C#

# API Client

using System;

using System.Collections;

using System.Collections.Generic;

using System.Globalization;

using System.Text.RegularExpressions;

using System.IO;

using System.Web;

using System.Linq;

using System.Net;

using System.Text;

using Newtonsoft.Json;

using RestSharp;

namespace IO.Swagger.Client

{

public partial class ApiClient

{

private JsonSerializerSettings serializerSettings = new JsonSerializerSettings

{

ConstructorHandling = ConstructorHandling.AllowNonPublicDefaultConstructor

};

partial void InterceptRequest(IRestRequest request);

partial void InterceptResponse(IRestRequest request, IRestResponse response);

public ApiClient()

{

Configuration = IO.Swagger.Client.Configuration.Default;

RestClient = new RestClient("/");

}

public ApiClient(Configuration config)

{

Configuration = config ?? IO.Swagger.Client.Configuration.Default;

RestClient = new RestClient(Configuration.BasePath);

}

public ApiClient(String basePath = "/")

{

if (String.IsNullOrEmpty(basePath))

throw new ArgumentException("basePath cannot be empty");

RestClient = new RestClient(basePath);

Configuration = Client.Configuration.Default;

}

public IReadableConfiguration Configuration { get; set; }

public RestClient RestClient { get; set; }

private RestRequest PrepareRequest(

String path, RestSharp.Method method,

List<KeyValuePair<String, String>> queryParams,

Object postBody,

Dictionary<String, String> headerParams,

Dictionary<String, String> formParams,

Dictionary<String, FileParameter> fileParams,

Dictionary<String, String> pathParams,

String contentType)

{

var request = new RestRequest(path, method);

// add path parameter, if any

foreach (var param in pathParams)

request.AddParameter(param.Key, param.Value, ParameterType.UrlSegment);

// add header parameter, if any

foreach (var param in headerParams)

request.AddHeader(param.Key, param.Value);

// add query parameter, if any

foreach (var param in queryParams)

request.AddQueryParameter(param.Key, param.Value);

// add form parameter, if any

foreach (var param in formParams)

request.AddParameter(param.Key, param.Value);

// add file parameter, if any

foreach (var param in fileParams)

{

request.AddFile(param.Value.Name, param.Value.Writer, param.Value.FileName, param.Value.ContentType);

}

if (postBody != null) // http body (model or byte[]) parameter

{

request.AddParameter(contentType, postBody, ParameterType.RequestBody);

}

return request;

}

public Object CallApi(

String path, RestSharp.Method method,

List<KeyValuePair<String, String>> queryParams,

Object postBody,

Dictionary<String, String> headerParams,

Dictionary<String, String> formParams,

Dictionary<String, FileParameter> fileParams,

Dictionary<String, String> pathParams,

String contentType)

{

var request = PrepareRequest(

path, method, queryParams, postBody, headerParams, formParams, fileParams,

pathParams, contentType);

// set timeout

RestClient.Timeout = Configuration.Timeout;

// set user agent

RestClient.UserAgent = Configuration.UserAgent;

InterceptRequest(request);

var response = RestClient.Execute(request);

InterceptResponse(request, response);

return (Object)response;

}

public async System.Threading.Tasks.Task<Object> CallApiAsync(

String path,

RestSharp.Method method,

List<KeyValuePair<String, String>> queryParams,

Object postBody,

Dictionary<String, String> headerParams,

Dictionary<String, String> formParams,

Dictionary<String, FileParameter> fileParams,

Dictionary<String, String> pathParams,

String contentType)

{

var request = PrepareRequest(

path, method, queryParams, postBody, headerParams, formParams, fileParams,

pathParams, contentType);

InterceptRequest(request);

var response = await RestClient.ExecuteTaskAsync(request);

InterceptResponse(request, response);

return (Object)response;

}

public object Deserialize(IRestResponse response, Type type)

{

IList<Parameter> headers = response.Headers;

if (type == typeof(byte[])) // return byte array

{

return response.RawBytes;

}

if (type == typeof(Stream))

{

if (headers != null)

{

var filePath = String.IsNullOrEmpty(Configuration.TempFolderPath)

? Path.GetTempPath()

: Configuration.TempFolderPath;

var regex = new Regex(@"Content-Disposition=.\*filename=['""]?([^'""\s]+)['""]?$");

foreach (var header in headers)

{

var match = regex.Match(header.ToString());

if (match.Success)

{

string fileName = filePath + SanitizeFilename(match.Groups[1].Value.Replace("\"", "").Replace("'", ""));

File.WriteAllBytes(fileName, response.RawBytes);

return new FileStream(fileName, FileMode.Open);

}

}

}

var stream = new MemoryStream(response.RawBytes);

return stream;

}

if (type.Name.StartsWith("System.Nullable`1[[System.DateTime")) // return a datetime object

{

return DateTime.Parse(response.Content, null, System.Globalization.DateTimeStyles.RoundtripKind);

}

if (type == typeof(String) || type.Name.StartsWith("System.Nullable")) // return primitive type

{

return ConvertType(response.Content, type);

}

// at this point, it must be a model (json)

try

{

return JsonConvert.DeserializeObject(response.Content, type, serializerSettings);

}

catch (Exception e)

{

throw new ApiException(500, e.Message);

}

}

public String Serialize(object obj)

{

try

{

return obj != null ? JsonConvert.SerializeObject(obj) : null;

}

catch (Exception e)

{

throw new ApiException(500, e.Message);

}

}

public bool IsJsonMime(String mime)

{

var jsonRegex = new Regex("(?i)^(application/json|[^;/ \t]+/[^;/ \t]+[+]json)[ \t]\*(;.\*)?$");

return mime != null && (jsonRegex.IsMatch(mime) || mime.Equals("application/json-patch+json"));

}

public String SelectHeaderContentType(String[] contentTypes)

{

if (contentTypes.Length == 0)

return "application/json";

foreach (var contentType in contentTypes)

{

if (IsJsonMime(contentType.ToLower()))

return contentType;

}

return contentTypes[0]; // use the first content type specified in 'consumes'

}

public String SelectHeaderAccept(String[] accepts)

{

if (accepts.Length == 0)

return null;

if (accepts.Contains("application/json", StringComparer.OrdinalIgnoreCase))

return "application/json";

return String.Join(",", accepts);

}

public static dynamic ConvertType(dynamic fromObject, Type toObject)

{

return Convert.ChangeType(fromObject, toObject);

}

public static byte[] ReadAsBytes(Stream inputStream)

{

byte[] buf = new byte[16 \* 1024];

using (MemoryStream ms = new MemoryStream())

{

int count;

while ((count = inputStream.Read(buf, 0, buf.Length)) > 0)

{

ms.Write(buf, 0, count);

}

return ms.ToArray();

}

}

public static string SanitizeFilename(string filename)

{

Match match = Regex.Match(filename, @".\*[/\\](.\*)$");

if (match.Success)

{

return match.Groups[1].Value;

}

else

{

return filename;

}

}

}

}

# Account API

## /api/v1/Account/GetAccountSettings

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using RestSharp;

using IO.Swagger.Client;

using IO.Swagger.Model;

namespace IO.Swagger.Api

{

public interface IAccountApi : IApiAccessor

{

System.Threading.Tasks.Task<Settings> GetSettingsAsync(string account, string password);

System.Threading.Tasks.Task<ApiResponse<Settings>> GetSettingsWithHttpInfoAsync(string account, string password);

}

public partial class AccountApi : IAccountApi

{

private IO.Swagger.Client.ExceptionFactory \_exceptionFactory = (name, response) => null;

public AccountApi(String basePath)

{

this.Configuration = new IO.Swagger.Client.Configuration { BasePath = basePath };

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public AccountApi()

{

this.Configuration = IO.Swagger.Client.Configuration.Default;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public AccountApi(IO.Swagger.Client.Configuration configuration = null)

{

if (configuration == null)

this.Configuration = IO.Swagger.Client.Configuration.Default;

else

this.Configuration = configuration;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public IO.Swagger.Client.Configuration Configuration { get; set; }

public IO.Swagger.Client.ExceptionFactory ExceptionFactory

{

get

{

if (\_exceptionFactory != null && \_exceptionFactory.GetInvocationList().Length > 1)

{

throw new InvalidOperationException("Multicast delegate for ExceptionFactory is unsupported.");

}

return \_exceptionFactory;

}

set { \_exceptionFactory = value; }

}

public async System.Threading.Tasks.Task<Settings> GetSettingsAsync(string account, string password)

{

ApiResponse<Settings> localVarResponse = await GetSettingsWithHttpInfoAsync(account, password);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Settings>> GetSettingsWithHttpInfoAsync(string account, string password)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling AccountApi->ApiV1AccountGetAccountSettingsGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling AccountApi->ApiV1AccountGetAccountSettingsGet");

var localVarPath = "/api/v1/Account/GetAccountSettings";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1AccountGetAccountSettingsGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Settings>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Settings)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Settings)));

}

}

}

# Communication API

## /api/v1/Communication/EmailReceipt

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using RestSharp;

using IO.Swagger.Client;

using IO.Swagger.Model;

namespace IO.Swagger.Api

{

public interface ICommunicationApi : IApiAccessor

{

Response ApiV1CommunicationEmailReceiptPost(string account, string password, EmailReceipt body = null);

ApiResponse<Response> ApiV1CommunicationEmailReceiptPostWithHttpInfo(string account, string password, EmailReceipt body = null);

System.Threading.Tasks.Task<Response> ApiV1CommunicationEmailReceiptPostAsync(string account, string password, EmailReceipt body = null);

System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1CommunicationEmailReceiptPostAsyncWithHttpInfo(string account, string password, EmailReceipt body = null);

}

public partial class CommunicationApi : ICommunicationApi

{

private IO.Swagger.Client.ExceptionFactory \_exceptionFactory = (name, response) => null;

public CommunicationApi(String basePath)

{

this.Configuration = new IO.Swagger.Client.Configuration { BasePath = basePath };

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public CommunicationApi()

{

this.Configuration = IO.Swagger.Client.Configuration.Default;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public CommunicationApi(IO.Swagger.Client.Configuration configuration = null)

{

if (configuration == null) // use the default one in Configuration

this.Configuration = IO.Swagger.Client.Configuration.Default;

else

this.Configuration = configuration;

ExceptionFactory = IO.Swagger.Client.Configuration.DefaultExceptionFactory;

}

public String GetBasePath()

{

return this.Configuration.ApiClient.RestClient.BaseUrl.ToString();

}

[Obsolete("SetBasePath is deprecated, please do 'Configuration.ApiClient = new ApiClient(\"http://new-path\")' instead.")]

public void SetBasePath(String basePath)

{

// do nothing

}

public IO.Swagger.Client.Configuration Configuration { get; set; }

public IO.Swagger.Client.ExceptionFactory ExceptionFactory

{

get

{

if (\_exceptionFactory != null && \_exceptionFactory.GetInvocationList().Length > 1)

{

throw new InvalidOperationException("Multicast delegate for ExceptionFactory is unsupported.");

}

return \_exceptionFactory;

}

set { \_exceptionFactory = value; }

}

[Obsolete("DefaultHeader is deprecated, please use Configuration.DefaultHeader instead.")]

public IDictionary<String, String> DefaultHeader()

{

return new ReadOnlyDictionary<string, string>(this.Configuration.DefaultHeader);

}

[Obsolete("AddDefaultHeader is deprecated, please use Configuration.AddDefaultHeader instead.")]

public void AddDefaultHeader(string key, string value)

{

this.Configuration.AddDefaultHeader(key, value);

}

public Response ApiV1CommunicationEmailReceiptPost(string account, string password, EmailReceipt body = null)

{

ApiResponse<Response> localVarResponse = ApiV1CommunicationEmailReceiptPostWithHttpInfo(account, password, body);

return localVarResponse.Data;

}

public ApiResponse<Response> ApiV1CommunicationEmailReceiptPostWithHttpInfo(string account, string password, EmailReceipt body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling CommunicationApi->ApiV1CommunicationEmailReceiptPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling CommunicationApi->ApiV1CommunicationEmailReceiptPost");

var localVarPath = "/api/v1/Communication/EmailReceipt";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)this.Configuration.ApiClient.CallApi(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1CommunicationEmailReceiptPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

public async System.Threading.Tasks.Task<Response> ApiV1CommunicationEmailReceiptPostAsync(string account, string password, EmailReceipt body = null)

{

ApiResponse<Response> localVarResponse = await ApiV1CommunicationEmailReceiptPostAsyncWithHttpInfo(account, password, body);

return localVarResponse.Data;

}

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1CommunicationEmailReceiptPostAsyncWithHttpInfo(string account, string password, EmailReceipt body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling CommunicationApi->ApiV1CommunicationEmailReceiptPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling CommunicationApi->ApiV1CommunicationEmailReceiptPost");

var localVarPath = "/api/v1/Communication/EmailReceipt";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1CommunicationEmailReceiptPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

}

}

# Reporting API

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

public async System.Threading.Tasks.Task<ApiResponse<List<Batch>>> ApiV1ReportingGetBatchesByDateBatchDateGetAsyncWithHttpInfo(DateTime? batchDate, string account, string password)

{

// verify the required parameter 'batchDate' is set

if (batchDate == null)

throw new ApiException(400, "Missing required parameter 'batchDate' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetBatchesByDateBatchDateGet");

var localVarPath = "/api/v1/Reporting/GetBatchesByDate/{batchDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batchDate != null) localVarPathParams.Add("batchDate", this.Configuration.ApiClient.ParameterToString(batchDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetBatchesByDateBatchDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<Batch>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<Batch>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<Batch>)));

}

## /api/v1/Reporting/GetCheckTerminals

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalsGetAsyncWithHttpInfo(string account, string password)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalsGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminals";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalsGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGetAsyncWithHttpInfo(string secCode, string account, string password)

{

// verify the required parameter 'secCode' is set

if (secCode == null)

throw new ApiException(400, "Missing required parameter 'secCode' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettingsBySEC/{secCode}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (secCode != null) localVarPathParams.Add("secCode", this.Configuration.ApiClient.ParameterToString(secCode)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsBySECSecCodeGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<List<TerminalSettings>>> ApiV1ReportingGetCheckTerminalSettingsTerminalIDGetAsyncWithHttpInfo(int? terminalID, string account, string password)

{

// verify the required parameter 'terminalID' is set

if (terminalID == null)

throw new ApiException(400, "Missing required parameter 'terminalID' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet");

var localVarPath = "/api/v1/Reporting/GetCheckTerminalSettings/{terminalID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (terminalID != null) localVarPathParams.Add("terminalID", this.Configuration.ApiClient.ParameterToString(terminalID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetCheckTerminalSettingsTerminalIDGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<TerminalSettings>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<TerminalSettings>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<TerminalSettings>)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByBatchBatchGetAsyncWithHttpInfo(string batch, string account, string password)

{

// verify the required parameter 'batch' is set

if (batch == null)

throw new ApiException(400, "Missing required parameter 'batch' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByBatchBatchGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByBatch/{batch}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (batch != null) localVarPathParams.Add("batch", this.Configuration.ApiClient.ParameterToString(batch)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByBatchBatchGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<List<QueryTransaction>>> ApiV1ReportingGetTransactionsByDateTransactionDateGetAsyncWithHttpInfo(DateTime? transactionDate, string account, string password)

{

// verify the required parameter 'transactionDate' is set

if (transactionDate == null)

throw new ApiException(400, "Missing required parameter 'transactionDate' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling ReportingApi->ApiV1ReportingGetTransactionsByDateTransactionDateGet");

var localVarPath = "/api/v1/Reporting/GetTransactionsByDate/{transactionDate}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (transactionDate != null) localVarPathParams.Add("transactionDate", this.Configuration.ApiClient.ParameterToString(transactionDate)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1ReportingGetTransactionsByDateTransactionDateGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<QueryTransaction>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<QueryTransaction>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<QueryTransaction>)));

}

}

# Vault API

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

public async System.Threading.Tasks.Task<ApiResponse<Record>> ApiV1VaultGetVaultRecordReferenceGetAsyncWithHttpInfo(string reference, string account, string password)

{

// verify the required parameter 'reference' is set

if (reference == null)

throw new ApiException(400, "Missing required parameter 'reference' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultGetVaultRecordReferenceGet");

var localVarPath = "/api/v1/Vault/GetVaultRecord/{reference}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (reference != null) localVarPathParams.Add("reference", this.Configuration.ApiClient.ParameterToString(reference)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.GET, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultGetVaultRecordReferenceGet", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Record>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Record)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Record)));

}

## /api/v1/Vault/SaveVault

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo(string account, string password, int? vaultID, VaultCreditCard body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'vaultID' is set

if (vaultID == null)

throw new ApiException(400, "Missing required parameter 'vaultID' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

var localVarPath = "/api/v1/Vault/SaveVaultCard/{vaultID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultID != null) localVarPathParams.Add("vaultID", this.Configuration.ApiClient.ParameterToString(vaultID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSaveVaultCardVaultIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultSaveVaultCardVaultIDPostAsyncWithHttpInfo(string account, string password, int? vaultID, VaultCreditCard body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

// verify the required parameter 'vaultID' is set

if (vaultID == null)

throw new ApiException(400, "Missing required parameter 'vaultID' when calling VaultApi->ApiV1VaultSaveVaultCardVaultIDPost");

var localVarPath = "/api/v1/Vault/SaveVaultCard/{vaultID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultID != null) localVarPathParams.Add("vaultID", this.Configuration.ApiClient.ParameterToString(vaultID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSaveVaultCardVaultIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

## /api/v1/Vault/SearchVault

public async System.Threading.Tasks.Task<ApiResponse<List<Record>>> ApiV1VaultSearchVaultPostAsyncWithHttpInfo(string account, string password, SearchVault body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSearchVaultPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSearchVaultPost");

var localVarPath = "/api/v1/Vault/SearchVault";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSearchVaultPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<List<Record>>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(List<Record>)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(List<Record>)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCardByIDVaultCardIDPostAsyncWithHttpInfo(int? vaultCardID, string account, string password)

{

// verify the required parameter 'vaultCardID' is set

if (vaultCardID == null)

throw new ApiException(400, "Missing required parameter 'vaultCardID' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultDeleteVaultCardByIDVaultCardIDPost");

var localVarPath = "/api/v1/Vault/DeleteVaultCardByID/{vaultCardID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultCardID != null) localVarPathParams.Add("vaultCardID", this.Configuration.ApiClient.ParameterToString(vaultCardID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultDeleteVaultCardByIDVaultCardIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPostAsyncWithHttpInfo(int? vaultCheckID, string account, string password)

{

// verify the required parameter 'vaultCheckID' is set

if (vaultCheckID == null)

throw new ApiException(400, "Missing required parameter 'vaultCheckID' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost");

var localVarPath = "/api/v1/Vault/DeleteVaultCheckByID/{vaultCheckID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultCheckID != null) localVarPathParams.Add("vaultCheckID", this.Configuration.ApiClient.ParameterToString(vaultCheckID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultDeleteVaultCheckByIDVaultCheckIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<CheckTransaction>> ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPostAsyncWithHttpInfo(string account, string password, int? vaultCheckID, CheckTransaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

// verify the required parameter 'vaultCheckID' is set

if (vaultCheckID == null)

throw new ApiException(400, "Missing required parameter 'vaultCheckID' when calling VaultApi->ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost");

var localVarPath = "/api/v1/Vault/SubmitCheckWithVaultCheckID/{vaultCheckID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultCheckID != null) localVarPathParams.Add("vaultCheckID", this.Configuration.ApiClient.ParameterToString(vaultCheckID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSubmitCheckWithVaultCheckIDVaultCheckIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<CheckTransaction>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(CheckTransaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(CheckTransaction)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Transaction>> ApiV1VaultSubmitWithVaultCardIDVaultCardIDPostAsyncWithHttpInfo(string account, string password, int? vaultCardID, Transaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

// verify the required parameter 'vaultCardID' is set

if (vaultCardID == null)

throw new ApiException(400, "Missing required parameter 'vaultCardID' when calling VaultApi->ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost");

var localVarPath = "/api/v1/Vault/SubmitWithVaultCardID/{vaultCardID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (vaultCardID != null) localVarPathParams.Add("vaultCardID", this.Configuration.ApiClient.ParameterToString(vaultCardID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VaultSubmitWithVaultCardIDVaultCardIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Transaction>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Transaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Transaction)));

}

}

# VirtualTerminal API

## /api/v1/VirtualTerminal/Submit

public async System.Threading.Tasks.Task<ApiResponse<CheckTransaction>> ApiV1VirtualTerminalSubmitCheckPostAsyncWithHttpInfo(string account, string password, CheckTransaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalSubmitCheckPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalSubmitCheckPost");

var localVarPath = "/api/v1/VirtualTerminal/SubmitCheck";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalSubmitCheckPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<CheckTransaction>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(CheckTransaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(CheckTransaction)));

}

## /api/v1/VirtualTerminal/SubmitCheck

public async System.Threading.Tasks.Task<ApiResponse<CheckTransaction>> ApiV1VirtualTerminalSubmitCheckPostAsyncWithHttpInfo(string account, string password, CheckTransaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalSubmitCheckPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalSubmitCheckPost");

var localVarPath = "/api/v1/VirtualTerminal/SubmitCheck";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalSubmitCheckPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<CheckTransaction>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(CheckTransaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(CheckTransaction)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VirtualTerminalMarkTransactionIDPostAsyncWithHttpInfo(string transactionID, string account, string password)

{

// verify the required parameter 'transactionID' is set

if (transactionID == null)

throw new ApiException(400, "Missing required parameter 'transactionID' when calling VirtualTerminalApi->ApiV1VirtualTerminalMarkTransactionIDPost");

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalMarkTransactionIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalMarkTransactionIDPost");

var localVarPath = "/api/v1/VirtualTerminal/Mark/{transactionID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (transactionID != null) localVarPathParams.Add("transactionID", this.Configuration.ApiClient.ParameterToString(transactionID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalMarkTransactionIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

## /api/v1/VirtualTerminal/MarkTransactions

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VirtualTerminalMarkTransactionsPostAsyncWithHttpInfo(string account, string password, List<string> body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalMarkTransactionsPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalMarkTransactionsPost");

var localVarPath = "/api/v1/VirtualTerminal/MarkTransactions";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalMarkTransactionsPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}

## /api/v1/VirtualTerminal/Query

public async System.Threading.Tasks.Task<ApiResponse<QueryTransaction>> ApiV1VirtualTerminalQueryPostAsyncWithHttpInfo(string account, string password, Transaction body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalQueryPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalQueryPost");

var localVarPath = "/api/v1/VirtualTerminal/Query";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalQueryPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<QueryTransaction>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(QueryTransaction)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(QueryTransaction)));

}

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

public async System.Threading.Tasks.Task<ApiResponse<Response>> ApiV1VirtualTerminalUpdateTransactionInfoTransactionIDPostAsyncWithHttpInfo(string account, string password, string transactionID, UpdateTransactionInfo body = null)

{

// verify the required parameter 'account' is set

if (account == null)

throw new ApiException(400, "Missing required parameter 'account' when calling VirtualTerminalApi->ApiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost");

// verify the required parameter 'password' is set

if (password == null)

throw new ApiException(400, "Missing required parameter 'password' when calling VirtualTerminalApi->ApiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost");

// verify the required parameter 'transactionID' is set

if (transactionID == null)

throw new ApiException(400, "Missing required parameter 'transactionID' when calling VirtualTerminalApi->ApiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost");

var localVarPath = "/api/v1/VirtualTerminal/UpdateTransactionInfo/{transactionID}";

var localVarPathParams = new Dictionary<String, String>();

var localVarQueryParams = new List<KeyValuePair<String, String>>();

var localVarHeaderParams = new Dictionary<String, String>(this.Configuration.DefaultHeader);

var localVarFormParams = new Dictionary<String, String>();

var localVarFileParams = new Dictionary<String, FileParameter>();

Object localVarPostBody = null;

// to determine the Content-Type header

String[] localVarHttpContentTypes = new String[] {

"application/json",

"text/json",

"application/\_\*+json"

};

String localVarHttpContentType = this.Configuration.ApiClient.SelectHeaderContentType(localVarHttpContentTypes);

// to determine the Accept header

String[] localVarHttpHeaderAccepts = new String[] {

"text/plain",

"application/json",

"text/json"

};

String localVarHttpHeaderAccept = this.Configuration.ApiClient.SelectHeaderAccept(localVarHttpHeaderAccepts);

if (localVarHttpHeaderAccept != null)

localVarHeaderParams.Add("Accept", localVarHttpHeaderAccept);

if (transactionID != null) localVarPathParams.Add("transactionID", this.Configuration.ApiClient.ParameterToString(transactionID)); // path parameter

if (account != null) localVarHeaderParams.Add("Account", this.Configuration.ApiClient.ParameterToString(account)); // header parameter

if (password != null) localVarHeaderParams.Add("Password", this.Configuration.ApiClient.ParameterToString(password)); // header parameter

if (body != null && body.GetType() != typeof(byte[]))

{

localVarPostBody = this.Configuration.ApiClient.Serialize(body); // http body (model) parameter

}

else

{

localVarPostBody = body; // byte array

}

// make the HTTP request

IRestResponse localVarResponse = (IRestResponse)await this.Configuration.ApiClient.CallApiAsync(localVarPath,

Method.POST, localVarQueryParams, localVarPostBody, localVarHeaderParams, localVarFormParams, localVarFileParams,

localVarPathParams, localVarHttpContentType);

int localVarStatusCode = (int)localVarResponse.StatusCode;

if (ExceptionFactory != null)

{

Exception exception = ExceptionFactory("ApiV1VirtualTerminalUpdateTransactionInfoTransactionIDPost", localVarResponse);

if (exception != null) throw exception;

}

return new ApiResponse<Response>(localVarStatusCode,

localVarResponse.Headers.ToDictionary(x => x.Name, x => string.Join(",", x.Value)),

(Response)this.Configuration.ApiClient.Deserialize(localVarResponse, typeof(Response)));

}