tell *Authorize.Net* where to send an HTTP POST containing transaction details.
x_login	Merchant's API login.
x_fp_sequence	A merchant-defined field. Should be unique per transaction.
x_fp_timestamp	Form generation timestamp.
x_fp_hash	A value calculated by the SDK helper function (an MD5 hash of merchant *API login id, sequence #, timestamp,* and *amount*).
x_version	*Authorize.Net* API version (currently 3.1).
x_method	Payment method (CC or ECHECK)
x_type	Transaction type (AUTH_CAPTURE *or* AUTH_ONLY; developers will usually want to use AUTH_CAPTURE)
x_amount	Amount of transaction.
x_test_request	TRUE *or* FALSE.

The developer can add any other desired fields (called *Merchant Defined Data Fields*), which pass through to the *relay;* for example, in the Java example, the “notes” field can contain special shipping instructions or other information.

Warning Merchant-Defined Data fields are not intended, and **MUST NOT** be used, to capture personally identifying information. Accordingly, Merchant is prohibited from capturing, obtaining, and/or transmitting any personally identifying information in or by means of the Merchant-Defined Data fields. Personally identifying information includes, but is not limited to: *n*ame, address, credit card number, social security number, driver's license number, state-issued identification number, passport number, and card verification numbers (CVV, CVC2, CVV2, CID, CVN). In the event that Authorize.Net, a CyberSource solution, discovers that Merchant is capturing and/or transmitting personally identifying information by means of the Merchant-Defined Data fields, whether or not intentionally, CyberSource **WILL** immediately suspend Merchant’s account, which will result in a rejection of any and all transaction requests submitted by Merchant after the point of suspension.