PHP

# Account API

## /api/v1/Account/GetAccountSettings

protected function apiV1AccountGetAccountSettingsGetRequest($account, $password)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1AccountGetAccountSettingsGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1AccountGetAccountSettingsGet'

);

}

$resourcePath = '/api/v1/Account/GetAccountSettings';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

# Communication API

## /api/v1/Communication/EmailReceipt

protected function apiV1CommunicationEmailReceiptPostRequest($account, $password, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1CommunicationEmailReceiptPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1CommunicationEmailReceiptPost'

);

}

$resourcePath = '/api/v1/Communication/EmailReceipt';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

# Reporting API

## /api/v1/Reporting/GetBatchesByDate/{batchDate}

protected function apiV1ReportingGetBatchesByDateBatchDateGetRequest($batch\_date, $account, $password)

{

// verify the required parameter 'batch\_date' is set

if ($batch\_date === null || (is\_array($batch\_date) && count($batch\_date) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $batch\_date when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetBatchesByDateBatchDateGet'

);

}

$resourcePath = '/api/v1/Reporting/GetBatchesByDate/{batchDate}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($batch\_date !== null) {

$resourcePath = str\_replace(

'{' . 'batchDate' . '}',

ObjectSerializer::toPathValue($batch\_date),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/Reporting/GetCheckTerminals

protected function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetRequest($sec\_code, $account, $password)

{

// verify the required parameter 'sec\_code' is set

if ($sec\_code === null || (is\_array($sec\_code) && count($sec\_code) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $sec\_code when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($sec\_code !== null) {

$resourcePath = str\_replace(

'{' . 'secCode' . '}',

ObjectSerializer::toPathValue($sec\_code),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}

protected function apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetRequest($sec\_code, $account, $password)

{

// verify the required parameter 'sec\_code' is set

if ($sec\_code === null || (is\_array($sec\_code) && count($sec\_code) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $sec\_code when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($sec\_code !== null) {

$resourcePath = str\_replace(

'{' . 'secCode' . '}',

ObjectSerializer::toPathValue($sec\_code),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/Reporting/GetCheckTerminalSettings/{terminalID}

protected function apiV1ReportingGetCheckTerminalSettingsTerminalIDGetRequest($terminal\_id, $account, $password)

{

// verify the required parameter 'terminal\_id' is set

if ($terminal\_id === null || (is\_array($terminal\_id) && count($terminal\_id) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $terminal\_id when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetCheckTerminalSettingsTerminalIDGet'

);

}

$resourcePath = '/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($terminal\_id !== null) {

$resourcePath = str\_replace(

'{' . 'terminalID' . '}',

ObjectSerializer::toPathValue($terminal\_id),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/Reporting/GetTransactionsByBatch/{batch}

protected function apiV1ReportingGetTransactionsByBatchBatchGetRequest($batch, $account, $password)

{

// verify the required parameter 'batch' is set

if ($batch === null || (is\_array($batch) && count($batch) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $batch when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetTransactionsByBatchBatchGet'

);

}

$resourcePath = '/api/v1/Reporting/GetTransactionsByBatch/{batch}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($batch !== null) {

$resourcePath = str\_replace(

'{' . 'batch' . '}',

ObjectSerializer::toPathValue($batch),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/Reporting/GetTransactionsByDate/{transactionDate}

protected function apiV1ReportingGetTransactionsByDateTransactionDateGetRequest($transaction\_date, $account, $password)

{

// verify the required parameter 'transaction\_date' is set

if ($transaction\_date === null || (is\_array($transaction\_date) && count($transaction\_date) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $transaction\_date when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1ReportingGetTransactionsByDateTransactionDateGet'

);

}

$resourcePath = '/api/v1/Reporting/GetTransactionsByDate/{transactionDate}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($transaction\_date !== null) {

$resourcePath = str\_replace(

'{' . 'transactionDate' . '}',

ObjectSerializer::toPathValue($transaction\_date),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'GET',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

# Vault API

## /api/v1/Vault/GetVaultRecord/{reference}

## /api/v1/Vault/SaveVault

public function apiV1VaultSaveVaultPostAsyncWithHttpInfo($account, $password, $body = null)

{

$returnType = '\Swagger\Client\Model\Response[]';

$request = $this->apiV1VaultSaveVaultPostRequest($account, $password, $body);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/SaveVaultCard/{vaultID}

public function apiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo($account, $password, $vault\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultSaveVaultCardVaultIDPostRequest($account, $password, $vault\_id, $body);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/SearchVault

public function apiV1VaultSearchVaultPostAsyncWithHttpInfo($account, $password, $body = null)

{

$returnType = '\Swagger\Client\Model\Record[]';

$request = $this->apiV1VaultSearchVaultPostRequest($account, $password, $body);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/DeleteVaultCardByID/{vaultCardID}

public function apiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo($vault\_card\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultDeleteVaultCardByIDVaultCardIDPostRequest($vault\_card\_id, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/DeleteVaultCheckByID/{vaultCheckID}

public function apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo($vault\_check\_id, $account, $password)

{

$returnType = '\Swagger\Client\Model\Response';

$request = $this->apiV1VaultDeleteVaultCheckByIDVaultCheckIDPostRequest($vault\_check\_id, $account, $password);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/SubmitCheckWithVaultCheckID/{vaultCheckID}

public function apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo($account, $password, $vault\_check\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\CheckTransaction';

$request = $this->apiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostRequest($account, $password, $vault\_check\_id, $body);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

## /api/v1/Vault/SubmitWithVaultCardID/{vaultCardID}

public function apiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo($account, $password, $vault\_card\_id, $body = null)

{

$returnType = '\Swagger\Client\Model\Transaction';

$request = $this->apiV1VaultSubmitWithVaultCardIDVaultCardIDPostRequest($account, $password, $vault\_card\_id, $body);

return $this->client

->sendAsync($request, $this->createHttpClientOption())

->then(

function ($response) use ($returnType) {

$responseBody = $response->getBody();

if ($returnType === '\SplFileObject') {

$content = $responseBody; //stream goes to serializer

} else {

$content = $responseBody->getContents();

if ($returnType !== 'string') {

$content = json\_decode($content);

}

}

return [

ObjectSerializer::deserialize($content, $returnType, []),

$response->getStatusCode(),

$response->getHeaders()

];

},

function ($exception) {

$response = $exception->getResponse();

$statusCode = $response->getStatusCode();

throw new ApiException(

sprintf(

'[%d] Error connecting to the API (%s)',

$statusCode,

$exception->getRequest()->getUri()

),

$statusCode,

$response->getHeaders(),

$response->getBody()

);

}

);

}

# VirtualTerminal API

## /api/v1/VirtualTerminal/Submit

protected function apiV1VirtualTerminalSubmitPostRequest($account, $password, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalSubmitPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalSubmitPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/Submit';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/VirtualTerminal/SubmitCheck

protected function apiV1VirtualTerminalSubmitCheckPostRequest($account, $password, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalSubmitCheckPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalSubmitCheckPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/SubmitCheck';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/VirtualTerminal/Mark/{transactionID}

protected function apiV1VirtualTerminalMarkTransactionIDPostRequest($transaction\_id, $account, $password)

{

// verify the required parameter 'transaction\_id' is set

if ($transaction\_id === null || (is\_array($transaction\_id) && count($transaction\_id) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $transaction\_id when calling apiV1VirtualTerminalMarkTransactionIDPost'

);

}

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalMarkTransactionIDPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalMarkTransactionIDPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/Mark/{transactionID}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($transaction\_id !== null) {

$resourcePath = str\_replace(

'{' . 'transactionID' . '}',

ObjectSerializer::toPathValue($transaction\_id),

$resourcePath

);

}

// body params

$\_tempBody = null;

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

[]

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/VirtualTerminal/MarkTransactions

protected function apiV1VirtualTerminalMarkTransactionsPostRequest($account, $password, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalMarkTransactionsPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalMarkTransactionsPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/MarkTransactions';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/VirtualTerminal/Query

protected function apiV1VirtualTerminalQueryPostRequest($account, $password, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalQueryPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalQueryPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/Query';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}

## /api/v1/VirtualTerminal/UpdateTransactionInfo/{transactionID}

protected function apiV1VirtualTerminalUpdateTransactionInfoTransactionIDPostRequest($account, $password, $transaction\_id, $body = null)

{

// verify the required parameter 'account' is set

if ($account === null || (is\_array($account) && count($account) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $account when calling apiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost'

);

}

// verify the required parameter 'password' is set

if ($password === null || (is\_array($password) && count($password) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $password when calling apiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost'

);

}

// verify the required parameter 'transaction\_id' is set

if ($transaction\_id === null || (is\_array($transaction\_id) && count($transaction\_id) === 0)) {

throw new \InvalidArgumentException(

'Missing the required parameter $transaction\_id when calling apiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost'

);

}

$resourcePath = '/api/v1/VirtualTerminal/UpdateTransactionInfo/{transactionID}';

$formParams = [];

$queryParams = [];

$headerParams = [];

$httpBody = '';

$multipart = false;

// header params

if ($account !== null) {

$headerParams['Account'] = ObjectSerializer::toHeaderValue($account);

}

// header params

if ($password !== null) {

$headerParams['Password'] = ObjectSerializer::toHeaderValue($password);

}

// path params

if ($transaction\_id !== null) {

$resourcePath = str\_replace(

'{' . 'transactionID' . '}',

ObjectSerializer::toPathValue($transaction\_id),

$resourcePath

);

}

// body params

$\_tempBody = null;

if (isset($body)) {

$\_tempBody = $body;

}

if ($multipart) {

$headers = $this->headerSelector->selectHeadersForMultipart(

['text/plain', 'application/json', 'text/json']

);

} else {

$headers = $this->headerSelector->selectHeaders(

['text/plain', 'application/json', 'text/json'],

['application/json', 'text/json', 'application/\_\*+json']

);

}

// for model (json/xml)

if (isset($\_tempBody)) {

// $\_tempBody is the method argument, if present

$httpBody = $\_tempBody;

// \stdClass has no \_\_toString(), so we should encode it manually

if ($httpBody instanceof \stdClass && $headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($httpBody);

}

} elseif (count($formParams) > 0) {

if ($multipart) {

$multipartContents = [];

foreach ($formParams as $formParamName => $formParamValue) {

$multipartContents[] = [

'name' => $formParamName,

'contents' => $formParamValue

];

}

// for HTTP post (form)

$httpBody = new MultipartStream($multipartContents);

} elseif ($headers['Content-Type'] === 'application/json') {

$httpBody = \GuzzleHttp\json\_encode($formParams);

} else {

// for HTTP post (form)

$httpBody = \GuzzleHttp\Psr7\build\_query($formParams);

}

}

$defaultHeaders = [];

if ($this->config->getUserAgent()) {

$defaultHeaders['User-Agent'] = $this->config->getUserAgent();

}

$headers = array\_merge(

$defaultHeaders,

$headerParams,

$headers

);

$query = \GuzzleHttp\Psr7\build\_query($queryParams);

return new Request(

'POST',

$this->config->getHost() . $resourcePath . ($query ? "?{$query}" : ''),

$headers,

$httpBody

);

}