

OPC – An introductory guide to building payment applications



European
Commission

Horizon 2020
European Union funding
for Research & Innovation



Contents

Here you will find everything you need to get started with writing your Application:

1. A brief introduction to the OPE project
2. Steps to develop an OPC Application
3. An overview of the Payment Model pre-created by Ixaris
4. The APIs
5. Testing the Application
6. 'Hello World' example





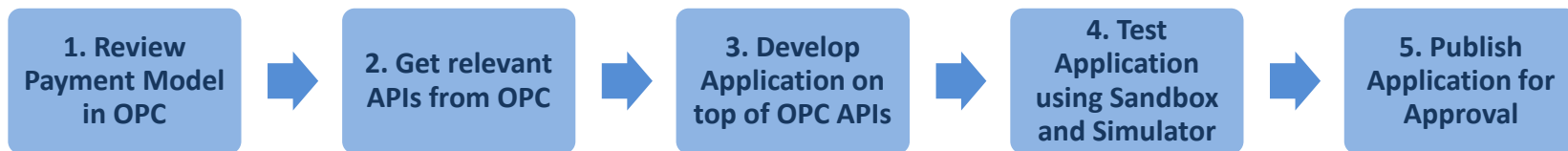
A brief introduction to OPE

- OPE is a two-year project, led by Ixaris and co-funded by the European Union that aims to open access to innovative and competitive payment services to SMEs in the EU.
- The aim of the project is to deliver the Open Payments Cloud (OPC) platform that brings together Application Developers like you, Programme Managers (that operate Applications on behalf of their SME and corporate clients) and Banks that ultimately power the underlying payment networks and infrastructure. This leads to an ecosystem of Developer, Programme Manager and Financial Service Provider stakeholders.
- In the spirit of Lean and Agile development, we have been engaging with Developers at various stages to ensure that we deliver a system that addresses the more important problems for the various stakeholders.
- To deal with the challenge of opening up access to financial systems we have had to come up with innovative concepts of how to enable Programme Managers and Financial Service Providers to trust Applications developed by third parties. To simplify things, we have done a bit of work behind the scenes so that you can spend most of your time focusing on the actual Application rather than figuring out the concepts.



Steps to developing a Payment Application in OPC

- In OPC, a developer develops their Application in their own preferred IDE, using their own preferred programming language and using whichever UI or API technology is most appropriate for their Application.
- Unlike other banking APIs, OPC does not publish a standard set of banking APIs for the developer to choose from – a developer is first expected to describe the functionality required by the Application they intend to develop and then the relevant APIs are made available by OPC. The Application description is called the Payment Model of an Application.
- Ixaris has pre-created an Application definition so that you, as the Developer, do not need to create one from scratch and we will provide you with the corresponding APIs directly.
- When development is complete, the Application can be tested using the OPC SandBox and the OPC Simulator and subsequently published for Approval.



The Payment Model defined by Ixaris

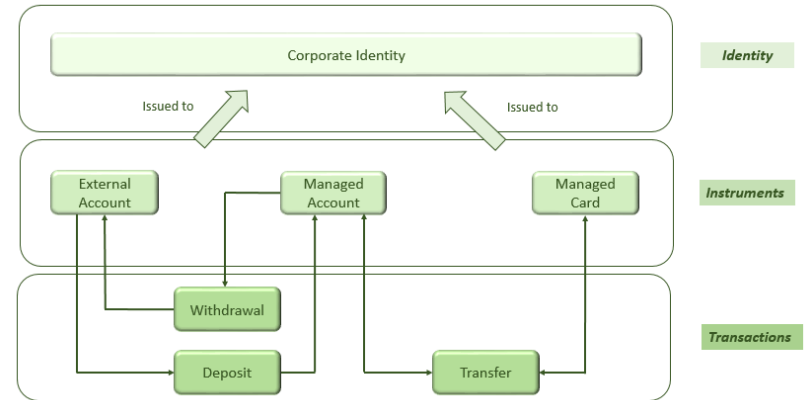
The illustration below shows the Payment Model pre-created by Ixaris. The model allows a Corporate to issue Virtual Cards and load these cards with funds from a corporate balance funded via bank transfer. Based on this Payment Model, the developer will get access to APIs that perform the following:

Manage entities of each type represented in the Payment model:

- Manage instances of Corporates (Corporate Identity)
- Manage instances of Virtual Cards (Managed Cards)
- Manage instances of Balances (Managed Accounts)
- Manage details of external bank accounts (External Account)

Move funds across instrument instances:

- From external bank account to internal balance (Deposit)
- From internal balance back to external bank Account (Withdrawal)
- From internal balance onto Virtual Card (Transfer)





Example Application using the defined Payment Model

Example Application: A developer would like to extend the use of SharePoint intranet by a Corporate to allow individuals to request and receive a Virtual Card to be used for corporate purchases (e.g. IT equipment)

The Developer will need to decide on the architecture of the Application, such as:

- Hard-coding the Application to work only with one corporate, or choose to develop a more generic variant that allows the Application to be used by multiple corporates
- Determine how best to implement workflow – use SharePoint’s workflow or to add it as a feature of the Application

The Developer can subsequently use the APIs to:

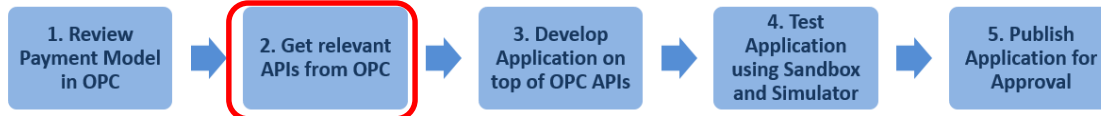
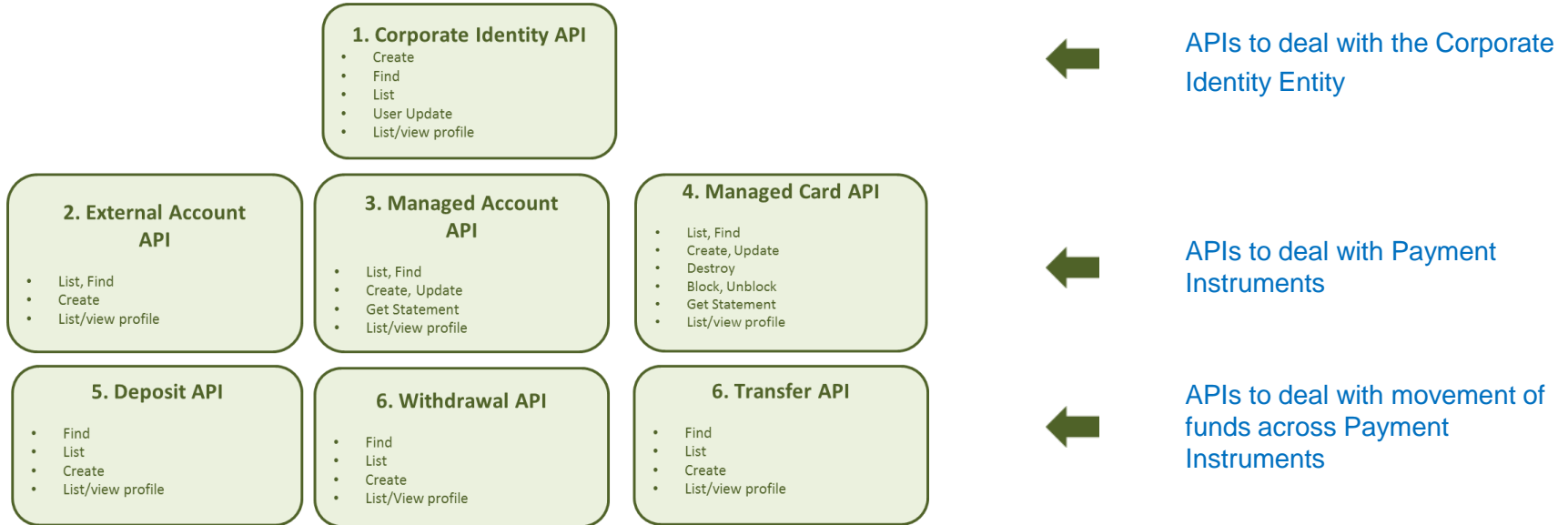
- Store information about a corporate on OPC
- Create one or more managed account linked to a corporate
- Allow an administrator to register an external bank account
- Process payments from a registered external bank account to a managed account
- Create a virtual card and assign it to an authorized employee
- Load a pre-approved balance on a virtual card
- Withdraw funds assigned to an employee for a specific purchase if transactions are not completed within period
- Generate a report with all purchases completed by employees, organized by department





The OPC APIs

OPC defines these APIs to match the Application's Payment Model – nothing more, nothing less. A developer will have access to these APIs:

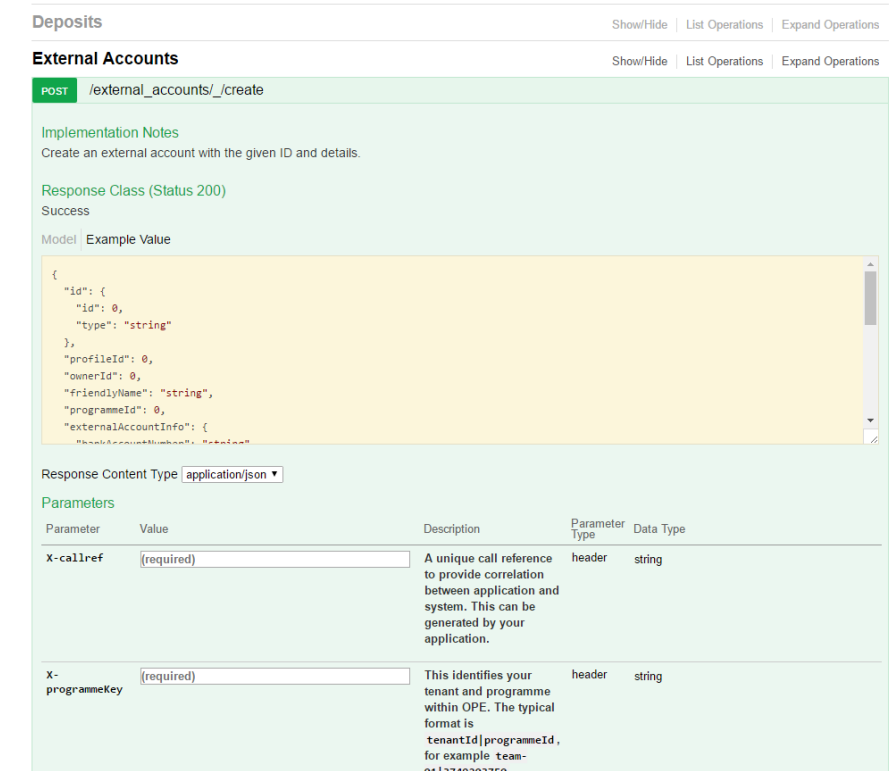


The OPC APIs

You can use **Swagger** to access the API methods:

Instructions for accessing our APIs using Swagger are available on the website.

You can refer to the Sample Application, or find them directly via the Swagger link.



Deposits Show/Hide List Operations Expand Operations

External Accounts Show/Hide List Operations Expand Operations

POST /external_accounts/_/create

Implementation Notes
Create an external account with the given ID and details.

Response Class (Status 200)
Success

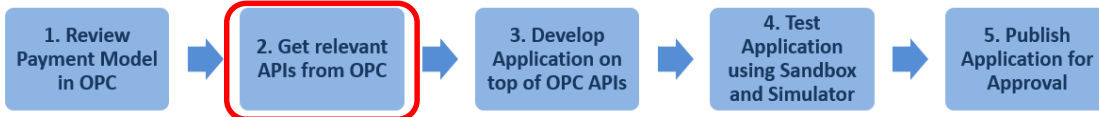
Model: **Example Value**

```
{
  "id": {
    "id": 0,
    "type": "string"
  },
  "profileId": 0,
  "ownerId": 0,
  "friendlyName": "string",
  "programmeId": 0,
  "externalAccountInfo": {
    "bankAccountNumber": "string"
  }
}
```

Response Content Type:

Parameters

Parameter	Value	Description	Parameter Type	Data Type
X-callref	<input type="text" value="(required)"/>	A unique call reference to provide correlation between application and system. This can be generated by your application.	header	string
X-programmeKey	<input type="text" value="(required)"/>	This identifies your tenant and programme within OPE. The typical format is tenantId programmeId, for example team-01 1234567890	header	string



The OPC APIs

To integrate with the API you will need to set up an API client locally.

The client libraries are responsible for creating and sending the actual HTTP request and handling the mapping to the appropriate response type

And here is an example create card request and subsequent transfer of 10 GBP on to the card. Please note, you will need to use the hard coded values (circled) in the request

```
/**
 * An example client making use of the OPE APIs to create a card and transfer some funds on it.
 */
public class ExampleClient {

    // Instrument types required for performing transactions
    private static final String MANAGED_CARD_TYPE = "managed_cards";
    private static final String MANAGED_ACCOUNT_TYPE = "managed_accounts";
    private static final String EXTERNAL_ACCOUNT_TYPE = "external_accounts";

    // Configuration settings - replace the following values with your own
    private static final String API_BASE_PATH = "http://localhost:8324/api";
    private static final String USERNAME = "user";
    private static final String PASSWORD = "password123!";
    private static final String TENANT_ID = "system";
    private static final String PROGRAMME_ID = "1";

    // The format of the programme key is as follows: '%tenant_id|%programme_id'
    private static final String PROGRAMME_KEY = String.format("%s|%s", TENANT_ID, PROGRAMME_ID);

    // Set the base path of the API
    final ApiClient client = new ApiClient().setBasePath(API_BASE_PATH);
    final DefaultApi api = new DefaultApi(client);

    // Login with your credentials to get an Authorisation token
    final LoginParams loginRequest = new LoginParams().credentialCode(USERNAME).password(PASSWORD).programmeId(PROGRAMME_ID);
    final LoginResult loginResult = api.authLogin(generateCallRef(), PROGRAMME_KEY, loginRequest);

    // Create the authorisation header in the following format: 'X-TOKEN %auth_token%'
    final String authHeader = String.format("X-TOKEN %s", loginResult.getToken());

    // Create a managed card instrument using an existing profile
    final CreateManagedCardParams createCardRequest = new CreateManagedCardParams()
        .profileId(cardProfileId)
        .ownerId(cardOwnerId)
        .friendlyName("John's GBP card")
        .currency("GBP")
        .issuingProvider("Issuing Provider")
        .processingProvider("Processing Provider")
        .nameOnCard("John Doe");
    final ManagedCard card = api.managedCardsIdCreate(generateCallRef(), PROGRAMME_KEY, authHeader, createCardRequest);

    // Transfer 10 GBP from an existing managed account instrument to the newly created card
    final CreateTransferParams createTransferParams = new CreateTransferParams()
        .profileId(transferProfileId)
        .amount(new CurrencyAmountMessage().currency("GBP").amount("1000")) // 10.00 GBP
        .sourceInstrumentId(new TypedId().type(MANAGED_ACCOUNT_TYPE).id(managedAccountId))
        .destinationInstrumentId(new TypedId().type(MANAGED_CARD_TYPE).id(card.getId()));
    api.transfersIdCreate(generateCallRef(), PROGRAMME_KEY, authHeader, createTransferParams);
}
```

1. Review Payment Model in OPC

2. Get relevant APIs from OPC

3. Develop Application on top of OPC APIs

4. Test Application using Sandbox and Simulator

5. Publish Application for Approval

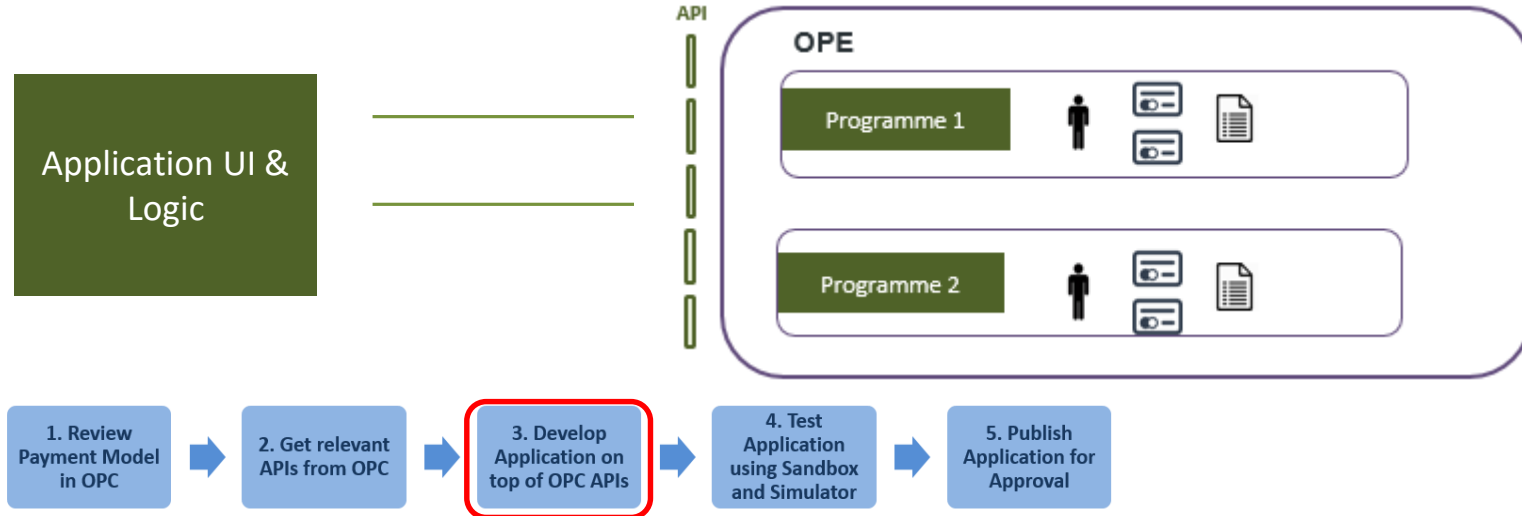


Dealing with multiple clients through Programmes

An important consideration when designing an Application is to determine how an Application deals with multiple clients. After the developer completes development and testing, a Programme Manager will use the Application for one or more clients.

To do so, the Programme Manager creates a Programme for a client and defines all the client-specific configuration in that Programme.

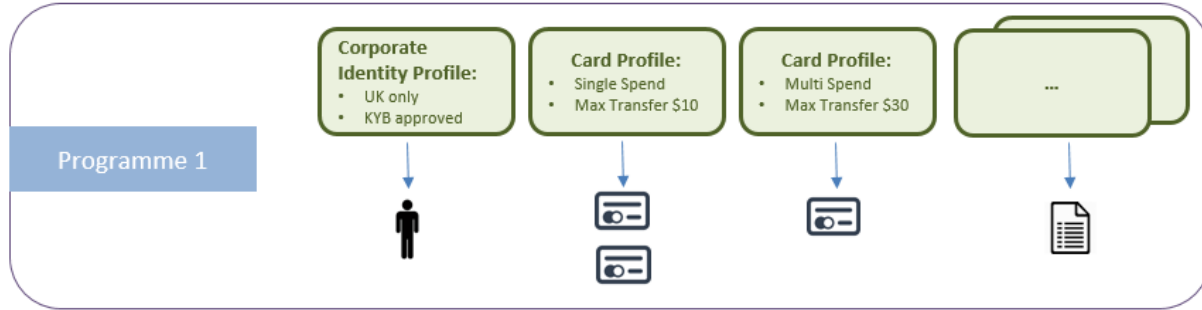
Across all API method invocations, the developer has to specify the Programme upon which the operation is to take place.





Pre-defined Profiles

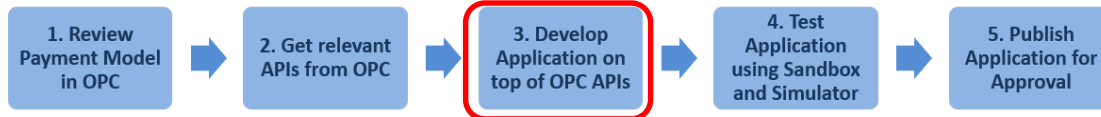
You will observe that some of the API methods have a parameter called 'Profile'. A Profile consists of a group of properties that define a type of the requested instance, be it an identity, a payment instrument or a transaction. Multiple profiles can be defined for a Programme:



The developer specifies the Profile name for the type of Virtual card that is to be created.

Profiles cannot be created on-the-fly: this is not a limitation of the APIs or OPC. Typically configurations need to be pre-approved by the Programme Manager and it is important that the developer is not allowed to create new unapproved types of cards without authorisation.

Ixaris has pre-created a set of Profiles - these Profiles can be updated and new ones added from the OPC SandBox when the Developer is testing the Application.





Testing the Application in the OPC SandBox

An Application is to be tested in the context of a specific Programme that defines specific Profiles.

The OPC SandBox is a portal that enables the developer to create Programmes, define Profiles and to check instances (of identities, cards, transactions etc.) to confirm expected behaviour.

A test Programme, with test Profiles, has been created by Ixaris for testing purposes.

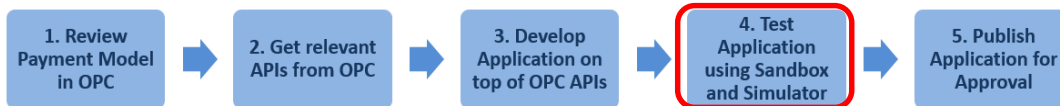
Below is the view of the sandbox that shows a list of profiles that have been set up, and the actions available from a given profile:

The screenshot shows the 'OPEN PAYMENTS CLOUD' interface. At the top left is the logo. Below it is a navigation bar with 'Configure', 'Manage', and 'Reports'. The main content area is titled 'Profiles' and contains a table with the following data:

NAME	PAYLET TYPE	PAYLET NAME	STATE	TAGS
Profile 97593089101271040	Withdraw	WITHDRAWAL	ACTIVE	
Profile 97593089101270016	Deposit	DEPOSIT	ACTIVE	
Profile 97593089101269760	Transfer	TRANSFER	ACTIVE	
Profile 97593089101269504	External Account	EA	ACTIVE	

A context menu is open over the 'Deposit' profile, showing three options: 'Manage Profile', 'View Programme', and 'View Instances'. A yellow arrow points from the text 'Navigate and search for instances' to the 'View Instances' option.

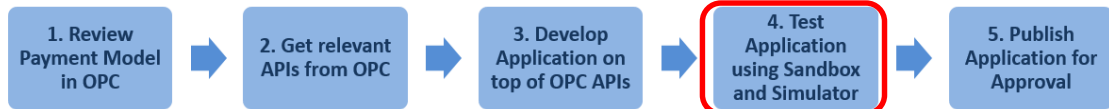
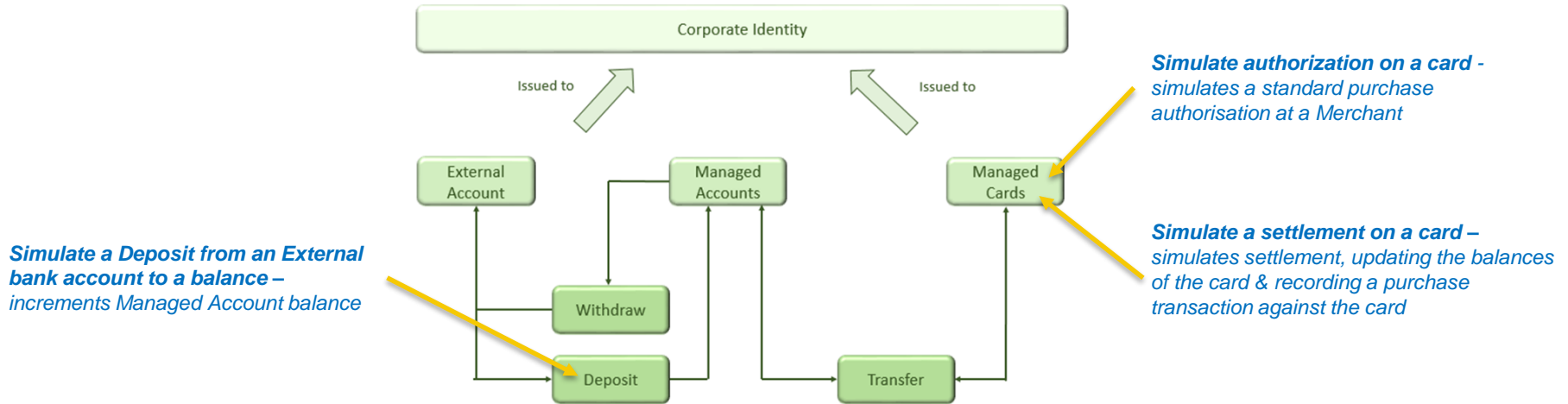
Navigate and search for instances



Using the OPC Simulator

There are a number of functions related to an Application that cannot be tested through the API – primarily these are interactions and data flows that involve external 3rd party services.

The OPC Simulator is a testing tool that allows for the simulation of these external events. The following are the functions covered through the Simulator





'Hello World' Application

Now that you are familiar with the payment model we can take a look at an example application, showing how you can use OPC to issue a virtual card and load it with funds in order to perform a purchase (it assumes the corporate identity has been created and there are available funds in the managed account). Go to Swagger and try the following operations:

