Application provisioning protocol

Introduction

This document specifies the *provisioning protocol*, that is the say the various exchanges between the OASIS Platform and the Applications that start when a Manager of an organization subscribes (or “buys”) an application and ends when the application instance is created and its various services are declared in the catalog.

Definitions

Application — an abstract application, not directly useable, declared in the catalog and visible in the app store, that can be the object of an instantiation. For instance, “Ushahidi” is an application.

Application instance — a runnable copy of an application, created within the context of an identified authority (which may be an individual or an organization, for instance the “buying organization” or the application provider).

Service — an “endpoint” of an application instance, which is addressable through a URL. Services are declared in the catalog and may be visible in the app store. The icons visible in a user’s dashboard on the Portal are Services. An application instance has at least one and possibly many Services.

The Catalog — the catalog is a database of all applications, instances and services. It is used internally by the Portal to display a user’s dashboard, to browse the app store, etc.

The App Store — a system that allows users to :

- find Services that are publicly available and add them to their Dashboards,
- find Applications that are instantiable and request the creation of an instance in the context of their organization.

App Factory — a REST service ***provided by application providers*** that the Platform calls to initiate application instantiation.
Important note: the App Factory endpoint is used to provide critical security information to the application (for instance, client ID / client secret). We therefore mandate the use of SSL or TLS security for that endpoint.

Protocol overview

Step 0 : declare your application in the Catalog

(This step will be done manually by Atol/Open Wide for the prepilot / start of the pilot phase)

The application must exist in the catalog to be instantiated.
The exact information required is annexed to this document, but in particular the Catalog contains :

- the App Factory's URL
- an instantiation_secret string used to sign instantiation requests
    - this should not be confused with client_secret. The instantiation_secret is application-wide and is used to authorize the Kernel to ask for new instances to the App Factory. A client_secret (and in particular the client_id) is always related to a particular app instance.
- a cancellation_uri and cancellation_secret used to cancel pending instantiations.

Step 1 : a Manager buys the application

The platform verifies that the user is indeed allowed to act on behalf of their organization, then generates :

- a unique instance_id
- a unique client_id
- a client_secret

It then issues a POST request to the App Factory's URL, providing :

- instance_id
- client_id
- client_secret
- user (representing the Manager, with its OASIS identifier and display name)
- organization (*optional* – the Organization on behalf of which the Manager is acting, with its OASIS identifier, name and type)
- instance registration URL
- *deprecated:*
    - user_id (OASIS identifier of the Manager)
    - organization_id (*optional* – OASIS identifier of the Organization on behalf of which the Manager is acting)
    - organization_name (*optional* – name of that organization)

This content is encoded in JSON. A HMAC-SHA1 hash of the JSON string is computed using the application's instantiation_secret as key, and inserted in the X-Hub-Signature header as per https://pubsubhubbub.googlecode.com/git/pubsubhubbub-core-0.4.html#authednotify.

This reasonably certifies that the request does come from the OASIS Kernel (which is the only one to know the instantiation_secret string).

Example request (with the signature mocked out for now):

```
POST /admin/create-instance HTTP/1.1
Accept: application/json, application/*+json
X-Hub-Signature: sha1=** HMAC-SHA1 digest of payload **
Content-Type: application/json;charset=UTF-8
Content-Length: 284
Host: localhost:9090
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Accept-Encoding: gzip,deflate

{"instance_id":"8f814322-68ac-4fa3-87a8-e4e8d28f5706",
 "client_id":"41184194-d40b-4720-87a9-284d2fa9d5ed",
 "client_secret":"41184194-d40b-4720-87a9-284d2fa9d5ed",
 "user": {"id":"a399684b-4ea3-49c3-800b-b8a0bf1131cb",
      "name": "John Doe", email_address: "jd@example.org"},
 "user_id":"a399684b-4ea3-49c3-800b-b8a0bf1131cb",
 "organization":{"id"::"a2342900-f9eb-4d54-bf30-1e0d763ec4af",
      "organization_name": "Valence", "type": "PUBLIC_BODY"},
 "organization_id":"a2342900-f9eb-4d54-bf30-1e0d763ec4af",
 "organization_name": "Valence",
 "instance_registration_uri":
"https://kernel.ozwillo.eu/apps/pending-instance/8f814322-68ac-4fa3-8
7a8-e4e8d28f5706"}
```

The Factory replies with either:

- 2xx: request acknowledged, will do the job
- any 4xx error: request cannot be honoured (for instance, the organization already possesses an instance of this application and it is limited to one)
- any other non-2xx will be declared a failure too (maybe we'll follow redirect, but let's say for now that we won't).

Step 2: provision the application

Application providers are free to implement this step as they will. It can be:

- fully automatic
- half automatic (requires a manual validation)
- fully manual

Or anything in between.

Note that the new instance *MUST* use the client_id / client_secret it's been provided with for authentication.

Step 3: acknowledge the result

Application providers must POST to the Kernel's Application Instantiated service, providing:

- instance_id (the one that's been provided by the Kernel)
- a destruction URI and secret
- a list of Services that the application wants created
- a list of scopes that the application wants defined (scopes needed to use the application's own exposed APIs)
- a list of scopes that the application's Services will use, each with a motivation.

For each Service, the following information is provided (the full list is annexed to this document):

- a local identifier (for instance "back office")
- the Service's URL and notification URL (see annex for the difference between the two)
- its visibility status: "true" means the service is visible in the app store, "false" means it isn't. Also, when visible=false, the Kernel will prevent access to the Service from users that have not been explicitly allowed to use the instance by the Manager in the portal. Typically a public-body-only back office will have visible=false, while a public service will have visible=true. A public service might however have visible=false while being configured, or for *dogfooding*, before being made publicly accessible (and visible=true).
- whether the service is has restricted access: "true" means that the application cannot be made visible. Typically a public-body-only back office will have restricted=true, while a public service will have restricted=false.
- the name of the service (in all OASIS-supported languages)
- the description of the service (in all OASIS-supported languages)
- the app store metadata: categories, territory tag, payment option, target audience. These fields are only relevant for visible=true services
- redirect_uris: whitelist of values used as redirect_uri during the authentication; **values must be different for each service within an application instance.**
- post_logout_redirect_uris: whitelist of values used as post_logout_redirect_uri during single sign-out (optional if post_logout_redirect_uri is not used)

For each defined Scope, the following information is provided (the full list is annexed to this document):

- an identifier (for instance "addevent", the namespace for scopes is the application instance)
- the name of the scope (in all OASIS-supported languages; for instance "add Ushahidi events")
- the description of the scope (in all OASIS-supported languages)

Scope identifiers are a simple string that corresponds to the permission scopes that client applications require to access part of this API's functionality. For instance, application "Ushahidi" may define an API to add new events on a map; it would then register the scope "addevent". When a third party application wants to add an event on behalf of a user, it requires the "<instance_id>:addevent" scope (where "<instance_id>" is the instance ID of the application instance) as part of its token negotiation (the user is asked to confirm that they want to give the "add Ushahidi events" permission to the app).

Finally, for each needed Scope, the following information is provided (the full list is annexed to this document):

- The scope's full identifier (e.g. "address", "email", "phone" or "c75aa12d-2f78-4fc5-b935-2945362f05a1:addevent")
- the motivation for its use (in all OASIS-supported language). This will be displayed to the user when he'll be asked to authorize the application so he knows why the application wants those scopes and can therefore make an informed choice.

Applications need not give a motivation for each and every scope they'll use. If they don't though, the user will have no way of knowing how the scopes will be used by the application and why it needs them.

The Kernel responds with an object mapping, for each service identifier, its unique id in the Catalog.

Example request (this instance creates three Services: back, front, and electoral_roll_registration).

```
POST /apps/pending-instance/8f814322-68ac-4fa3-87a8-e4e8d28f5706
HTTP/1.1
Accept: application/json, application/*+json
Authorization: Basic
NDExODQxOTQtZDQwYi00NzIwLTg3YTktMjg0ZDJmYTlkNWVkOjQxMTg0MTk0LWQ0MGItN
DcyMC04N2E5LTI4NGQyZmE5ZDVlZA==
Content-Type: application/json;charset=UTF-8
Content-Length: 3198
Host: kernel.ozwillo.eu
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.3.1 (java 1.5)
Accept-Encoding: gzip,deflate

{"services":[
{"local_id":"back",
 "service_uri":"http://localhost:9090/back/valence",
 "redirect_uris": [
"http://localhost:9090/back/valence/oasis_profile_callback" ],
 "visible":false,
 "restricted":true,
 "name":"Gestion des procédures citoyennes",
 "description":"Outil de traitement des des procédures citoyennes en
cours",
 "icon":"http://www.whatever.com",

"notification_uri":"http://localhost:9090/back/valence/notifications"
,
 "category_ids":[],
 "payment_option":"PAID",
 "target_audience":["PUBLIC_BODIES"],
 "territory_id":"26000",
 "name#it":"Citizen Procedures for Valence",
 "name#en":"Citizen Procedures for Valence",
 "name#es":"Citizen Procedures for Valence",
 "name#fr":"Procédures citoyennes de Valence",
 "description#bg":"Citizen procedures for Valence",
 "description#tr":"Citizen procedures for Valence",
 "description#ca":"Citizen procedures for Valence",
 "name#tr":"Citizen Procedures for Valence",
 "name#ca":"Citizen Procedures for Valence",
 "description#it":"Citizen procedures for Valence",
 "description#fr":"Portail de dématérialisation de procédures pour
Valence",
 "description#en":"Citizen procedures for Valence",
 "name#bg":"Citizen Procedures for Valence",
 "description#es":"Citizen procedures for Valence"
},
{"local_id":"front",
 "service_uri":"http://localhost:9090/front/valence",
 "redirect_uris": [
"http://localhost:9090/front/valence/oasis_profile_callback" ],
 "visible":true,
 "name":"Procédures citoyennes de Valence",
 "description":"Portail de dématérialisation de procédures pour
Valence",
 "icon":"http://www.whatever.com",

"notification_uri":"http://localhost:9090/front/valence/notifications
",
 "category_ids":[],
 "payment_option":"FREE",
 "target_audience":["CITIZENS"],
 "territory_id":"26000",
 "name#it":"Citizen Procedures for Valence",
 "name#en":"Citizen Procedures for Valence",
 "name#es":"Citizen Procedures for Valence",
 "name#fr":"Procédures citoyennes de Valence",
 "description#bg":"Citizen procedures for Valence",
 "description#tr":"Citizen procedures for Valence",
 "description#ca":"Citizen procedures for Valence",
 "name#tr":"Citizen Procedures for Valence",
 "name#ca":"Citizen Procedures for Valence",
 "description#it":"Citizen procedures for Valence",
 "description#fr":"Portail de dématérialisation de procédures pour
Valence",
 "description#en":"Citizen procedures for Valence",
 "name#bg":"Citizen Procedures for Valence",
 "description#es":"Citizen procedures for Valence"
},
{"local_id":"electoral_roll_registration",
 "service_uri":
"http://localhost:9090/front/valence/form/electoral_roll_registration
/init",
 "redirect_uris": [
"http://localhost:9090/front/valence/form/electoral_roll_registration
/oasis_profile_callback" ],
 "visible":true,
 "name":"Pré-inscription sur liste électorale",
 "description":null,
 "category_ids":[],
 "payment_option":"FREE",
 "target_audience":["CITIZENS"],
 "territory_id":"26000",
 "name#it":"Pré-inscription sur liste électorale",
 "name#en":"Pré-inscription sur liste électorale",
 "name#tr":"Pré-inscription sur liste électorale",
 "name#ca":"Pré-inscription sur liste électorale",
 "name#es":"Pré-inscription sur liste électorale",
 "name#fr":"Pré-inscription sur liste électorale",
 "name#bg":"Pré-inscription sur liste électorale"
}
],
"scopes": [
{"scope_id": "ck_files",
 "name": "Pièces jointes administratives pour Valence",
 "description": "Fichiers joints aux formulaires pour la Ville de
Valence"
}
],
"needed_scopes": [
{"scope_id": "profile",
 "motivation": "Utilisé pour pré-remplir les formulaires",
 "motivation#fr": "Utilisé pour pré-remplir les formulaires",
 "motivation#en": "Used to auto-fill form fields"
},
{"scope_id": "email",
 "motivation": "Utilisé pour pré-remplir les formulaires",
 "motivation#fr": "Utilisé pour pré-remplir les formulaires",
 "motivation#en": "Used to auto-fill form fields"
},
{"scope_id": "address",
 "motivation": "Utilisé pour pré-remplir les formulaires",
 "motivation#fr": "Utilisé pour pré-remplir les formulaires",
 "motivation#en": "Used to auto-fill form fields"
},
],
"instance_id":"8f814322-68ac-4fa3-87a8-e4e8d28f5706",
"destruction_uri":"http://localhost:9090/admin/drop-instance",
"destruction_secret":"78L0C3RKq6ovP0rXAp6FOd5UXG70YpC56enl3If5DIe"
}
```

And the response:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Thu, 03 Jul 2014 14:57:41 GMT

{"back":"a15243e3-17a0-4511-b15c-c88e6784e287",
 "front":"31336385-f2ff-4488-8835-1f7da53669b9",
 "electoral_roll_registration":"28e513d2-881a-4856-a542-0f09889cfb6a"
}
```

At this stage:

- the application instance is created

- its services are declared in the catalog
- its scopes are declared and available for use by other applications
- the Manager is automatically subscribed to all private services
- the public services are visible in the app store (and available for use by users)

The next stages will be for the Manager to use the App management tab in the Portal to assign the back office services to organization users, and to switch on / off the public services (for instance most applications will declare their services as private, to let the manager do basic customization before going live towards the public).

If the instance creation has failed for some reason, the App Factory **must** issue a `DELETE` request on the instance registration URL (in the example: `/apps/pending-instance/8f814322-68ac-4fa3-87a8-e4e8d28f5706`) so that the platform does not show endless “pending” instance creation requests.

Step 4 (optional): subscription callback

NOT IMPLEMENTED YET

Individual Services may register a callback URL that will be notified (through POST + signature as above) when a user adds the application to their dashboard.
The Kernel provides a subscription id along with the user and organization id; if the service provides this URL, it **must** call the subscription-created service to approve (or deny) the subscription.

Instance destruction

When a user or organization no longer needs an instance, they can destroy it. The Kernel then calls the registered destruction_uri in a manner similar to the way it called the App Factory.

The Kernel issues a POST request to the destruction URI providing the instance_id of the instance to be destroyed (this allows sharing the same destruction URI and secret by several instances, but the Kernel also supports having separate destruction URI and secret for each instance).

Example request:

```
POST /admin/drop-instance HTTP/1.1
Host: localhost:9090
X-Hub-Signature: sha1=** HMAC-SHA1 digest of payload **
Content-Type: application/json;charset=UTF-8

{"instance_id": "8f814322-68ac-4fa3-87a8-e4e8d28f5706"}
```

The destruction URI must respond with a successful status (200, 202 or 204) in a timely manner. The Kernel will then delete the instance from its database and it will be impossible for users to authenticate to it.
If the request times out, the Kernel will delete the instance from its database nevertheless.
Any (timely) non-successful status will abort the destruction (so it can be retried later).

Pending instances (those for which a request has been made to the App Factory, but the instance_registration_uri hasn't been called back yet) can be cancelled using the same mechanism, but at the application-wide cancellation_uri. If that URI is different from provisioned instances' destruction_uri, it **should** reject requests to destroy provisioned instances (and the instance-specific destruction_uri should be the only way to destroy the instance).

Appendices

Appendix 1: catalog information

The following are required for catalog entries (applications):

Field name	Field description
name	Name in the default language (try to remain below 20 characters)
name#<language> (ex: name#tr)	Name in the given language (optional, try to remain below 20 characters)
description	Description in the default language (at least 300 characters)
description#<Language> (ex: description#tr)	Description in the given language (optional, at least 300 characters)
tos_uri	Terms of service URI implicitly accepted on purchase, in the default language
tos_uri#<Language> (ex: tos_uri#fr)	Terms of service URI implicitly accepted on purchase, in the given language
policy_uri	Privacy Policy URI implicitly accepted on purchase, in the default language

policy_uri#<Language> (ex: policy_uri#fr)	Privacy Policy URI implicitly accepted on purchase, in the given language
icon	URL of the app icon in the default language (expected size: 64px x 64px)
icon#<Language> (ex: icon#tr)	URL of the app icon in the give language (optional)
screenshot_uris	list of screenshot URLs (expected size: 800px x 450px)
contacts	list of URLs (or mailto) to contact/support
category_ids	IDs of the app store categories
payment_option	for app store: FREE or PAID
target_audience	for app store: list of CITIZENS, PUBLIC_BODIES, COMPANIES
instantiation_uri	Instantiation endpoint, URL of the App Factory
instantiation_secret	Secret used to compute the instantiation request signature
cancellation_uri	Cancellation endpoint, to cancel pending instantiations.
cancellation_secret	Secret used to compute the cancellation request signature.
visible	if false, the application is not visible in the app store (can be used for single-instance apps)
provider_id	ID of the application provider organization

Appendix 2: JSON messages

Kernel -> App Factory (Create instance request)

Field name	Description

instance_id	ID of the instance created
client_id	For authentication
client_secret	For authentication
user_id	Id of the user who “bought” the app **DEPRECATED**
user	Object representing the user who “bought” the app (contains his ID, name and email address). Scheduled to replace user_id.
organization_id	(optional) ID of the organization (if any) for which the app has been “bought”. **DEPRECATED**
organization_name	(optional) Name of that organization **DEPRECATED**
organization	(optional) Object representing the organization (if any) for which the app has been “bought” (contains its ID, name and type). Scheduled to replace organization_id and organization_name.
instance_registration_uri	For acknowledgement

App factory -> Kernel (Instance created)

Field name	Description
instance_id	
services	List of Service objects (there must be at least one service)
scopes	List of Scope objects
needed_scopes	List of NeededScope objects
destruction_uri	Destruction endpoint for the instance

destruction_secret	Secret used to compute the destruction request signature

Service object:

Field name	Description
local_id	Internal identifier (eg “front office”)
name	Name in the default language
name#lang	Name in language “lang” (two-letter code, eg es, ca…)
description	Description in the default language
description#lang	Description in language “lang”
tos_uri	Terms of service URI implicitly accepted on purchase, in the default language
tos_uri#lang	Terms of service URI implicitly accepted on purchase, in the given language
policy_uri	Privacy Policy URI implicitly accepted on purchase, in the default language
policy_uri#lang	Privacy Policy URI implicitly accepted on purchase, in the given language
icon	URL of the icon in the default language
icon#lang	URL of the icon in language “lang”
screenshot_uris	list of screenshot URLs
contacts	list of URLs (or mailto) to contact/support
category_ids	list of app store category ids (to be defined)
payment_option	“FREE” or “PAID”
target_audience	list of target audiences (eg [“CITIZENS”, “COMPANIES”])
territory_id	ID of a territorial tag in the data core

visible	true = public service, false = private service; defaults to false. When a service is private, the Kernel restricts access to members (app_user / app_admin); typically used for *dogfooding* a service.
restricted	true = the service restricts access to members (app_user / app_admin). A restricted service cannot be made visible; typically used for back office services. Defaults to false.
service_uri	Main URL of the service
notification_uri	(optional) URL used when there are notifications on the service
redirect_uris	Whitelist of authentication callbacks
post_logout_redirect_uris	(optional) Whitelist of post-logout callbacks
subscription_uri	(optional) URL of the Subscription Callback **NOT IMPLEMENTED YET**

Scope object:

Field name	Description
local_id	“Local” identifier for the scope (local to the instance)
name	Name of the scope in the default language
name#lang	Name of the scope in language “lang”
description	Description of the scope in the default language
description#lang	Description of the scope in language “lang”

NeededScope object:

Field name	Description
scope_id	

motivation	Motivation for using the scope, in the default language
motivation#lang	Motivation for using the scope, in language “lang”

Kernel -> Service (subscription callback)

NOT IMPLEMENTED YET

Field name	Description
subscription_id	
user_id	Id of the user
organization_id	ID of the user’s organization (if any)
organization_name	Name of that organization
service_id	ID of the service
subscription_management_uri	For acknowledgement

Service -> Kernel (subscription created)

NOT IMPLEMENTED YET

POST {subscription_management_uri} => approve the subscription
DELETE {subscription_management_uri} => deny the subscription

Appendix 3: integration with the rest of the system

User access:
Users (end users and administrators alike) always reach your application through a service. So you have to declare at least one service.
In the Portal, Managers declare a whitelist of users allowed to use the private services of an application instance. The authentication system will block all other users from accessing the service. Public services (with visible=true) are not subject to such blocking.
Applications can declare services as having restricted access (restricted=true) to signal that they will never allow a user that has not been whitelisted to access it. Such services cannot be made public.

Authentication:

Authentication requests **must** be done using the client_id and client_secret provided by the Kernel during instance creation, and **must** use the redirect_uri provided during service creation.
At the end of the authentication process, the Kernel informs your application if the logged-in user is a regular user, an application administrator, both, or none (can only be the case for public services).

Notifications:
If your application sends notifications, it is required to use (as much as possible) the service id matching the user's action - this is used in the Portal to display notification badges on the correct icons.

Appendix 4: "degenerate" cases

What if your application scenario doesn't fit the idealized process?

- Single instance applications
    - it's best if you can obtain a different client_id for each client (municipality…) so it can fit in the model here
    - otherwise what you pre-publish is not an application but directly a service and you use the Subscription callback to be notified / create permissions to the users.
- Other cases?