



# **Merchant Web Application Integration Quick Start Guide**

## **Payeezy ACH API**

# Contents

- Section 1: Introduction ..... 3
- Section 2: General Workflow ..... 4
- Section 3: Integration Details - Web Application ..... 5
- Section 4: Validation Call Type Example - PayWithMyBank ..... 7
  - 1. Method 1: An example Spring Boot / JSP Web Application ..... 7
  - 2. Major components of sample Spring Boot / JSP web application ..... 16
  - 3. Building and deployment of the application ..... 17
  - 4. Validation Call Methods for PayWithMyBank. .... 18
- Troubleshooting ..... 19
  - 1. Compatibility with Internet Explorer 8 ..... 19
  - 2. Mismatched URL Hash Validation ..... 19
  - 3. Merchant Boarding ..... 20
- Appendix ..... 21
  - 1. PayWithMyBank Return Status Values ..... 21
  - 2. PayWithMyBank Transaction Test Passwords ..... 22
- Revision History ..... 25

# Section 1: Introduction

This Quick Start Integration guide provides an overview of the steps required to integrate a Merchant Web Application (that can handle a JavaScript call) with Payeezy ACH API (Application Program Interface).

The Payeezy ACH API provides the controls and functions needed to validate the Merchant in the TeleCheck system, retrieve the information needed to establish the call to the various Account Validation Partners, and retrieve the requested information from the Account Provider Partner for the Merchant’s Web Application. Payeezy ACH API will track the validation request in an internal database.

For this release, we have PayWithMyBank as the available Account Validation Partner (AVP) for this service. A sample web application utilizing PayWithMyBank is available for download here: <http://github.com/payeezy>

The Account Validation Partner requested is specified through the “Validation Call Type” sent to Payeezy ACH API.

Validation Call Type	Account Provider Partner	Call Type Methods
Pwmb	Pay With My Bank	pwmb.getAll pwmb.getConsumerInfo pwmb.getBankInfo
<FutureProviderPartners>	<FutureProviderPartners>	<FutureProviderPartners>

To use the Payeezy ACH API within the Merchant Web Application, the following is required:

- 1) The Merchant Web Application must be able to support javascript.
- 2) Network Firewalls must be configured to connect to the Payeezy URL.
- 3) Network Firewalls must be configured to connect to the Account Validation Partner URL.
- 4) Must obtain the “apikey”, “token”, and “apisecret” from Payeezy to be able to execute any API calls from the Payeezy URL. (Development/Testing values are provided as part of the git project)

## Section 2: General Workflow

The diagram below depicts the general workflow interactions between each component required to use the Payeezy ACH API in the Merchant Web Application.

The diagram references PayWithMyBank as the Account Validation Partner for this example.

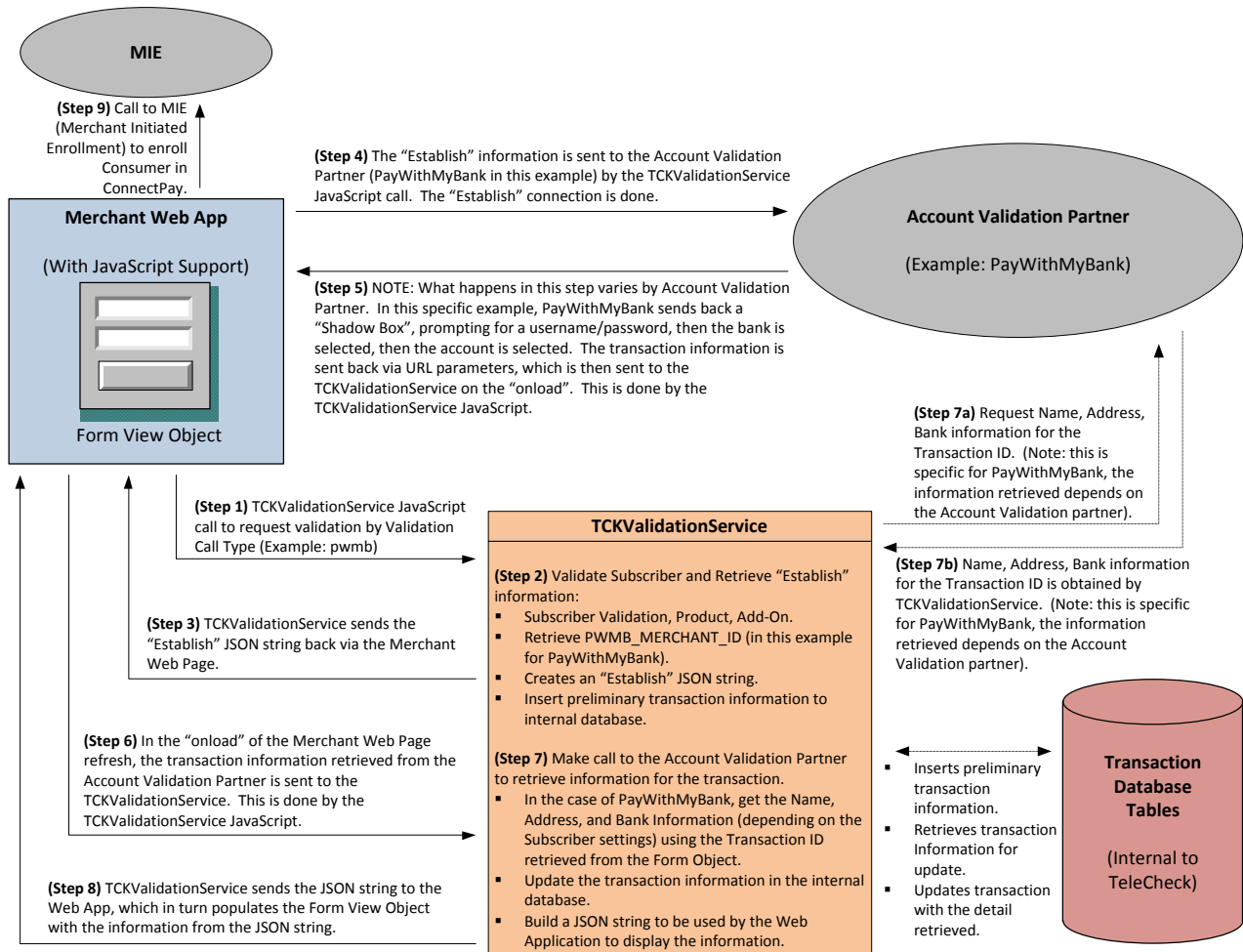


Figure 2.1: General Workflow Diagram of Merchant web app.

## Section 3: Integration Details - Web Application

The Merchant Web Application needs to include the following:

- 1) The following java packages located in the git project referenced above. (Detailed contents described on pg. 16)  
**com.firstdata.ach.pwmb.web.client**  
**com.firstdata.ach.pwmb.web.client.payload**  
**com.firstdata.ach.pwmb.web.client.util**
- 2) The **application.properties** file, located in the git project. (Detailed contents described on pg. 17)
- 3) The JavaScript for the interface (hosted locally), located in the git project:

```
<script type="text/javascript"  
src="https://<MerchantDomainHere>/TCKValidationInterface.js"></script>
```

Figure 3.1: Inclusion of TCKValidationInterface JavaScript inside of a Merchant Web Application.

- 4) The JavaScript and Cascading Style Sheet for the Account Validation Partner must be included in the Merchant Web Application (For PayWithMyBank, these are hosted remotely on PayWithMyBank's server).

This is an example for using PayWithMyBank as the Account Validation Partner:

(Note that the values below are for development/testing purposes. Production values will be provided upon Merchant boarding)

```
<link href='https://sandbox.paywithmybank.com/start/styles/pwmb.css' rel='stylesheet'  
type='text/css'>  
  
<script type="text/javascript"  
src="https://sandbox.paywithmybank.com/start/scripts/pwmb/pwmb.js?accessId=RqBNyqzqT  
VGhmvyV74NM"></script>
```

Figure 3.2: Inclusion of JavaScript and Cascading Style Sheet from an Account Validation Partner (PayWithMyBank).

- 5) A call to Payeezy ACH Establish API via the TCKValidationInterface JavaScript must be added to the Merchant Web Application to initiate the "Establish" request, Subscriber Validation, establish connection to the Account Validation Partner, information retrieval, and display.

This is all done in one call to TCKValidationService Javascript.

This example (included in the git project) shows the call to TCKValidationService by clicking on a button on the screen:

```
<button name="pwmb" onclick="javascript:TCKValidationService('pwmb.getAll')">Enroll using PWMB</button><br><br>
```

Figure 3.3: Form Button to initiate "Establish" call to TCKValidationService.

This table shows the parameters for the call to TCKValidationService (in this example we are using PayWithMyBank as the Validation Call Type):

Parameter Number	Parameter Name	Option(s) / Value(s)	Description
P1	Validation Call Type Options	pwmb.getAll pwmb.getConsumerInfo pwmb.getBankInfo	This tells TckValidationService which Account Validation Partner (AVP) will be used for the request, and the type of information that will be requested from the AVP.

- 6) A Form View to retrieve and contain the information retrieved from the Account Validation Partner.

Fields are populated automatically by naming the fields with the exact field object name as provided for the Validation Call Type. The following is an example for PayWithMyBank:

```
<!-- Field Extraction Method: name fields exactly as below, fields will be populated automatically -->
<tr><td>Bank RTN:</td><td><input type='text' id='bank_routing' /></td></tr>
<tr><td>Bank Account:</td><td><input type='text' id='bank_acct' /></td></tr>
<tr><td>Transaction ID:</td><td><input type='text' id='transactionID' /></td></tr>
<tr><td>Customer Name:</td><td><input type='text' id='username' /></td></tr>
<tr><td>Address:</td><td><input type='text' id='address' /></td></tr>
<tr><td>Address2:</td><td><input type='text' id='address2' /></td></tr>
<tr><td>City:</td><td><input type='text' id='city' /></td></tr>
<tr><td>State:</td><td><input type='text' id='state' /></td></tr>
<tr><td>Zip:</td><td><input type='text' id='zip' /></td></tr>
<tr><td>Phone:</td><td><input type='text' id='phone' /></td></tr>
<tr><td>Email:</td><td><input type='text' id='email' /></td></tr>
```

Figure 3.4: Sample code for Web Form Objects which will be populated automatically by TCKValidationService after transaction information is retrieved from Account Validation Partner (PayWithMyBank).

In the above example, by naming the field "bank\_routing", the call to TckValidationService will look for a form object called "bank\_routing" and automatically fill it with the Bank Routing retrieved from PayWithMyBank, and also look for a form object called "bank\_acct" and automatically fill it with the Bank Account retrieved from PayWithMyBank, etc.

## Section 4: Validation Call Type Example - PayWithMyBank

This section is a specific example showing integration with PayWithMyBank as the Account Validation Partner.

The example in this section was tested in the following configurations:

Configuration Number	Configuration Details
1	Java Version: 1.8 Servlet API Version: 3 Application Server: Cloud Foundry
2	Java Version: 1.8.0_73 Servlet API Version: 2.5 Application Server: Apache Tomcat 6.0.45

### 1. Method 1: An example Spring Boot / JSP Web Application

This is an example with a Spring Boot /JSP web page accessing TckValidationService. (<http://github.com/payeezy>)

- Code Structure:

```
<Web Application Root>/  
├── pwmb_merchant.jsp  
    └── transactions.jsp
```

- File Descriptions:
  - *pwmb\_merchant.jsp*: The Web Application Form Page; includes the references to PayWithMyBank sandbox JavaScript and Cascading Style Sheet, TckValidationInterface JavaScript for accessing TckValidationService, and the Form Objects to display the values of the information retrieved from PayWithMyBank.
  - *Transactions.jsp*: Displays the output of transaction call.

#### A. Create a sample Merchant Web Application in JSP containing:

- Interface to TckValidationService JavaScript.
- Interface to Payeezy ACH API calls.
- Call to TckValidationService to validate subscriber, establish connection to PayWithMyBank, and retrieve the selected Account Information.
- Display the Account Information in a Form View Object.

The following are highlights of the file: *pwmb\_merchant.jsp*.

- i. Interface to TckValidationService (via the TckValidationInterface JavaScript).

This code shows the inclusion of the TckValidationInterface JavaScript containing the method(s) to access the TckValidationService API. Note that “name” attribute would be passed from App Controller (more details below in Section B).

```
<script type="text/javascript"
  src="<%= request.getAttribute("name")%>/TCKValidationInterface.js">
</script>
```

Figure 4.1: Section of code in “pmwb\_merchant.jsp” showing inclusion of the TckValidationInterface Javascript.

- ii. Interface to PayWithMyBank sandbox.

This code shows the inclusion of PayWithMyBank’s cascading style sheet and the inclusion of PWMB’s JavaScript to run their “Shadow box”:

```
<link href='https://sandbox.paywithmybank.com/start/styles/pwmb.css'
  rel='stylesheet' type='text/css'>
<script type="text/javascript"
src="https://sandbox.paywithmybank.com/start/scripts/pwmb/pwmb.js?accessId=RqBNyqzgTVGhmvy
V74NM">
</script>
```

Figure 4.2: Section of code in “pmwb\_merchant.jsp” showing of PayWithMyBank Javascript and CSS.

- iii. Call to TckValidationService to validate subscriber, establish connection to PayWithMyBank, and retrieve the selected Account Information.

In this section of code, a button is clicked to initiate the call to TckValidationService. The available methods for the PWMB Validation Call Type are: pwmb.getAll, pwmb.getConsumerInfo, pwmb.getBankInfo.

```
<!-- Pl options: pwmb.getAll, pwmb.getConsumerInfo, pwmb.getBankInfo -->
<button name="pwmb" onclick="javascript:TCKValidationService('pwmb.getAll')">Enroll using
PWMB</button><br><br>
```

Figure 4.3: Section of code in “pmwb\_merchant.jsp” showing button to initiate call to TckValidationService.

Once the account information from PayWithWithBank is retrieved, the information can be displayed in a Form Object in the Web Application. Below is the method for displaying the information.

- iv. Automatically display the Account Information in a Form View Object.

This method requires that the Form Objects are named specifically as shown. TCKValidationService will automatically fill the values of these Form Objects once the data is retrieved from PayWithMyBank.

```
<table>
  <!-- Field Extraction Method 1: name fields exactly as below,
  fields will be populated automatically -->
  <tr><td>Transaction ID:</td><td><input type='text' id='transactionID' /></td></tr>
  <tr><td>Customer Name:</td><td><input type='text' id='username' /></td></tr>
  <tr><td>Bank RTN:</td><td><input type='text' id='bank_routing' /></td></tr>
  <tr><td>Bank Account:</td><td><input type='text' id='bank_acct' /></td></tr>
  <tr><td>Address:</td><td><input type='text' id='address' /></td></tr>
```



```

<tr><td>Address2:</td><td><input type='text' id='address2' /></td></tr>
<tr><td>City:</td><td><input type='text' id='city' /></td></tr>
<tr><td>State:</td><td><input type='text' id='state' /></td></tr>
<tr><td>Zip:</td><td><input type='text' id='zip' /></td></tr>
<tr><td>Phone:</td><td><input type='text' id='phone' /></td></tr>
<tr><td>Email:</td><td><input type='text' id='email' /></td></tr>
</table>
<br><br>
<table>

```

Figure 4.4: Section of code in “pmwb\_merchant.jsp” showing method to auto-populate values retrieved by TCKValidationService.

- v. Perform enrollment once transaction ID is obtained

This method is used by Web Application to pull individual fields as needed. First, a JavaScript function is added to the Web Application.

```

function Enroll()
{
    var obj = new Object();
    obj.transactionId = document.getElementById("transactionID").value;
    obj.name = document.getElementById("username").value;
    obj.bank_rt = document.getElementById("bank_routing").value;
    obj.bank_acct = document.getElementById("bank_acct").value;
    obj.city = document.getElementById("city").value;
    obj.state = document.getElementById("state").value;
    obj.zip = document.getElementById("zip").value;
    obj.phone = document.getElementById("phone").value;
    obj.email = document.getElementById("email").value;
    obj.address1 = document.getElementById("address").value;
    obj.address2 = document.getElementById("address2").value;

    var data = JSON.stringify(obj);

    // alert(data);

    xmlhttp_enroll.send(data);
}

```

Figure 4.5: Section of code in “pmwb\_merchant.jsp” showing JavaScript method to manually populate values retrieved by TCKValidationService.

## B. Implement sample “pmwb\_merchant.jsp” in your Web Application Container and make it accessible through a URL.

On this example, we will place “pmwb\_merchant.jsp” in a Web Application Container in your local PC as a “PWMBMethod1” Web Application.

Here we are implementing a Controller with **@RequestMapping("/")** that forwards the request to “pmwb\_merchant.jsp”. The Controller can also automatically determine if request is coming from Cloud Foundry and sets “name” field appropriately.

After deployment in a Web Application Container in your local PC, invoke the application through a Web Browser.

**https://localhost:8443/pwmb/**

**PWMB Test Merchant**

**Pay with my bank validation call response**

Transaction ID:

Customer Name:

Bank RTN:

Bank Account:

Address:

Address2:

City:

State:

Zip:

Phone:

Email:

**Perform transaction (need enrollment id first)**

Enrollment ID:

Enter amount:

Figure 4.6: How "pwmb\_merchant.jsp" looks like in a web browser.

- C. Clicking on the button "Enroll using PWMB" will initiate the call to TckValidationService to validate the merchant, and establish the connection to PayWithMyBank (the Account Validation Partner).

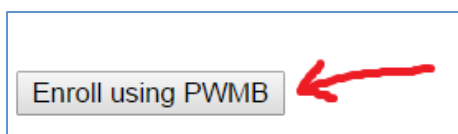


Figure 4.7: Button to click to initiate call to TckValidationService.

This executes the call in this line of “pwmb\_merchant.jsp”:

```
<!-- P1 options: pwmb.getAll, pwmb.getConsumerInfo, pwmb.getBankInfo -->  
<button name="pwmb" onclick="javascript:TCKValidationService('pwmb.getAll')">Enroll using  
PWMB</button><br><br>
```

Figure 4.8: Line of code that the button executes.

This will call TCKValidationController on within sample application. TCKValidationController will do the following:

- Retrieve the Payeezy “apikey”, “token” and “secret” credentials from “application.properties”, which are required in order to use the Payeezy API.
- Create an “Establish” JSON string to be used by the Merchant Web Page to establish the connection to PayWithMyBank.
- Make API call to Payeezy ACH Establish API call
  - <https://developer.payeezy.com/payeezy-api/apis/post/ach/establish>
  - This will internally call Telecheck service that will insert preliminary transaction information to the internal database used by TCKValidationService, to keep track of the transaction request.

TCKValidationController then sends the “Establish” JSON string back via the Merchant Web Page. This “Establish” information is sent to the Account Validation Partner (PayWithMyBank in this example) by the TCKValidationService JavaScript call, and the “Establish” connection is done.

PayWithMyBank then sends back a “Shadow Box”, where the user selects the bank, prompts for a username/password for PayWithMyBank, and then the account is selected.

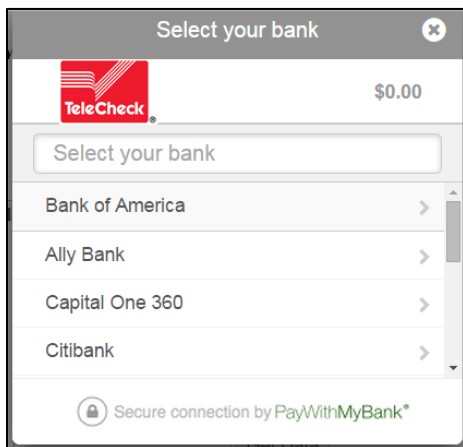
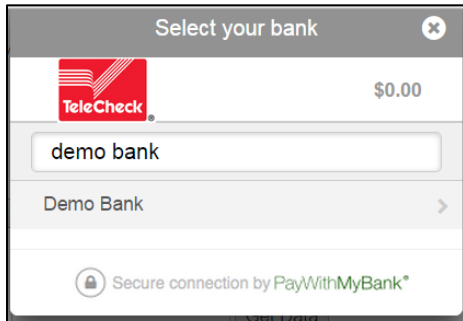


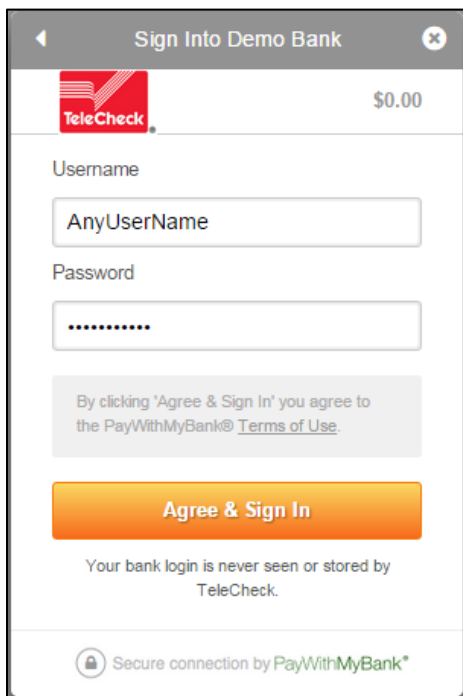
Figure 4.9: PayWithMyBank’s “Shadow Box” where the user selects the bank.

For development, PayWithMyBank has setup a sandbox with “Demo Bank” with a sample Checking and Savings account.



**Figure 4.10:** User can either scroll down or search for “Demo Bank”.

In this sandbox for “Demo Bank”, there is no username and password setup. The user just needs to enter something in those fields and click on “Agree & Send”.



**Figure 4.11:** A value is entered in the username and password fields, and “Agree & Sign In” is clicked.

Two available demo bank accounts are available for information retrieval, a Checking Account and a Savings Account. In this example, we will have the user select the Checking Account, and click on “Continue” to retrieve the account information.

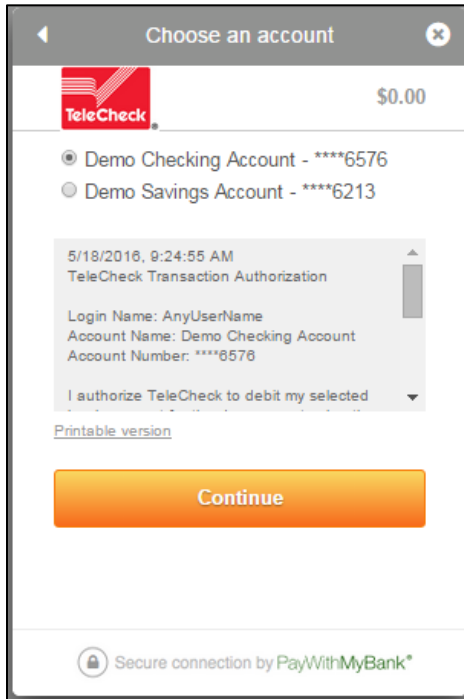


Figure 4.12: Demo Checking Account is selected and “Continue” button is clicked.

The transaction information from PayWithMyBank is sent back via URL parameters. On the “onload” of the Merchant Web Page refresh, this transaction information is sent to the TCKValidationService Javascript.

TCKValidationService Javascript then makes a call to PayWithMyBank to retrieve the detailed account information for the transaction.

- This calls Payeezy ACH API Validate:
  - <https://developer.payeezy.com/payeezy-api/apis/post/ach/validate>
  - In the case of PayWithMyBank, TCKValidationService retrieves the Name, Address, and Bank Information (depending on the Subscriber settings) using the Transaction ID retrieved from the Form Object.
  - That updates the transaction information in the internal database with the additional details about the transaction.
- TCKValidationController then builds a JSON string to be used by the Web Application to display the information.

TCKValidationController sends the JSON string to the Web App, which in turn populates the Form View Object with the information from the JSON string.

If the Merchant Web Application has the Form Objects named as follows ...

```

<table>
  <!-- Field Extraction Method 1: name fields exactly as below, fields will be populated
  automatically -->
  <tr><td>Transaction ID:</td><td><input type='text' id='transactionID' /></td></tr>
  <tr><td>Customer Name:</td><td><input type='text' id='username' /></td></tr>
  <tr><td>Bank RTN:</td><td><input type='text' id='bank_routing' /></td></tr>

```

```

<tr><td>Bank Account:</td><td><input type='text' id='bank_acct' /></td></tr>
<tr><td>Address:</td><td><input type='text' id='address' /></td></tr>
<tr><td>Address2:</td><td><input type='text' id='address2' /></td></tr>
<tr><td>City:</td><td><input type='text' id='city' /></td></tr>
<tr><td>State:</td><td><input type='text' id='state' /></td></tr>
<tr><td>Zip:</td><td><input type='text' id='zip' /></td></tr>
<tr><td>Phone:</td><td><input type='text' id='phone' /></td></tr>
<tr><td>Email:</td><td><input type='text' id='email' /></td></tr>
</table>
<br><br>
<table>

```

Figure 4.13: Section of code showing specific Form Object names to be auto-populated with the transaction information retrieved.

... TCKValidationService will auto-populate the values retrieved from PayWithMyBank.

### PWMB Test Merchant

### Pay with my bank validation call response

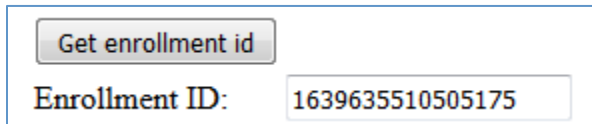
Transaction ID:	<input type="text" value="1001366632"/>
Customer Name:	<input type="text" value="John Smith"/>
Bank RTN:	<input type="text" value="*****3116"/>
Bank Account:	<input type="text" value="*****6576"/>
Address:	<input type="text" value="2000 Broadway Street"/>
Address2:	<input type="text"/>
City:	<input type="text" value="Redwood City"/>
State:	<input type="text" value="CA"/>
Zip:	<input type="text" value="94063"/>
Phone:	<input type="text" value="2145553434"/>
Email:	<input type="text" value="jsmith@email.com"/>

Figure 4.14: TCKValidationService auto-populates values retrieved from PayWithMyBank.

In the Merchant Web Application, the user clicks the “Get Enrollment ID” button so that PWMB enrollment ID is obtained using Payeezy ACH Enrollment / PWMB call:

<https://developer.payeezy.com/payeezy-api/apis/post/ach/consumer/enrollment/pwmb>

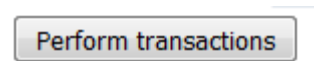
This is done via TCKValidationController's enroll method.



The image shows a rectangular form with a light blue border. At the top, there is a button with the text "Get enrollment id". Below the button, on the left, is the label "Enrollment ID:" followed by a text input field containing the alphanumeric string "1639635510505175".

*Figure 4.15: "Get Enrollment ID" button is clicked.*

Once that's done, one can enter amount and click "Perform Transactions".



The image shows a single button with the text "Perform transactions" centered on it.

*Figure 4.16: "Perform Transactions" button.*

This will execute:

<https://developer.payeezy.com/payeezy-api/apis/post/transactions-15>

This should return successful transaction response:

```
{"correlation_id":"55.1481653456458","transaction_status":"approved","validation_status":"success","transaction_type":"purchase","transaction_id":"140032000000007307304","method":"ach","amount":"10","currency":"USD","gateway_resp_code":"07","gateway_message":"3250"}
```

*Figure 4.17: Example transaction response*

## 2. Major components of sample Spring Boot / JSP web application

- Code Structure:

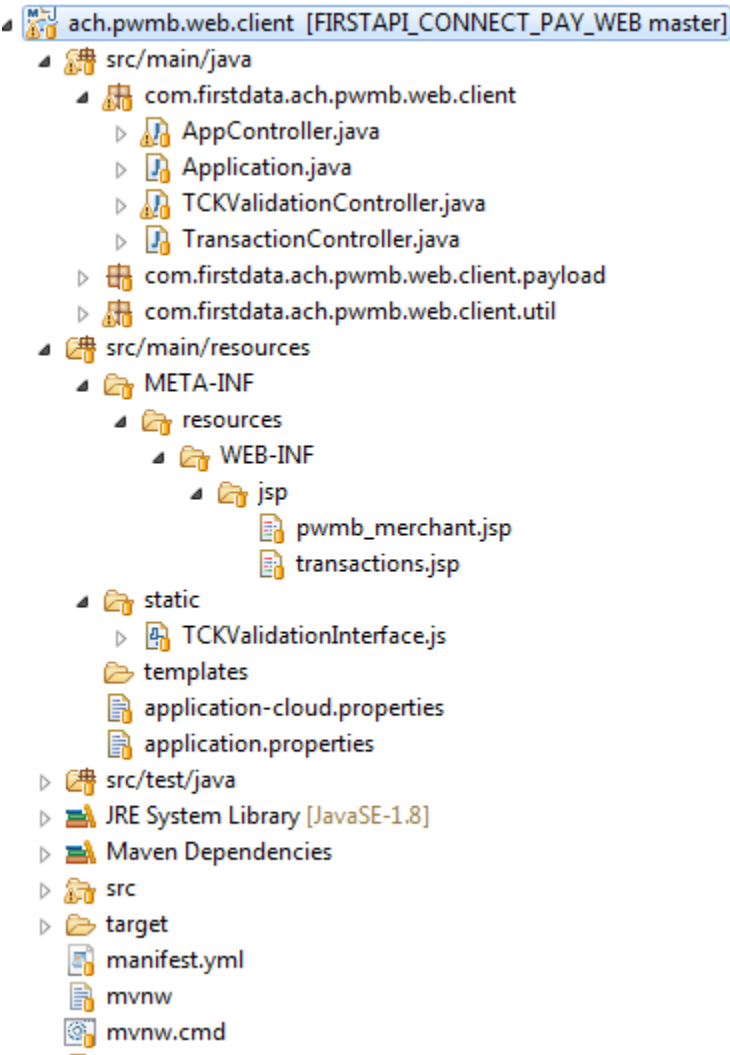


Figure 4.18: Code Structure for Sample Application.

Full Code repository is available at: <http://github.com/payeezy>

- File Descriptions:

<i>pwmb_merchant.jsp</i>	The Main Web Application Form Page. Includes the references to PayWithMyBank sandbox JavaScript and Cascading Style Sheet, the Form Action call to a <i>PWMBServlet</i> that specifically serves the requests from this Form Page, and the Form Objects to display the values of the information retrieved from PayWithMyBank.
<i>transactions.jsp</i>	Displays output of transactions call.
<i>TCKValidationInterface.js</i>	JavaScript that fills the Form Objects with the values retrieved from



	PayWithMyBank by TckValidationService and makes API calls to Payeezy ACH services (establish/validate/enroll).
<i>Application.java</i>	Entry point for Spring Boot application. Initializes the application
<i>AppController.java</i>	Initial controller that sets server name argument and forwards to pwmb_merchant.jsp page.
<i>TCKValidationController.java</i>	Provides methods for calling “establish”, “validate” and “enroll”.
<i>TransactionController.java</i>	Execute Payeezy API transaction call.
<i>Payload package</i>	Provides sample Java objects for constructing payload for Payeezy APIs.
<i>Util package</i>	Provides sample Java objects for Hmac utility validation.

- **application.properties**

Contains the “apikey”, “token”, and “secret” credentials to allow use of the Payeezy API calls.

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp

# Parameters required to make API calls with Payeezy
# These are associated with particular enrollment subscriber ID and auth subscriber ID
apikey=y6pWAJNyJyjGv66IsVuWnklkKUPFbb0a
token=fdoa-a480ce8951daa73262734cf102641994c1e55e7cdf4c02b6
apisecret=2b940ece234ee38131e70cc617aa2afa3d7ff8508856917958e7feb3ef190447
payeezy.url=https://api-cert.payeezy.com/v1/ach

# Needed in case one tries to execute calls internally within FDC network
useProxy=true
proxy.server=fdcproxy.1dc.com
proxy.port=80
```

Figure 4.19: Contents of “application.properties”.

- **TCKValidationInterface.js**

This JavaScript fills the Form Objects with the values retrieved from PayWithMyBank by TckValidationService and makes API calls to Payeezy ACH services for “establish”, “validate”, “enroll”.

### 3. Building and deployment of the application

- To build the application you will need to use “Maven” by Apache. It is a build automation tool used primarily for Java projects. Once you have Maven, open a command line and simply execute:

> **mvn clean install**

This will create an executable JAR file, “ach.pwmb.web.client-0.0.1-SNAPSHOT.jar”

- This Spring Boot application can be executed from command line simply by using:

> **java -jar ach.pwmb.web.client-0.0.1-SNAPSHOT.jar**

It can also be deployed to any compatible J2EE application server (such as Tomcat) and or Cloud Foundry.

## 4. Validation Call Methods for PayWithMyBank.

TckValidationService has three available call methods for PayWithMyBank, which is passed as Parameter 1 as shown in this code:

```
<!-- P1 options: pwmb.getAll, pwmb.getConsumerInfo, pwmb.getBankInfo -->  
<button name="pwmb" onclick="javascript:TCKValidationService('pwmb.getAll')">Enroll using  
PWMB</button><br><br>
```

Figure 4.20: Section of code in "pwmb\_merchant.jsp" showing the call to TCKValidationService using the CallType method "pwmb.getAll".

Call Type Methods	Fields Retrieved
pwmb.getConsumerInfo	transactionID username name phone email address1 address2 city state email
pwmb.getBankInfo	transactionID bank_routing bank_acct
pwmb.getAll	All fields from both getConsumerInfo and BankInfo

# Troubleshooting

## 1. Compatibility with Internet Explorer 8

“TCKValidationInterface.js” uses the FormData object to build payloads, which is not supported by Internet Explorer 8 (according to Microsoft documentation, this is supported with Internet Explorer 10 or later). This works successfully in Firefox 41.0.1 and Chrome 44.0.2403.155.

If you use it in Internet Explorer 8, you will get an error.

## 2. Mismatched URL Hash Validation

When using the TCKValidationService in your Web Application, care must be taken so that the same URL that was passed on the establish call is passed to the validation call to retrieve the data from the Account Validation Partner.

No extra parameters are to be added in the URL between the establish call and the validation call to retrieve the data or TCKValidationService will return a Hash Validation Error. TCKValidationService needs to do this to verify that the same establish call and the validation call are from the same established request to the Account Validation Partner.

Here is an example of an event that triggered an error in the Hash Validation. The “establish” call is already done and completed. You successfully login to PayWithMyBank Demo Bank, but then afterwards you get an error when performing the “Validate” call to retrieve the data.

On the second call (“validate”), the URL being passed in has all the PayWithMyBank parameters attached. TCKValidationService is expecting the exact same URL that was passed on the establish call.

```
{"Error": "processRequest:Failed to validate requestSignature from PWMB:xIAY\xn6awue3tlum8huF88NAEA=", "ReturnCode": "1"}
```



Figure T.2: Error message due to mismatched URL hash value.

If you look at the detail in this error message, you will see that TCKValidationService appends the parameters to the URL passed in, so there are now duplicates (the highlighted section should not be passed in as part of the URL string):

```
log/server.log:02:07:41,167 ERROR [com.tck.validationservice.PayWithMyBankClient]
processRequest:Hash validation failed: hashStr:

http://local.sd:7001/OPSCClientWebSite/LoadWebEnrollment.do?ID=226333
&transactionId=1001269493&transactionType=1&merchantReference=249
&status=2&payment.paymentType=6&payment.paymentProvider.type=1&payment.account.verifi
ed=true&panel=1
&requestSignature= xIAYlxn6awue3tlum8huF88NAEA =
&transactionId=1001269493&transactionType=1&merchantReference=249
&status=2&payment.paymentType=6&payment.paymentProvider.type=1&payment.account.verifi
ed=true&panel=1,

PWMB: xIAYlxn6awue3tlum8huF88NAEA =, calculated:Vyo4dLJx6m4wFkVZp87BBilqEyo=
```

The solution for this is to make sure the Web Application does not add any extra parameters in between the “Establish” call and the subsequent “Validate” call to TCKValidationService.

### 3. Merchant Boarding

Your merchant should be properly boarded onto Payeezy in order for you to be able use the application.

As shown in application.properties above, you need to obtain API key, token, secret credentials to be able to execute any API calls.

## Appendix

### 1. PayWithMyBank Return Status Values

When the call is established and information is retrieved from PayWithMyBank, PayWithMyBank sends the transaction information back to the requestor via URL parameter, for example:

```
https://localhost:8443/pwmb/pwmb_merchant.jsp?transactionId=1001273536&transactionType=1&merchantReference=406&status=2&payment.paymentType=6&payment.paymentProvider.type=1&payment.account.verified=true&panel=1&requestSignature=HNLQRDEcxa%2Bpd0JcxHkTe9cpjSs%3D
```

*Figure A.1: Sample URL parameter contents showing the status code retried from the Establish connection to PayWithMyBank.*

One of the parameters is the status of the transaction. In the above example, the status returned is 2, which according to the table below, represents that the transaction has been “Authorized”.

Below is the table of the Return Status Values from transaction requests to PayWithMyBank (Note: some of these you might not see due to the transaction type being used by TCKValidationService).

Name	Value	Description
New	0	This is the initial status after the transaction is created but before the consumer chooses a financial institution for payment. Transactions in then New state are visible only through Notification or the API status operations, not through the Merchant Portal.
Pending	1	Payment is pending. This is the initial transaction status when the consumer has the Pay Panel open but has not yet authorized the transaction. Transactions in then Pending state are visible only through Notification or the API status operations, not through the Merchant Portal.
Authorized	2	Payment was authorized by the consumer (they have selected their account and clicked 'Pay') but the ACH has not yet been submitted for processing.
Processed	3	Transaction was processed. The ACH debit has been submitted to the ACH network after authorization by the consumer.
Completed	4	Transaction was paid, funds transferred.
Failed	5	Internal failure of transaction. (Unrelated to the merchant).
Expired	6	Pending transaction timed out before the consumer authorized the payment via the Pay Panel.
Canceled	7	Transaction was canceled by the consumer by closing the Pay Panel prior to the transaction being authorized or canceling the transaction from the Pay Panel.
Denied	8	Money transfer was denied, consumer account does not have enough funds.
Reversed	10	Payment was reversed (charged back by the bank or PayWithMyBank).
Partially Refunded	11	Payment was partially refunded.
Refunded	12	Payment was refunded.
Voided	13	Authorization was voided by the merchant. This occurs when the 'cancel' API is invoked or the transaction is canceled in the Merchant Portal (this must be

		done prior to the transaction being moved to 'Processed' status).
OnHold	14	<p>A transaction is put on hold after authorization if something is preventing the normal flow, such as when a ACH debit could not be initiated, or if the PayWithMyBank wants the merchant to verify that it really approves of this customer authorizing this amount. The payment may take longer to process than usual.</p> <p>Note: if a transaction is On Hold, you can verify it through the Merchant Portal to let it resume the normal flow of payment processing. Currently, there is no equivalent way to do that verification through the API.</p>

## 2. PayWithMyBank Transaction Test Passwords

PayWithMyBank provides a list of Passwords for “Demo Bank” to use in their Shadow Box to test different transaction error.

For example, if you enter “LoginError” as the password, as shown in this example:

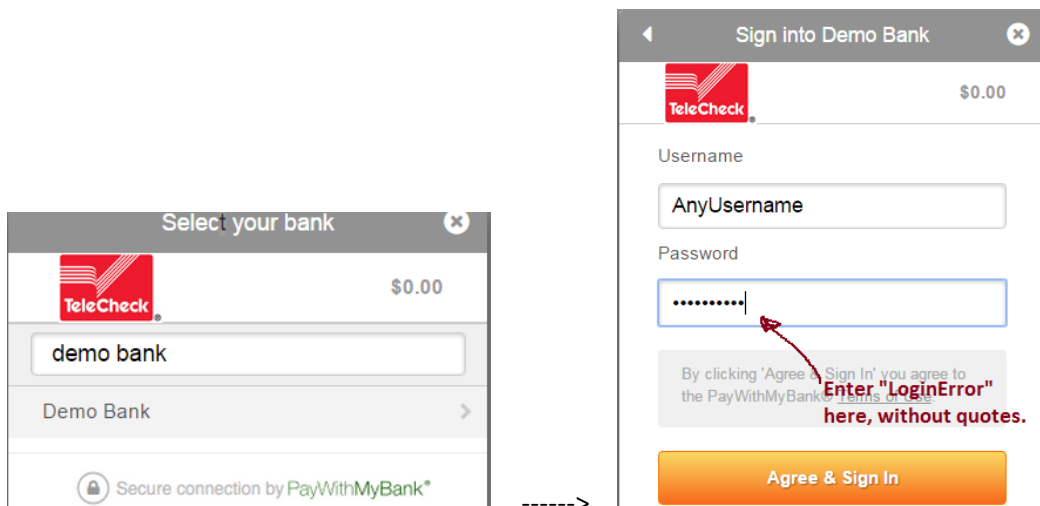


Figure A.2: Example for entering a specific password to test different return cases from PayWithMyBank.

After you click “Agree & Sign In”, the PayWithMyBank Shadow Box will return with a message regarding a Login Error or Wrong Username or Password:

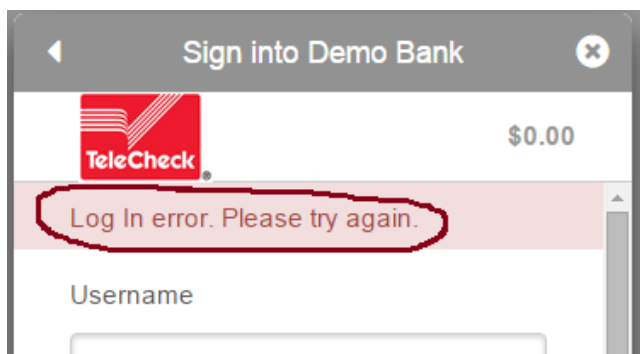


Figure A.3: Example test error message for a specific password to test different return cases from PayWithMyBank.

Below is the table of Password phrases to use for testing with “Demo Bank”, and what the expected returned message will be (Note: this list is provided by PayWithMyBank, and messages may differ as this is a Demo Bank provided by PayWithMyBank).

Phrase	Notes
<b>NoEligibleAccounts</b>	No eligible accounts found
<b>LoginError</b>	Wrong username or password
<b>NotRecognized</b>	Main Error that users see when using an ACA
<b>NoSuchField</b>	This error ultimately ends up as a PageNotRec error. It happens when an item cannot be found on the page. ACA will try to execute another page. If there is no another page, “page not recognized” error will be returned. Customers shouldn’t see this error.
<b>PostError</b>	HTTP connection error using GET. Customer shouldn’t see this error. In real ACA, this will result in a Site not available” error.
<b>GetError</b>	HTTP connection error using POST. Customer shouldn’t see this error. In real ACA, this will result in a Site not available” error.
<b>PromptTypeError</b>	When an ACA fails to create a prompt, this error is returned. If this error appears, it means the ACA has a bug.
<b>JsError</b>	When ACA tries to run javascript code and there are any errors during running, this error will be thrown.
<b>Unavailable</b>	Bank Site cannot be reached.
<b>AccountLocked</b>	User’s account is locked.
<b>Others</b>	There are some run time exceptions that not captured by ACA, like NPE(null pointer exception), array out of bounds exception and so on. This probably means that it needs more work.
<b>BankAction</b>	The bank requires the user to login and perform some action on their site.
<b>ConnectError</b>	There was a connection problem when accessing bank site
<b>BlockedIpError</b>	The bank indicates the caller IP was blocked
<b>ValidRouteCodeExtra</b>	Connector returns 2 accounts whose route codes are larger than 9 digits: one of them has a valid route code as substring, so both accounts use the same valid code
<b>InvalidRouteCodeExtra</b>	Connector returns a single account whose route code is larger than 9 digits, but no valid route code is found as substring. Hence, the account is ignored
<b>TimeoutError</b>	In order to simulate a timeout, connector sleeps for at least a minute before actually doing anything.
<b>TestPrompts</b>	To test different bank inputs. This is to test the prompts on next page, including (Checkbox, radio, text, password, date, description and so on)
<b>NotEnoughFunds</b>	Connector returns a single account with zero balance. This is similar to having no eligible accounts, but with different reason.
<b>NotEnoughFundsExtra</b>	Connector returns two accounts. One with zero balance, the other with a valid balance.

<b>InvalidAccountNumberSize</b>	Connector returns a single account, but with account number shorter than the required. This is to test how the screen filters invalid accounts
<b>InvalidAccountNumberSizeExtra</b>	Connector returns two accounts. One with account number shorter (3 characters) than the required, the other with valid account number.
<b>PartialAccountNumbers</b>	Connector returns two accounts, however only with partial numbers. Simulating when for example the account is new and we still don't have statements to get full account number.
<b>OnlyPartials</b>	PartialAccountNumbers + NoRouteCode
<b>NoCustomer</b>	Simulates as if FIC was not able to retrieve customer information
<b>NoRouteCode</b>	Regular flow with 2 accounts, but none with route code. This prompts a question for account location, where user must select where the account was open (from the given options)
<b>InvalidRouteCode</b>	Regular flow but simulates an invalid routing code (will simulate if ProfitStars returns invalid routing code)
<b>2FA</b>	Simulates as the bank requested a challenge question to the user. The question should be answered with the word 'error' if it's necessary to simulate a wrong credential. Otherwise, it should be anything to have a successful access.
<b>WrongCredentials</b>	Simulates retry scenario, where the user provide wrong challenge (or anything that isn't userid or password) and is allowed to retry
<b>SiteRequestError</b>	Simulates as if the bank couldn't process a particular request, allowing user to retry it
<b>SessionTimeout</b>	Simulates as if the user took too long to provide the requested information, since the bank session is already expired
<b>StressTest</b>	Simulates FIC's file download (ask for an URL) and CPU consumption (asks for time in milliseconds)
<b>AccountsWithNameAndAddress</b>	Simulates an User with 2 accounts and each one with different names and addresses.
<b>ManyInformation</b>	Simulates an User with 10 accounts.
<b>AccountNotSupported</b>	Simulates an User with an account not supported by our service (Chase Liquid, etc)
<b>AmountNull</b>	Simulates the Demo Checking Account returning an amount with null value.



## Revision History

Version	Date	Revision
<b>V2.0</b>	March 2017	<ul style="list-style-type: none"><li>• Re-wrote major parts to make this compatible with sample Spring Boot application and services that Payeezy exposes.</li></ul>
<b>V1.01</b>	June 29, 2016	<ul style="list-style-type: none"><li>• Added PWMB Method 3 – Same as Method 2 with the addition of Application.properties.</li><li>• Specified which versions of Java Versions / Containers were used for the test code for the PWMB examples.</li><li>• Additional clarification for Troubleshooting # 3 about the need for the Subscriber ID to be correctly setup with the PWMB AddOns.</li><li>• Added section for document revision history.</li></ul>
<b>V1.0</b>	June 7, 2016	<ul style="list-style-type: none"><li>• Initial Version.</li></ul>