Author: Mats Agerstam
Email: mats.g.agerstam@intel.com

IoTivity – Protocol Plugin Manager

Authorization of Plugins

The Open Interconnect Consortium (OIC) defines a comprehensive security architecture to ensure that authorized subjects and devices in the OIC ecosystem can communicate securely with a standardized defined model for access control, policy enforcement polices etc. This document focuses on authorization flow of Protocol Plugins to the PPM.

One of the basic promises of OIC is standardization around the discovery, connectivity, security and data model level. To increase interoperability for non-OIC compliant devices, a protocol plugin manager has been defined that enables for interoperability at the data or resource model layer.

A key aspect when it comes to these plugins is how authorized is accomplished. In other words, how the end-user grants permission to a plug-in to interact with the non-OIC service(s) it can interact with. The main focus on this paper is to:

- ✓ *Define how this can be accomplished within the context of the data model in OIC*
- ✓ *How various authentication and authorization schemes can be supported with this model*
- ✓ *Life cycle management and updates of the plugins.*

Introduction to PPM

The PPM is a software component in IoTivity that enables developers to bring in non-OIC technologies to the OIC eco-system by supporting the necessary translation or bridging functionality. Bridging/translation is used loosely here as might happens on several layers for a particular technology

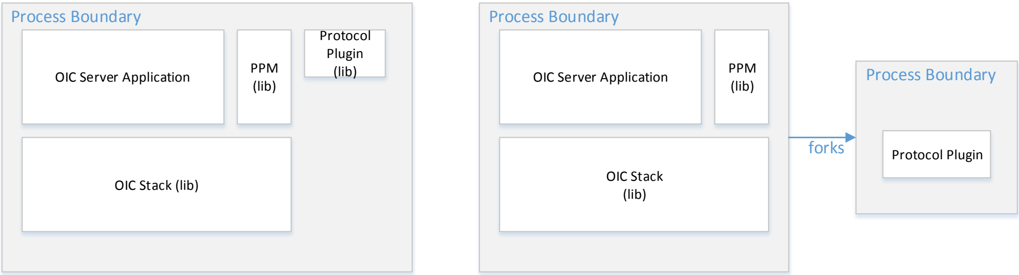
- Data or resource model
- Discovery and Connectivity
- Security

A non-OIC device would likely to differ from an OIC device in the categories outlined above. These protocol converters, implemented as plugins, are used to translate OIC data model and protocol to the non-OIC device-specific protocol and are managed by the Protocol Plugin Manager (PPM).

As an example, consider two smart light bulbs manufactured by different vendors – one bulb is OIC-compliant while the other is not (e.g. Hue, manufactured by Phillips, can be controlled over HTTP and is not OIC-compliant). The responsibility for a Hue plugin would be to register found Hue lights as generic OIC lights to IoTivity. Standard OIC lights and Hue lights can now be managed homogenously by a client that is only aware of OIC lights. The Hue plugin is responsible for converting between the OIC and Hue.

OIC is defining a plugin specification that standardizes certain minimum requirements for plugin so that plugins implemented by a 3rd party developer can be managed by PPM. A plugin will operate seamlessly with the Iotivity stack. To enable multiplatform support, Iotivity supports plugin written in C/C++ and Java. To manage plugins, PPM maintains a repository of available plugins, validates authenticity of plugin, dynamically loads/unloads plugin, plugin memory management, and handles plugin updates.

The figure below illustrates two deployment scenarios supported by the PPM.



In-process

Out-of-process

The OIC server application, or a dedicated PPM process is responsible for enumerating the available plugins and loads them one at time. In the first model the PPM process hosts all of the available plugins in the address space of the PPM itself. This is suitable for environments where the additional overhead of spawning dedicated processes for plugins is not suitable.

In the second approach the PPM process enumerates available plugins and spawns a dedicated process for each plugin. The PPM has a dedicated IPC channel to each plugin but they are process bounded from each other which provides a good isolation between individual plugins and the PPM.

The PPM only loads plugins that are deemed valid. A valid plugin must meet the following minimum criteria dictated by the PPM.

- ✓ Must reside in a predefined directory for PPM to find it.
- ✓ Must implement the mandatory interface and expose the symbols for the PPM to explicitly load it.
- ✓ A plugin manifest must be present for the plugin. Among other things the manifest provides details about versioning, which OIC resource types the plugin translates between etc.
- ✓ Plugin must have been signed by a trusted Certificate Authority (CA) on the local platform.

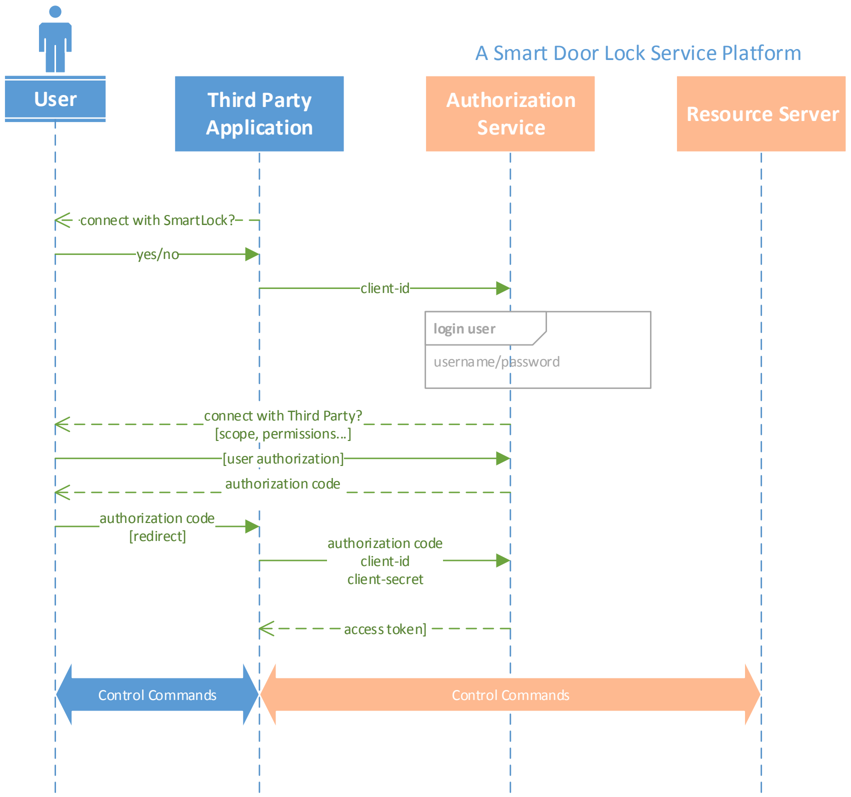
General Authorization Steps

This chapter outlines a general high level view of the services in play when it comes to granting a third-party application or component access to a service using *some* authorization protocol (e.g. OAuth or similar). Manufacturers implement different mechanisms that forces different levels of impact with regards to user experience but also security in terms of:

- ✓ Authentication protocol and Credentials
- ✓ Scope and Access Control Schema
- ✓ Validity period

A local light system that only provides access from within the home on a WiFi network might have less requirements on security than a smart door lock that can be accessed and controlled remotely by people.

In the diagram below the end user has a smart door lock and has also installed a separate application from a third-party developer not directly associated with the manufacturer of the door lock. The application has an option to connect the application with the smart lock service infrastructure. The following depicts the essential flows of the interaction between end user, application and door lock service infrastructure.



- ➢ The end user starts the TPV (Third Party Vendor) application and chooses to connect the application with the smart door lock infrastructure.
- ➢ The application loads the content of the authorization URL from the smart door lock vendor and also requests what type of permissions the application needs (e.g. monitor door state, lock the door, manage keys, etc.). The URL typically points to a web service on the cloud.
- ➢ The authorization will prompt the user to log in to the providers backend service, if he/she has not already logged in to their cloud.
- ➢ When the user has logged in, a permission screen will be displayed to the end user with exactly the scope (permissions) the application is requesting.
- ➢ If the end user grants permissions to the application an authorization code will be provided back to the application. The authorization code can be used by the application to exchange for an access token. *Typically a client-id and client-secret is associated with every REST call to the door lock infrastructure system. This provides them the ability to manage the ISVs that has access to the backend. The client-id is typically provided in every call to the backend and the client-secret used as a signing mechanism to provide 'proof-of-possession of the client-secret.*
- ➢ The access token provided to the TPV application comes with an expiry and would have to be refreshed periodically via a call back to the authorization service.

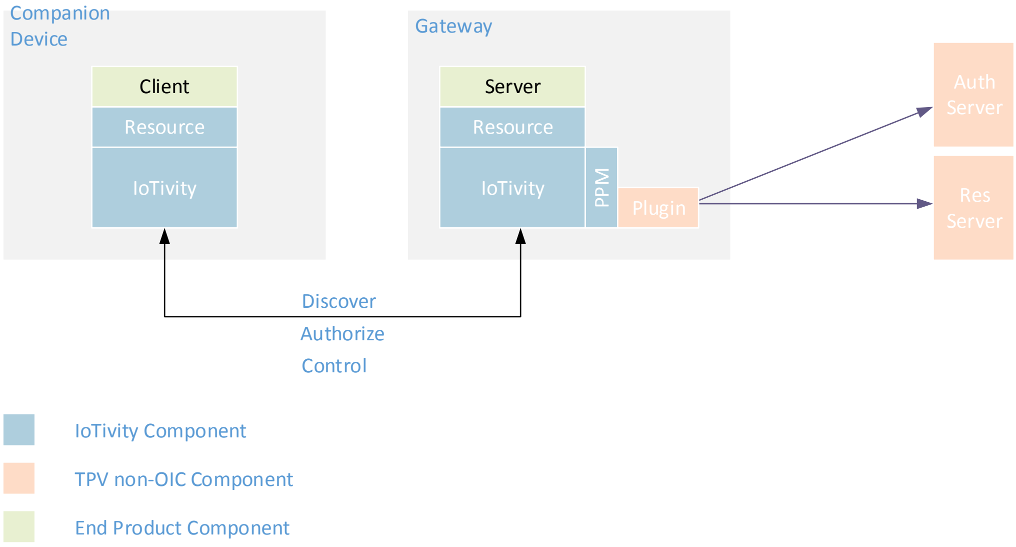
- Every command/control later performed to the resource server from the TPV application include the access token the resource server can use to ensure that the TPV is authorized to perform the requested operation.

For the non-OIC IoT service platform, a plugin is just another TPV (Third Party Application) application requesting access to the resources it provides. In order for the plug-in to be operational and expose its services through the OIC resource model it first has to be authorized. The next chapter will outline the flow and requirements on the PPM, plug-in and overall IoTivity stack to accomplish this.

PPM Interaction Model

The Protocol Plugin Manager serves primarily two roles in order to enable an end-2-end flow that allows an application to use the service/capability exposes by a plug-in.

(1) Load and register plug-ins service(s) to the IoTivity resource model.
(2) Provision and authorize the plug-ins.



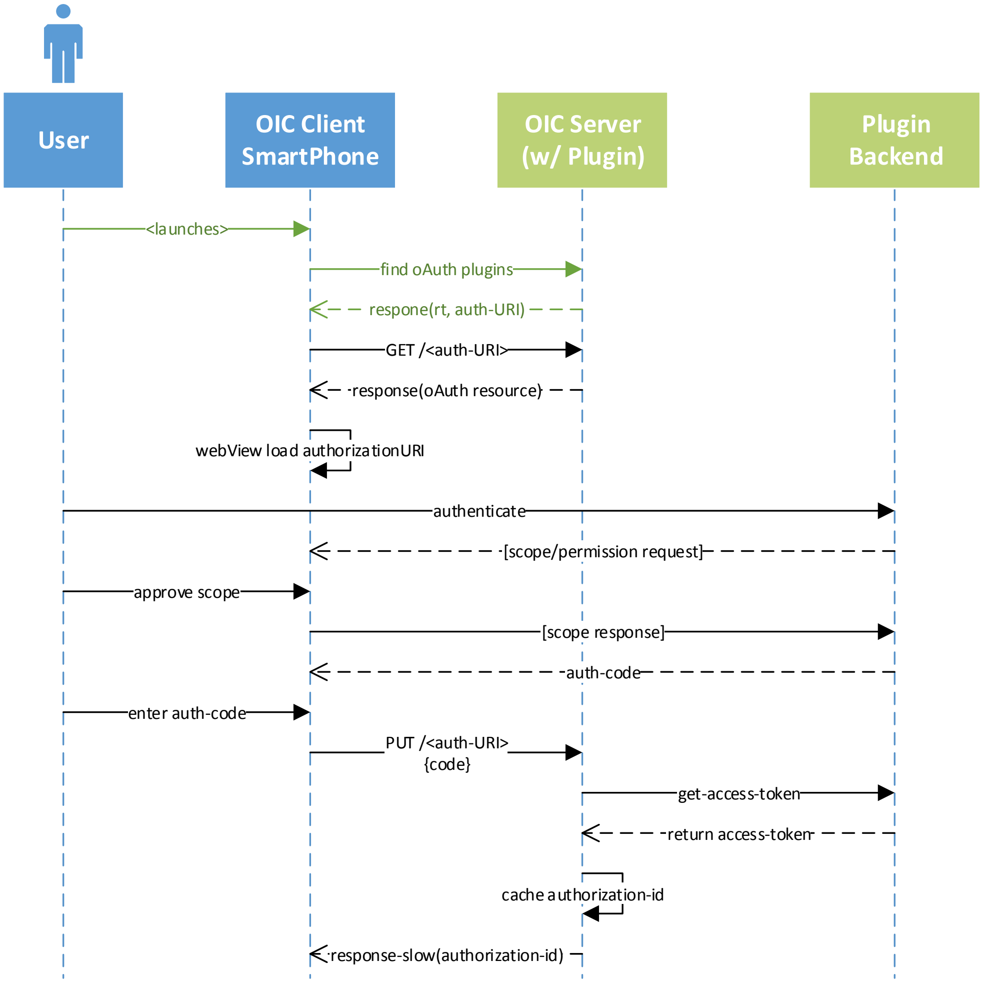
The Authorization and Resource server are components used by the plugin directly and are opaque to the rest of the IoTivity / OIC eco-system. The communication and interaction between the plug-in and its backend (whether locally or remotely) is achieved with protocols/standards outside the scope of OIC/IoTivity.

Authorization Flows

In order to support a wide variety of plugins from different manufactures it is necessary to have well defined flows to support the various authentication and authorization models defined and used by those eco-systems. This chapter outlines a few, commonly used mechanisms and the corresponding flows. The resource definition for the various authorization models is found in “PPM Resource Definition” chapter.

OAuth 2.0

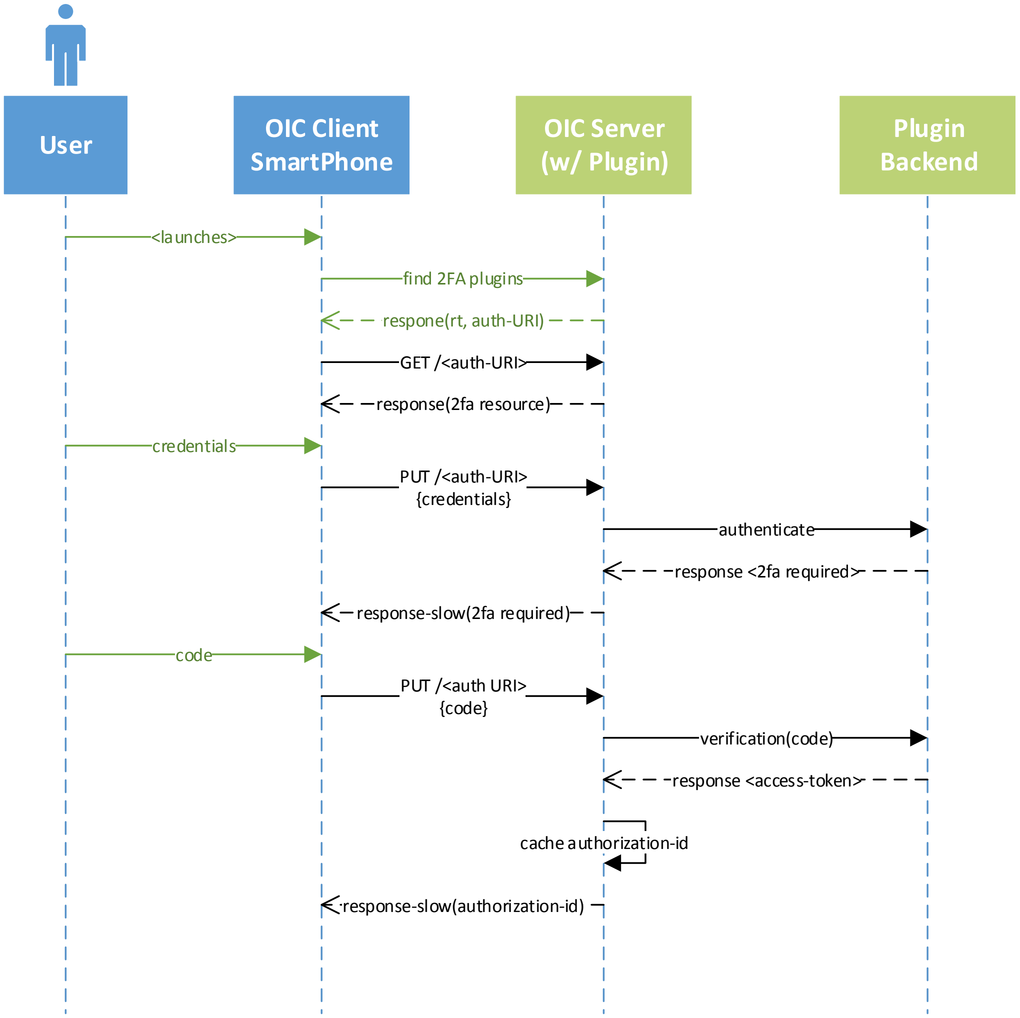
The diagram below outlines an OAuth flow.



- The smart device performs a discovery to find all of the plugins supporting OAuth.
- The smart device and the hosting server are presumed to have already been forming a trust relationship between them following the standardized security model and communication defined in the OIC specification.
- The smart device is provided with the URIs for the OAuth based plugin. The resource contains the URL to complete the authorization flow.
- The smart device loads the authorization URL in a webview and prompts the user to login and approve the requested permissions/scope requested by the plugin.
- The non OIC backend provides back an authorization code which can be used in exchange for an access token.
- The OIC client device uses the oAuth resource to write the authorization code to the plugin. The plugin internally uses the authorization code to get an access token. It caches the access token and generates an authorization-id which is returned back to the smart device.
- The smart device caches authorization ID which uniquely maps to this authorization flow. The authorization-id can later be used to ensure that the authorization is still valid as well as using it to de-authorize it.

2-Factor Authentication

The diagram below outlines a flow for a 2-factor authentication. Common 2-factor authentication mechanisms rely on a username/password tuple in conjunction with an additional factor of authentication which typically implies proof of possession of a device or an account. In the flow below, the form of the 2FA is generic and could be a phone, email address etc. Once the verification step has completed a code is provided to the plugin which allows for the authentication and authorization step to complete.

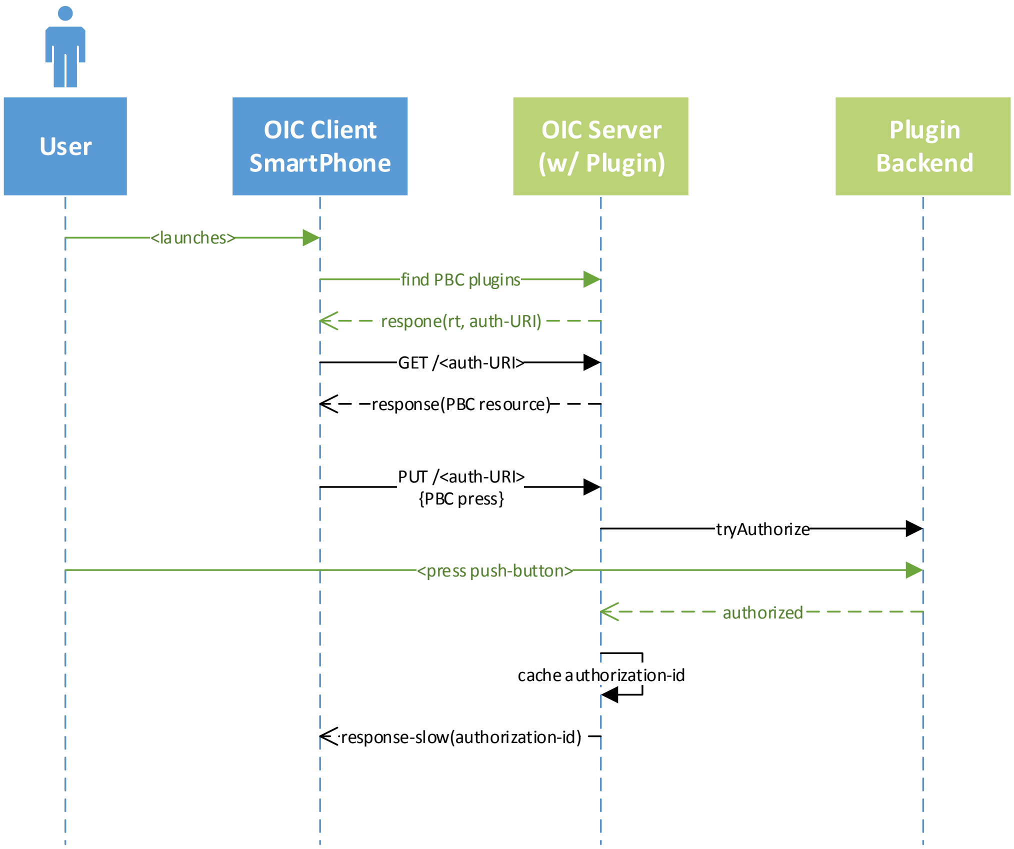


- The smart device performs a discovery to find all of the plugins supporting 2FA.
- The smart device and the hosting server are presumed to have already been forming a trust relationship between them following the standardized security model and communication defined in the OIC specification.
- The smart device is provided with the URIs for the 2FA based plugin. The resource contains the identifiers required to complete the authorization flow.
- The user is prompted for credentials, credentials are written to the 2FA resource on the plugin. The plugin attempts to establish a session with the credentials and is requested to validate a secondary device/account.

- The user receives and email or SMS containing the code that is required to establish the session and retrieve a verified access token that can be used for communication with the service.
- The code is written to the 2FA plugin, which uses it to retrieve a valid access token from the backend.
- It caches the access token and generates an authorization-id which is returned back to the smart device.
- The smart device caches authorization ID which uniquely maps to this authorization. The authorization-id can later be used to ensure that the authorization is still valid as well as using it to de-authorize it.

Push Button

The PBC (Push Button) model is one where the target device requires the user to open up an authorization Window for a brief period of time where pairing with a TPV client can be done before a timeout occurs. The authorization window is opened by the user physically presses some link/pairing button on the device and the plugin completing the authorization process before the timeout occurs. The timeout is vendor specific but might be in the range from 30sec – 2minutes.



- The smart device performs a discovery to find all of the plugins supporting PBC.
- The smart device and the hosting server are presumed to have already been forming a trust relationship between them following the standardized security model and communication defined in the OIC specification.
- The smart device receives the URI for the PBC resource and can initiate the PBC flow by a PUT operation on the resource. This will trigger the authorization flow to

commence from the plugin. The plugin might be used periodic polling, long-polling or other mechanism to achieve the PBC response before the timeout happens.

- The user physically presses the link/pair button on the target device, which informs the plugin that the authorization flow has completed.
- The OIC client is provided with the associated authorization-id for the pairing.

PPM Resource Definitions

This chapter defines the resource outlined and used in the examples above.

oAuth-Resource

oic.pp.oauth Resource Type Definition

Property title	Property name	Value type	Value rule	Unit	Access mode	Mandatory	Description
Authorization URL	aurl	string			R	Yes	Holds the OAuth authorization URL
Auth Code	acode	string			W	Yes	The authorization code.
Auth Code TTL	acode-ttl	Integer	> 0	Sec	W	No	The validity period of the auth-code
Manufacturer Name	mnmn	string			R	Yes	Friendly name of the manufacturer
Product Name	pn	string			R	Yes	Friendly name of the product.

2FA-Resource

oic.pp.2fa Resource Type Definition

Property title	Property name	Value type	Value rule	Unit	Access mode	Mandatory	Description
2FA Code	code	string			W	Yes	The 2FA code
Supported Identifiers	ids	uint16	Bitmask		R	Yes	Email: 0000000000000001b Phone: 0000000000000011b
Identifier	id	string			R	Yes	The identifier supplied. <email:><email>

							<phone:><phone> In e.164 format.
Manufacturer Name	mnmn	string			R	Yes	Friendly name of the manufacturer
Product Name	pn	string			R	Yes	Friendly name of the product.

PBC-Resource

oic.pp.pbc Resource Type Definition

Property title	Property name	Value type	Value rule	Unit	Access mode	Mandatory	Description
PBC action	pbc	bool	true		W	Yes	Initiate PBC challenge
Supported Identifiers	ids	uint16	Bitmask		R	Yes	Email: 0000000000000001b Phone: 0000000000000011b
Identifier	id	string			R	Yes	The identifier supplied. <email:><email> <phone:><phone> In e.164 format.
Manufacturer Name	mnmn	string			R	Yes	Friendly name of the manufacturer
Product Name	pn	string			R	Yes	Friendly name of the product.

Authorization-Context

oic.pp.auth_ctx Resource Type Definition

Property title	Property name	Value type	Value rule	Unit	Access mode	Mandatory	Description
Authorization ID	Aid	Char array	string		R	Yes	Holds the authorization identifier/context for a previous authorization