Specific Targeted Research Projects (STReP)

SOCIOTAL

Creating a socially aware citizen-centric Internet of Things

FP7 Contract Number: 609112

WP1 – Socially-aware citizen centric architecture and community APIs

Deliverable report

Contractual date of delivery: 31/08/2015

Actual submission date: 31/08/2015

Deliverable ID:	**D1.3.2**
Deliverable Title:	**Updated version of API Specification**
Responsible beneficiary:	CRS4
Contributing beneficiaries:	**CRS4, UC, CEA, DNET, UNIS, UMU**

Start Date of the Project: 1 September 2013 Duration: 36 Months

Revision: 1.0

Dissemination Level: Public

Document Information

Document ID: SOCIOTAL_D1.3.2_FINAL.docx
Version: Final 1.0
Version Date: 25/08/2015
Authors: Alberto Serra, Antonio Pintus, Davide Carboni (CRS4), Nenad Gligoric (DNET), Jorge Bernal Bernabé, Jose Luis Hernández (UMU), Ignacio Elicegui Maestro, Carmen López (UC), Colin O'Reilly, Michele Nati, Niklas Palaghias (UNIS), Etienne Gandrille, Levent Gurgen (CEA)
Security: Public

Approvals

	Name	Organization	Date	Visa
Project Management Team	Klaus Moessner	UNIS		
Internal reviewer	Ignacio Elicegui Maestro	UC		
Internal reviewer	Nenad Gligoric	DNET		

Document history

Revision	Date	Modification	Authors
Draft	18 Feb 15	First Draft, TOC and initial assignament	CRS4
Draft	23 Apr 15	General Contribution and comments. Added UC, CRS4, DNET, UNIS first draft APIs	CRS4
Draft	05 May 15	General Contribution	All
Draft	14 May 15	General Contribution	All
Draft	30 May 15	General Contribution	All
Draft	15 Jun 15	General Contribution	All
Draft	17 Jul 15	Sections completed	CRS4, CEA, UMU
Review Release	22 Jul 15	Version for internal reviewers	API
Final Release	19 Aug 15	Include all annexes and reviews.	CRS4, DNET, UMU, UC
Final Version	25 Aug 15	Final Version ready for submission	CRS4

Content

Table of Figures

Executive summary

Description of the deliverable content and purpose

The goal of this task (Task T1.3) is to specify a set of APIs (Application Programming Interfaces) that will expose a complete set of required functionalities of SocIoTal system to application developers.

This task collects information from the current status of the SocIoTal architecture (task T1.2) and from enablers and components coming from work packages WP2/3/4. At this stage, this deliverable reports an updated version of the first API description provided in Deliverable D1.3.1[1], taking into account the inputs from other workpackages.

Advances in API specification produced a full technical documentation of them, complete in all details. Documentation has been used during API evaluation sessions, where developers used it to test and evaluate the APIs following a defined workflow, providing a valuable feedback for each API group, as reported in a dedicated section of this document.

The deliverable is structured as follows:

Section 1 – reports a general overview of the SocIoTal API; in particular, it introduces a semi-formal notation to describe them in a uniform way, concerning: naming conventions, parameters and data types, returned values, entities representations, scope, description and access, and API provider as well.

Sections 2 to 4 – provides the API specification for all the envisioned SocIoTal macro-modules. Specifications comply with the notation introduced in Section 1. Macro modules are sets of APIs grouped by reflecting the particular functional area they belong.

Section 5 – describes the overall API Evaluation Workflow defined and adopted. The workflow is used during API evaluation sessions with developers. It also includes the definition of an evaluation card, filled by sessions' supervisors and designed to collect developers' feedback. Finally, the section provides the results of the first evaluation sessions and a summary.

[1] Sociotal D1.3.1 - First version of API Specification

Contribution of the Deliverable to the overall project objectives

This deliverable contributes to specify and document advances in the SocIoTal open API definition, which is one of the overall project objectives and provides a detailed documentation of them.
The main SocIoTal objective is to unleash the full potential of IoT, going beyond the enterprise centric systems and moving towards a citizen inclusive IoT in which IoT devices and contributed information flows provided by people are encouraged. To lower the barrier of IoT bottom-up development and adoption, some key aspects must be tackled: security, control, transparency and simplicity. These aspects are addressed in different blocks of the SocIoTal architecture and are also integrated and made available by means of tools.

An important aspect of any architecture is specification of the software interfaces between its components. WP1 specifically addresses the specification of the interface offered to the users, i.e. those developing solutions based on the SocIoTal architecture. API specification allows to define and expose all the SocIoTal architecture functionalities enabling the development of IoT applications on top of them.

The architecture and the APIs are used to guide the research and development work in WP2, WP3, WP4 and WP5.
This deliverable addresses the objective O1.4: To specify required application programming interfaces (APIs) related to task T1.3: Open APIs for service development and IoT device integration (M7- M36), that will enable development of services on top of the SocIoTal architecture, having in mind that the targeted users are public at large.

Section 1. Overview of SocIoTal API

Programming interfaces are a key element in a project with many components that aims at being open to third party applications developers. Thus, the function of programming interfaces is twofold; on one hand it allows to better define the interdependencies among different modules and work packages inside the SocIoTal project itself; on the other it allows to document with a concise set of entry points, the ways an external application could make use of SocIoTal functionalities.
The SocIoTal API are structured in three main macromodules grouped by their functionalities in the global architecture. These macro modules APIs are:

- **SocIoTal Context Manager API:** to provide access to context information and manage context entities (like devices).
- **Security & Privacy API:** to control the access to the system. It is a combination of different authorization, authentication and privacy control technologies and tools.
- **SocIoTal User Environment API:** to expose the User Environment Tools functionalities in the form of web APIs.

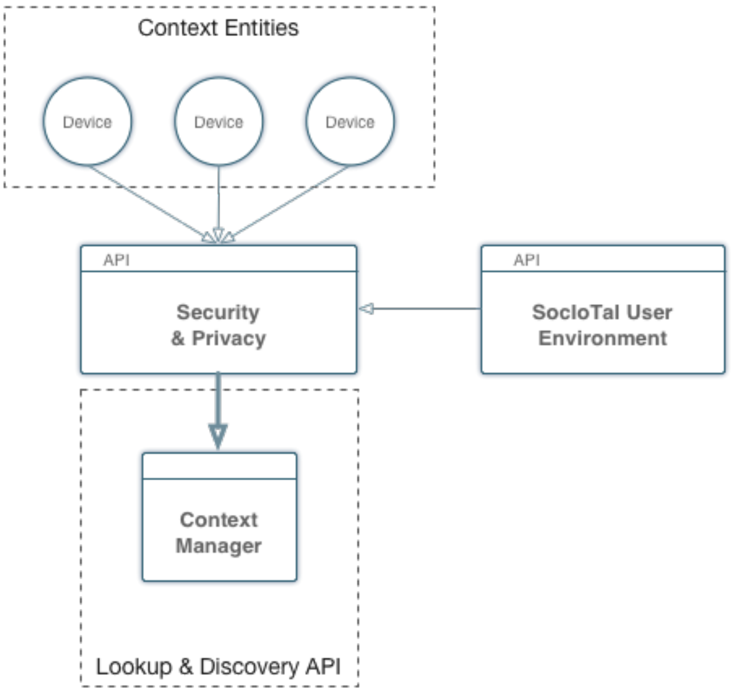

Figure 1. Overview of the API groups and their correlation

1.1. Introduction

The methodology used to retrieve the API is mainly based on a strict communication and alignment with the activities of software architecture design. In the following sections, more details about the format, scope and access are described.

1.1. Note on format

Unless specified differently, the APIs follow a pragmatic REST style using the HTTP protocol. Description of calls is given as follows.
For **POST** requests:

Name	**ThisIsAFunction**	
Description	Description here…	
SocIoTal Server	<uri:port>	
Method	**POST**	
	<uri>	
Headers		
	Content-type	application/json
	Accept	application/json
Request Payload		
<JSON>		
Response Payload		
<JSON>		
Status Codes:		
<All the HTTP status codes returned by API>		
Error Messages		
<All the messages returned by API in case of error>		

For **GET** requests:

Name	**ThisIsAFunction**	
Description	Description here…	
SocIoTal Server	<uri:port>	
Method	**GET**	
	<uri>	
Headers		
	Content-type	application/json
	Accept	application/json
Response Payload		
<JSON>		
Status Codes:		
<All the HTTP status codes returned by API>		
Error Messages		
<All the messages returned by API in case of error>		

If not specified, type of arguments and return values are generic object references with unspecified attributes. When attributes are known, the object is represented in the form {attr1, attr2, ...}. If arguments or return values are list or arrays they are represented as [item,item,...].

Examples:

SocIoTal Server	<sociotal_server>:<port>	
Method	**POST**	
	/SocIoTal_Context_UC_REST/NGSI9_API/registerContext	
Headers		
	Content-type	application/json
	Accept	application/json

These rows refer to the URI location of the API, the HTTP method (GET, POST, DELETE, PUT) used and the HTTP headers. If the method is a POST request a Request Payload should be provided. For example:

Request Payload:

```
{
   "contextRegistrations": [{
      "entities": [{
         "id": "SocIoTal:SAN:WeatherStation:Dev_001",
         "type": "urn:x-org:sociotal:resource:device",
         "isPattern": "false"
      }],
      "attributes": [{
         "name": "AmbientTemperature",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/AmbientTemperature"
      },
      {
         "name": "HumidityValue",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/HumidityValue"
      },
      {
         "name": "Location",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/Location"
      }],
      "providingApplication":
"http://an.example.server:3500/SocIoTal_Context_UC_REST/NGSI10_API"
   }],
   "duration": "P1M"
}
```

In case of error, a descriptive body is returned and the right HTTP status is used to classify the error.

Example of RESTful body error response
Status: **403**
Body: `{"error": "Your request is wrong, please check parameters"}`

For local API (non REST API, like library linking) will be used the common exception rising (which depends on the actual programming language) for synchronous calls, and callback parameters for asynchronous calls.

Section 2. Lookup & Discovery: the SocIoTal Context Manager

2.1. Introdution to the SocIoTal Context Manager

SocIoTal Context Manager will provide a standard REST API OMA compliant, implementing both NGSI9 and NGSI10 to gain access to context information and manage context entities.

The SocIoTal core platform implements and provides a centralized Context Manager, as described in D3.2.1[1].
Figure 1 presents the Context Manager architecture, including all the sub-components built-in.

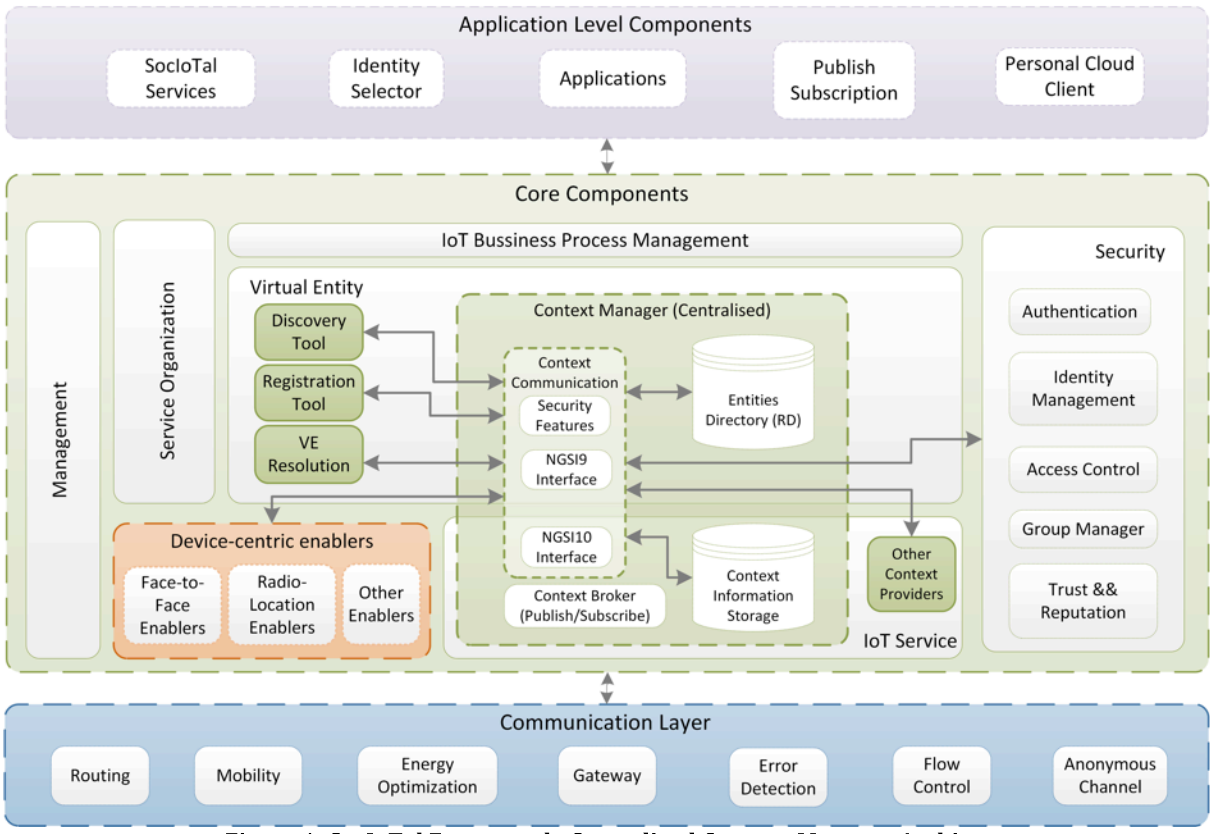

Figure 1. SocIoTal Framework. Centralized Context Manager Architecture

Context Entities are entities that are described by **Context Information.** Figure below gives some examples on entities that can be used as Context Entities. Context Entities are described by the **Context Information Model.**

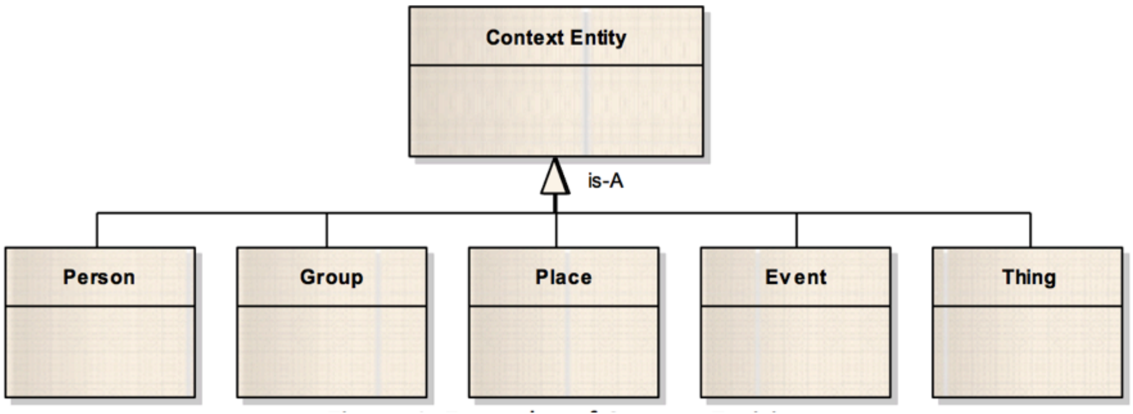

Figure 2. Examples of context entities

The Context Information Model details how Context Information is structured and associated to Context Entities in order to describe their situation. In this model, Context Information is organized as Context Elements, which contain set of Context Attributes and associated metadata (figure below).

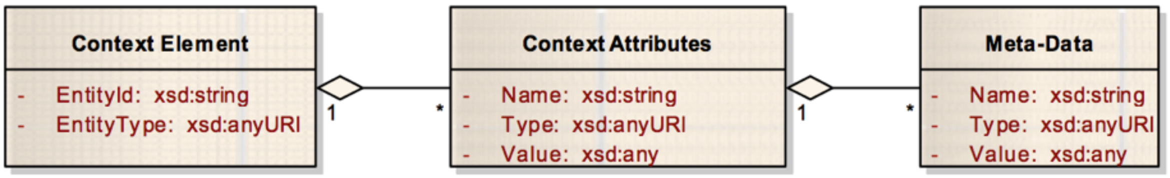

Figure 3. Context Information Model

An entity is a virtualization of "things" in IoT and could be a person, animal, car, a physical location or any other object in the physical world. The ***"entity"*** is the main focus of interactions by user and software agents in the IoT world and are the main ***"Context Producers"***. In addition to the profile properties such as name and identifier, the virtual entity model defined in IoT-A followed by SocIoTal D1.2.1[2], includes properties to describe location of an entity, and description elements for features of interest for an entity (features that can be observed by a sensing mechanism or can be changed/controlled by an actuation process).
In a very simplified way, following the context model introduced above, the model that define an entity is divided into two main blocks:

[2] Sociotal D1.2.1 – First Version of Sociotal Architecture

- *Entity Definition*, that encloses the information needed to identify the entity and the type it belongs
- *Set of attributes*, including all the different set of data the entity can somehow offer. These attributes also conforms the context of the entity. The attributes are composed by its name, type, metadata (definition of what the attribute includes and how it can be read) and the values themselves.

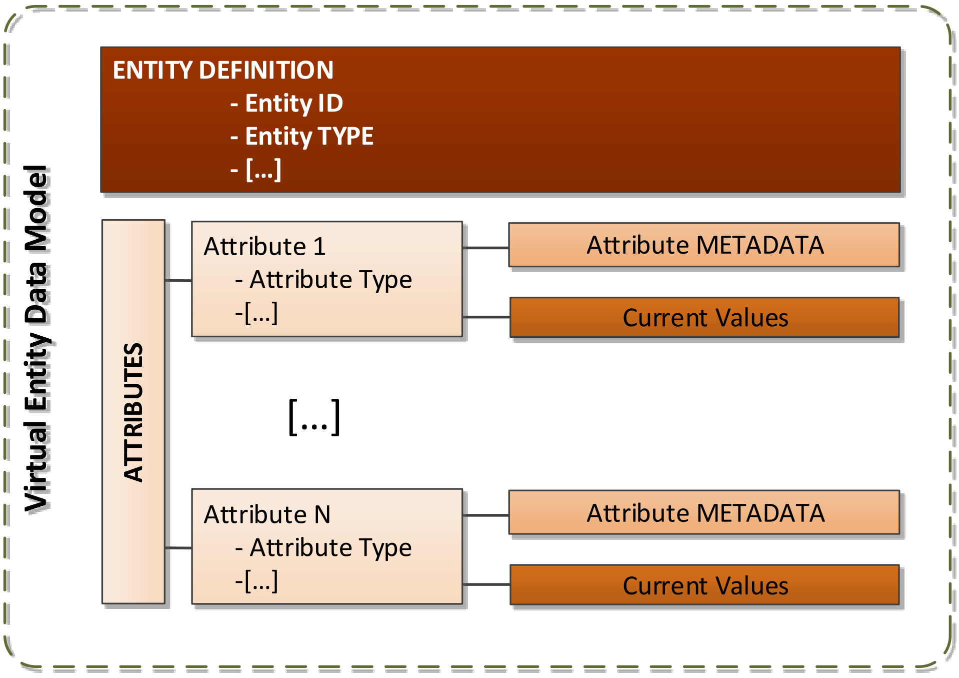

Figure 4. Virtual Entity Data Model

2.2. SocIoTal Context Manager DataModel mapping

Based on the OMA Context Model and the Virtual Entities datamodel presented in SocIoTal, we've created a first datamodel to register entities (as context producers) within the SocIoTal Context Manager as described in D3.2.1.
It is divided into two main blocks

- The Entity Definition, including the:
  - **Entity Type:** defines the type of the entity (Device, Resource, Context Data, Person, Sensor …)

- **Entity ID:** creates a unique ID for the created entity. Here, a URI, URN or URL could be usel

- *Set of attributes*, with the different context values and its structure that the entity offers. Every attribute will have a "name", a "type" and a "value", used to identify it and conform the context. Also, metadata with further information about the reported context attribute (such accuracy, unit of measurement, time when it was collected …) can be associated.

JSON DataModel (for entities)

JSON structure for Santander Weather Forecast Use Case Weather Station entity definition [OMA Compliant]

```
{
   "contextElements": [{
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "false",
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001", //real value
SocIoTal:SAN:WeatherStation:Dev_001
      "attributes": [{
         "name": "AmbientTemperature",
         "value": "25.44",
         "type": "http://sensorml.com/ont/swe/property/AmbientTemperature",
         "metadatas": [{
            "name": "DateTimeStamp",
            "value": "20141030T113343Z",
            "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
         },
         {
            "name": "Unit",
            "value": "celsius",
            "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
         },
         {
            "name": "accuracy",
            "value": "0,5",
            "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
         },
                                                   {
            "name": "DataDescription",
            "value": "float",
            "type": "http://sensorml.com/ont/swe/property/DataDescription"
         }]
      },
      {
         "name": "HumidityValue",
         "value": "59",
         "type": "http://sensorml.com/ont/swe/property/HumidityValue",
         "metadatas": [{
            "name": "DateTimeStamp",
            "value": "20141030T113343Z",
            "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
         },
         {
            "name": "Unit",
            "value": "percentage",
            "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
```

```
        },
        {
          "name": "accuracy",
          "value": "1",
          "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
        },
                                             {
          "name": "DataDescription",
          "value": "integer",
          "type": "http://sensorml.com/ont/swe/property/DataDescription"
        }]
      },
      {
        "name": "Location",
        "value": "43.472057, -3.800156",
        "type": "http://sensorml.com/ont/swe/property/Location",
        "metadatas": [{
          "name": "WorldGeographicReferenceSystem",
          "value": "WSG84",
          "type":
"http://sensorml.com/ont/swe/property/WorldGeographicReferenceSystem"
        }]
      }],
   }]
}
```

2.3. Capability Token

Every request sent to SocIoTal Context Manager must contain a **Capability-token** header (as part of all the HTTP request headers). This header will include, in JSON format, the needed data (credentials) to check the identity and policies that allows access to the requested resource. If the requestor identity and/or credentials do not match with the specified resource, the SocIoTal Context Manager will return an "Unauthorised" message (Error 401).
An Example of Capability Token is the following:

```
{
  "id": "7g3vfT_q9vTL2aQ4",
  "ii": 1415174237,
  "is": "issuer@um.es",
  "su": "zNwS5FetB4rwzSKsWwSBAxm5wDa=JgLjHU8zSnmeSFQgSG9HhdsJrE8=",
  "de": "coap://sensortemp.floor1.computersciencefaculty.um.es",
  "si": "SbUudG4zuXswFBxDeHB87N6t9hR=PBQqCN3gpu7nSkuPzDk7kaR3dq1=",
  "ar": [
      {
          "ac": "GET",
          "re": "temperature",
          "f": 1,
          "co": [
              {
                  "t": 5,
                  "v": 25,
            "u": "Cel",
          },
          {
            "t": 6,
            "v": 20,
            "u": "Cel",
          }
          ]
      }
  ],
  "nb": 1415174237,
  "na": 1415175381
}
```

The Capability Token JSON fields and the overall access control flows, including how to obtain a Capability Token, are described in detail in D2.2 [2].

2.4. NGSI9 API

Oriented to Manage Context Entities: define, register, modify, discover and subscribe context entities and their attributes. It can be seen as a ***Context Entities Directory***, (or a *RESOURCE DIRECTORY*) providing info about the entities registered and their set of attributes but not attribute values.
Version 2 of the SocIoTal CM's NGSI9 API is represented by the URI:

SocIoTal CM V2	**/SocIoTal_CM_REST_V2/NGSI9_API/{method}**

Name	**registerContext**	
Description	Register NEW Context Entities or UPDATE existing ones, including their attribute names and availability. *THIS should be the first operation to be performed when creating a new Context Entity.*	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI9_API/registerContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token – see example at 3.1.1]
Payload:		

```
{
   "contextRegistrations": [{
      "entities": [{
         "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
         "type": "urn:x-org:sociotal:resource:device",
```

Version Date: 25 Aug 2015
Security: Confidential

```
         "isPattern": "false"
      }],
      "attributes": [{
         "name": "AmbientTemperature",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/AmbientTemperature"
      },
      {
         "name": "HumidityValue",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/HumidityValue"
      },
      {
         "name": "Location",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/Location"
      }],
      "providingApplication":
"http://capella.tlmat.unican.es:3500/SocIoTal_Context_UC_REST/NGSI10_API"
   }],
   "duration": "P1M"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 "OK"	The request has been properly executed and the result has been included in the response payload. (Initial versions won't show error code in this case)
400 "BAD REQUEST"	Any of the fields have been not properly performed. The response payload will include further info if available.
401 "UNAUTHORIZED"	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
500 "INTERNAL SERVER ERROR"	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 "METHOD NOT ALLOWED"	Check you're using POST
415 "UNSUPPORTED MEDIA TYPE"	Check "Content-type application/json" header and the payload
500 "INTERNAL SERVER ERROR"	Error accessing Context Manager server/service

The payload includes a list of ***contextRegistration*** elements, each one with the following information:

- **entities**: entities to be registered; the example only shows one entity (*SocIoTal:IoTWeek:WeatherStation:Dev_001*) but one payload can be used to register several entities (if they share the same set of attributes) by just adding them as a list of entities. For each entity we specify a ***type*** and an ***ID***. The isPattern field is mandatory, it will be set as "true" when performing complex searches or subscriptions using regular expressions in the “id” field. However, within the rest of requests, as for example this contextRegistration, this pattern will be set as “false” because it is not actually used.

- **attributes**: a list of attributes to register for the entities. In our case, they are the “*AmbientTemperature*”, “*HumidityValue*” and “*Location*” attributes, according to what have been discussed within SocIoTal related to Context Model. For each one, we define a name, a type and whether it is a domain attribute or not. Current version of SocIoTal Context Manager does not support domain attributes, so isDomain must always be set to "false". IMPORTANT, this set of attributes will be defined for all entities within the list specified above.
- **providing application** (or Context Provider): the URL that represents the actual context information for the entities and attributes being registered. In the example we are assuming that all the sensors info (attribute values) are provided by [*http://a.sociotal.server:3500/SocIoTal_Context_UC_REST/NGSI10_API*] (currently, just an example). IMPORTANT: if we’re defining a list of entities within the same registration request, all will share the same “providing application” value.
- **duration**: sets the duration of the registration so after that time has passed it can be considered as expired (however, duration can be extended). ISO 8601 standard is used for duration format, for example "P1M" means "one month".

When performing the above example, the response payload is

Response JSON:

```
{
   "duration": "P1M",
   "registrationId": "549304a81860a3e2039f5ae4"
}
```

The **registrationId** provides a unique reference to the registration. It is used for updating (when needed) the registration of the entity/ies, by adding it to a new “registerContext” request (within the payload):

Payload JSON:

```
{
   "contextRegistrations": [{
      "entities": [{
         "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
         "type": "urn:x-org:sociotal:resource:device",
         "isPattern": "false"
      }],
      "attributes": [{
         "name": "AmbientTemperature",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/AmbientTemperature"
      },
      {
         "name": "HumidityValue",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/HumidityValue"
      },
      {
         "name": "Location",
         "isDomain": "false",
         "type": "http://sensorml.com/ont/swe/property/Location"
      }],
      "providingApplication":
"http://a.sociotal.server:3500/SocIoTal_Context_UC_REST/NGSI10_API"
   }],
   "duration": "PT1H",
   "registrationId": "549304a81860a3e2039f5ae4"
}
```

This is the way to **MODIFY** existing context registrations. The example above changes the duration of the registered context to 1 hour, but other attributes, types or names could also be modified by including them in the "registerContext" payload.
OMA NGSI9 does not (currently) provide methods to **DELETE** context registrations.
PROVIDING APPLICATION element: as mentioned above, this element will provide (when discovering request is performed) the URL of the service providing the related Context Information →*The Context Provider*.
Using FIWARE ORION Context Broker [3] when implementing SocIoTal Context Manager (centralized), when a query or update operation (NGSI10) is requested and the targeted context element is not stored locally (i.e. in Orion's internal database) but a Context Provider is registered ("providingApplication" element is defined) for that context element, then this request (NGSI10 context query/update) will be forwarded to the Context Provider URL specified. In this case, Orion acts as a pure "NGSI proxy" (i.e. doesn't cache the result of the query internally) and, from the point of view of the client launching the original request, the process is mostly transparent. In this case, *the Context Provider specified is supposed to implement the NGSI10 API (at least partially) to support the query/update operation*.

Name	**discoverContextAvailability**	
Description	Allows the synchronous discovery of Context Entities previously registered, providing access to their context structure (entities' ID, set of attributes and providing applications).	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI9_API/discoverContextAvailability	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token – see example at 3.1.1]
Payload:		

```
{
   "entities": [{
      "type": "",
      "isPattern": "true",
      "id": "SocIoTal.*"
   }]
}
```

Error Messages (Status codes):	
errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the discovering parameters/set of attributes)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Here you can use the "isPattern" field in "true" and perform slightly more complex searches using regular expressions within the "id". The payload example below searches for all context provided by SocIoTal entities. The response will include all registered entities (using NGSI9) which id starts with “SocIoTal”:

Response JSON:

```
{
   "contextRegistrationResponses": [{
      "contextRegistration": {
         "providingApplication":
"http://capella.tlmat.unican.es:3500/SocIoTal_Context_UC_REST/NGSI10_API"
         "attributes": [{
            "isDomain": "false",
            "name": "AmbientTemperature",
            "type": "http://sensorml.com/ont/swe/property/AmbientTemperature"
         },
         {
            "isDomain": "false",
            "name": "HumidityValue",
            "type": "http://sensorml.com/ont/swe/property/HumidityValue"
         },
         {
            "isDomain": "false",
            "name": "Location",
            "type": "http://sensorml.com/ont/swe/property/Location"
         }],
         "entities": [{
            "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
            "type": "urn:x-org:sociotal:resource:device",
            "isPattern": "false"
         }]
      }
   }]
}
```

Using an empty set of attributes (by not providing an “attributes” element, as in the example) in the request, the discovering process searches for any registered entity matching the “entities” element, no matter which attributes have been registered. If a discovering process based on an attribute or a set of attributes is required, the name (or names) of the attribute(s) to search for should be included in the request, within the “attributes” element:

Payload JSON:

```
{
   "entities": [{
      "type": "",
      "isPattern": "true",
      "id": "SocIoTal.*"
   }],
   "attributes": ["AmbientTemperature"]
}
```

Name	**subscribeContextAvailability**	
Description	Allows the asynchronous discovery of the potential set of context entities belonging to an identified type and/or providing desired attributes. Using this subscription call it is possible to be notified (by POST method) with an asynchronous message (to the “reference” registered callback function) when a new entity is added to SocIoTal group (or an existing one changes); e.g., when a new device appears or when a new “AmbientTemperature” attribute is added (by a new entity or an existing one). Also, different options can be combined, in order to subscribe e.g. to devices registered within SocIoTal and providing “AmbientTemperature” information.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI9_API/subscribeContextAvailability	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]
Payload:		

```
{
   "entities": [{
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "true",
      "id": "SocIoTal.*"
   }],
   "attributes": ["AmbientTemperature", "HumidityValue"],
   "reference": "http://a.sociotal.ref:1650/Callback_Apps/Listeners/LogWriterFile",
   "duration": "P1M"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

The payload is conformed by the following elements:

- **entityIdList and attributeList**: here it is defined the context availability information we are interested in, selecting the context registrations to include in the notifications. In the payload example, as we use regular expressions to define the “id”, the “isPattern” field is set to “true”.
- **reference**: is the callback URL to send notifications. Only one reference can be included per subscribeContextAvailability request. However, you can have several subscriptions on the same context availability elements.
- **duration**: specifies the duration of the subscription. Once the duration expires, the subscription is ignored. The format to specify the duration follows the ISO 8601 standard.

This request will return a subscriptionId which will identify the context subscription.

Response JSON:

```
{
    "duration": "P1M",
    "subscriptionId": "55c1bac3d9c4edb2597aee69"
}
```

Name	**unsubscribeContextAvailability**	
Description	Just deletes an existing subscription to discover Context Information. The payload will only contain the “subscriptionId” returned by the “subscribeContextAvailability” method.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI9_API/unsubscribeContextAvailability	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]

Payload:

```
{
    "subscriptionId": "55a51fbad9e4be3111e2a2d1"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the subscriptionId element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Name	**updateContextAvailabilitySubscription**	
Description	This method updates an existing context availability subscription (performed using NGSI9 “subscribeContextAvailability). The request includes, in its payload, the “subscriptionId” element that identifies the subscription to modify and the set of elements to be updated:	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI9_API/updateContextAvailabilitySubscription	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token – - see example at 3.1.1]

Payload:

```
{
   "entities": [{
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "true",
      "id": "SocIoTal.*"
   }],
   "attributes": ["HumidityValue"],
   "duration": "P1D",
   "subscriptionId": "54b406a01860a3cc3260a7e0"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the subscriptionId element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload

500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

This example will keep the devices, but changes the attributes (this time only devices providing Humidity values will be targeted) and the duration, set to 1 day.

- ***What can be updated through a Context Availability Subscription?***

Although the payload used by an updateContextAvailabilitySubscription is pretty similar to the one in the subscribeContextAvailability request, not all fields can be updated. In particular, the following fields cannot be updated:

- subscriptionId (you must include it in the request but you can’t change it)
- reference

The following fields can be updated:

- entityIdList
- attributeList
- duration

2.5. NGSI10 API

Oriented to Manage (get, update, discover and subscribe) Context Information. Version 2 of the SocIoTal CM’s NGSI10 API is represented by the URI:

SocIoTal CM V2	**/SocIoTal_CM_REST_V2/NGSI10_API/{method}**

Name	**updateContext**
Description	Update Context NGSI10 method provides 3 main functionalities (specified through its JSON payload element UPDATE ACTION values): • **APPEND→** to create a new context entity and/or add new attributes to the context (NGSI10 level). • **UPDATE→** to replace existing attributes values (and metadata) of a given entity (or context element) with the current values • **DELETE→** removes an existing context entity (only at NGSI10 level. It won’t remove the corresponding –if exists- NGSI9 instance)

Name	updateContext – { "updateAction": "APPEND" }	
Description	If the entity described through "type" and "id" elements does not exist, the given entity (a NEW contextElement) with the set of attributes (context) specified in the JSON payload will be created, included the values also given in the request. If the entity pointed already exists, it will add all the new attributes specified in the JSON payload to the entity context. No checking of any value included in the attribute elements is initially performed, so temperatures like "-10000 ºC" or "hot" could be added as context values.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/updateContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]

Payload:

```
{
   "contextElements": [{
      "type": " urn:x-org:sociotal:resource:device ",
      "isPattern": "false",
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
      "attributes": [{
         "name": "AmbientTemperature",
            "value": "25.44",
            "type": "http://sensorml.com/ont/swe/property/AmbientTemperature",
            "metadatas": [{
               "name": "DateTimeStamp",
               "value": "20141030T113343Z",
               "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
            },
            {
               "name": "Unit",
               "value": "celsius",
               "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
            },
            {
               "name": "accuracy",
               "value": "0,5",
               "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
            },
                                                                             {
               "name": "DataDescription",
               "value": "float",
               "type": "http://sensorml.com/ont/swe/property/DataDescription"
            }]
         },
         {
            "name": "HumidityValue",
            "value": "59",
            "type": "http://sensorml.com/ont/swe/property/HumidityValue",
            "metadatas": [{
               "name": "DateTimeStamp",
               "value": "20141030T113343Z",
               "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
            },
            {
               "name": "Unit",
               "value": "percentage",
               "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
            },
            {
               "name": "accuracy",
               "value": "1",
               "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
            },
                                                                             {
               "name": "DataDescription",
               "value": "integer",
               "type": "http://sensorml.com/ont/swe/property/DataDescription"
            }]
         },
         {
            "name": "Location",
            "value": "43.472057, -3.800156",
            "type": "http://sensorml.com/ont/swe/property/Location",
            "metadatas": [{
               "name": "WorldGeographicReferenceSystem",
               "value": "WSG84",
               "type":
"http://sensorml.com/ont/swe/property/WorldGeographicReferenceSystem"
            }]
         }],

   }],
   "updateAction": "APPEND"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	**Message/Additional info**
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED	Check “Content-type application/json” header and the payload

MEDIA TYPE”	
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

- ***How to manage attributes with the same name***

It could happen that you may need to add attributes with the same name and type to an entity. In order to differentiate them we will use a metadata named by ID. For example:

Payload:

```
{
    "contextElements": [
        {
            "type": "urn:x-org:sociotal:resource:device",
            "isPattern": "false",
            "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
            "attributes": [
                {
                    "name": "AmbientTemperature",
                    "value": "25.4",
                    "type": 
"http://sensorml.com/ont/swe/property/AmbientTemperature",
                    "metadatas": [
                        {
                            "name": "ID",
                            "value": "ground",
                            "type": "TBD"
                        }
                    ]
                },
                {
                    "name": "AmbientTemperature",
                    "value": "28.2",
                    "type": 
"http://sensorml.com/ont/swe/property/AmbientTemperature",
                    "metadatas": [
                        {
                            "name": "ID",
                            "value": "wall",
                            "type": "TBD"
                        }
                    ]
                }
            ]
        }
    ],
    "updateAction": "APPEND"
}
```

If we add some equal attributes and we do not use the differentiation by ID name we will receive the error presented below:

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info

472	"request parameter is invalid/not allowed" - Attributes with same name with ID and not ID at the same time in the same entity are forbidden

- ***Structures for attribute values***

Apart from simple strings such as "22.5" or "ground", you can use complex structures as **attribute** values (this is not valid for values within metadata). In particular, an attribute can be set to a vector or to a key-value map (usually referred to as an "object")

Payload:

```
{
    "contextElements": [
        {
            "type": "urn:x-org:sociotal:resource:device",
            "isPattern": "false",
            "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
            "attributes": [
                {
                    "name": "AmbientTemperature",
                    "value": [
                        "22",
                        {"x": ["x1","x2"],
                             "y": "3"
                           },
                           ["z1",
                            "z2"
                           ]
                       ],
                       "type":
    "http://sensorml.com/ont/swe/property/AmbientTemperature"
                },
                {
                    "name": "Humidity",
                    "value": {
                        "x": {
                            "x1": "a",
                            "x2": "b"
                        },
                        "y": ["y1","y2"]
                    },
                    "type":"http: //sensorml.com/ont/swe/property/HumidityValue"
                }
            ]
        }
    ],
    "updateAction": "APPEND"
}
```

Name	updateContext – { “updateAction”: “UPDATE” }	
Description	The Update Action “UPDATE” will put in the existing context of the pointed entity the last value (removing the previous one) measured for the attribute (or list of attributes) specified in the payload, as well as updates the associated metadata (if proceeds).	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/updateContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]

Payload:

```
{
   "contextElements": [{
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "false",
      "attributes": [{
         "name": "AmbientTemperature",
            "value": "25.44",
            "type": "http://sensorml.com/ont/swe/property/AmbientTemperature",
            "metadatas": [{
               "name": "DateTimeStamp",
               "value": "20141030T113343Z",
               "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
            },
            {
               "name": "Unit",
               "value": "celsius",
               "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
            },
            {
               "name": "accuracy",
               "value": "0,5",
               "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
            },
                                                                              {
               "name": "DataDescription",
               "value": "float",
               "type": "http://sensorml.com/ont/swe/property/DataDescription"
            }]
         }]
   }],
   "updateAction": "UPDATE"
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t

	show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the id element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Name	**updateContext – { “updateAction”: “DELETE” }**	
Description	**DELETE** value is used to: - Delete an attribute (and its associated metadata) of a given entity, pointing the selected attribute (or list of attributes) in the payload (but not including the attribute value nor the metadata specified). - Delete a whole context entity, by passing no attributes in the payload, only the type, isPattern and id elemens.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/updateContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]
Payload:		

```
{
   "contextElements": [{
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "false",
      "attributes": [{
         "name": "AmbientTemperature",
         "value": "",
         "type": "http://sensorml.com/ont/swe/property/AmbientTemperature"
      }]
   }],
   "updateAction": "DELETE"
}
```

Error Messages (Status codes):	
errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the id element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Name	queryContext	
Description	Retrieval of Context Information. The requestor can specify restrictions on returned Context Information and perform context searching. The standard response includes all the attributes belonging to the given entities with their current values. An empty attributes element in the request will return all the attributes of the entity. If you include an actual list of attributes (e.g. “AmbientTemperature”) only that are retrieved. A powerful feature (based on functionalities provided by Orion Context Broker) is that a regular expression for the entity ID can be used. For example, all SocIoTal Weather Stations context information could be retrieved by using “SocIoTal:IoTWeek:WeatherStation:.*” as “id” and “true” as “isPattern” elements. In the case that a list of context entities are required, a “limit” param can be included, so the response will be limited to the first “n” hits.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/queryContext?limit=1	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability

		token - see example at 3.1.1]

Payload:

```
{"entities":[
    {"type":"",
     "isPattern":"false",
     "id":"SocIoTal:IoTWeek:WeatherStation:Dev_001"
    }
  ],
"attributes":["AmbientTemperature"]
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the id element and attribute parameters)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Response JSON:

```
{
   "errorCode": {
      "details": "Count: 1",
      "reasonPhrase": "OK",
      "code": "200"
   },
   "contextResponses": [{
      "statusCode": {
         "reasonPhrase": "OK",
         "code": "200"
      },
      "contextElement": {
         "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
         "attributes": [{
            "name": "AmbientTemperature",
            "value": "25.44",
            "type": "http://sensorml.com/ont/swe/property/AmbientTemperature",
            "metadatas": [{
               "name": "DateTimeStamp",
               "value": "20141030T113343Z",
               "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
            },
            {
               "name": "Unit",
               "value": "celsius",
               "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
            },
            {
               "name": "accuracy",
               "value": "0,5",
               "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
            },
                                                                              {
               "name": "DataDescription",
               "value": "float",
               "type": "http://sensorml.com/ont/swe/property/DataDescription"
            }]
         }],
         "type": "urn:x-org:sociotal:resource:device",
         "isPattern": "false"
      }
   }]
}
```

Name	**subscribeContext**	
Description	According to NGSI10 specification, this method provides asynchronous retrieval of specified context information. Based on NGSI and the ORION context broker provided by FI-WARE, SocIoTal Context Manager will provide 3 different notification triggers: - **ONTIMEINTERVAL**: will send a notification when the time interval specified in the "notifyConditions" element is reached. - **ONCHANGE**: provides a notification every time a change in the specified context attributes happens - **ONVALUE** (currently not implemented)	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/subscribeContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]
Payload: [ONCHANGE]		

```
{
   "entities": [{
      "type": " urn:x-org:sociotal:resource:device ",
      "isPattern": "false",
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001"
   }],
   "reference":
"http://capela.unican.es:1650/Callback_Apps/Listeners/LogWriterFile",
   "duration": "P1M",
   "notifyConditions": [{
      "type": "ONCHANGE",
      "condValues": ["AmbientTemperature"]
   }],
   "throttling": "PT5S"
}
```

Payload: [ONTIMEINTERVAL]

```
{
   "entities": [{
      "type": "urn:x-org:sociotal:resource:device",
      "isPattern": "false",
      "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001"
   }],
   "attributes": ["HumidityValue"],
   "reference":
"http://capela.unican.es:1650/Callback_Apps/Listeners/LogWriterFile",
   "duration": "P1M",
   "notifyConditions": [{
      "type": "ONTIMEINTERVAL",
      "condValues": ["PT10S"]
   }]
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

- **“entities”** and **“attributes”** define which context elements will be included in the notification message. They work the same way as in the queryContext request, so lists of attributes or patterns in the entity id can be included to specify a set of entities and/or attributes the requestor would like to be subscribed.

- **"reference"** element specifies the callback function where the Context Manager is going to send the notifications (when happen). It will POST a JSON object including the context information to the URL pointed here.
- **"duration"** gives the subscription duration, specified using the ISO 8601 standard format, for example "P1M" means 1 moth.
- "condValues" specifies:
    - ONCHANGE subscription: the attribute or set of attributes that, when changing, triggers a notification.
    - ONTIMEINTERVAL subscription: the time between received notifications.
- **"throttling"** specifies a minimum inter-notification arrival time. So, setting throttling to 5 seconds as in the example above makes that a notification will not be sent if a previous notification was sent less than 5 seconds ago, no matter how many actual changes take place in that period.

Response JSON:

```
{
  "subscribeResponse" : {
     "duration": "P1M",
     "throttling": "PT5S",  //→ if "ONCHANGE" type is selected
     "subscriptionId" : "51de3b21e7256c1b36dad955",
  }
}
```

The response given by the SocIoTal context manager will include the subscription ID, used lately to modify and/or delete it.

Name	unsubscribeContext	
Description	This method deletes the context subscription pointed by the "subscriptionId" element of the payload.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/unsubscribeContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]
Payload:		
{		

```
   "subscriptionId" : "51de3b21e7256c1b36dad955"
}
```

Error Messages (Status codes):	
errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the subscriptionId element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

Name	**updateContextSubscription**	
Description	Subscriptions can be updated using this NGSI10 updateContextSubscription method. The request includes a subscriptionId that identifies the subscription to modify, and the actual update payload, including the new values of the subscription the requester would like to change.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: POST	/SocIoTal_CM_REST_V2/NGSI10_API/updateContextSubscription	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]

Payload: [ONCHANGE] e.g. changes the duration, the attribute triggering the notification and the throttling

```
{
   "subscriptionId" : "51de3b21e7256c1b36dad955",
   "duration": "P2M",
   "notifyConditions": [{
      "type": "ONCHANGE",
      "condValues": ["HumidityValue"]
   }],
   "throttling": "PT10S"
}
```

Payload: [ONTIMEINTERVAL] e.g. changes the time interval

```
{
   "subscriptionId" : "51de3b21e7256c1b36dad955",
   "notifyConditions": [{
      "type": "ONTIMEINTERVAL",
      "condValues": ["PT20S"]
   }]
}
```

Error Messages (Status codes):

errorCode element (Response body)	Message/Additional info
200 “OK”	The request has been properly executed and the result has been included in the response payload. (Initial versions won’t show error code in this case)
400 “BAD REQUEST”	Any of the fields have been not properly performed. The response payload will include further info if available.
401 “UNAUTHORIZED”	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
404 “NOT FOUND”	No context elements found (Check the subscriptionId element)
500 “INTERNAL SERVER ERROR”	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	**Message/Additional info**
405 “METHOD NOT ALLOWED”	Check you’re using POST
415 “UNSUPPORTED MEDIA TYPE”	Check “Content-type application/json” header and the payload
500 “INTERNAL SERVER ERROR”	Error accessing Context Manager server/service

• *What can be updated in a Context Subscription?*

Not all the fields can be updated through the updateContextSubscription request. In particular, the following fields can’t be updated:

- subscriptionId (you must include it in the request but you can’t change it)
- entityIDList
- attributeList
- reference

The following fields can be modified:

- notifyConditions
- throttling

- duration
- restriction

2.6. Extended methods

In addition to the NGSI9 and 10 interfaces, SocIoTal CM implements a set of extended functionalities to facilitate the use of its features. Version 2 of the SocIoTal CM's Extended methods API is represented by the URI:

SocIoTal CM V2	**/SocIoTal_CM_REST_V2/EXTENDED/{method}/{params}**

Name	**queryContext by entityID**	
Description	Retrieval of Context Information. This method (not NGSI native) will retrieve the whole current context of the Entity ID passed as an http parameter.	
SocIoTal SERVER	SocIoTal_CM_V2_IP:PORT	
Method: GET	/SocIoTal_CM_REST_V2/EXTENDED/queryContext/Entity_ID	
HEADERS		
	Content-type	application/json
	Accept	application/json
	Capability-token	[JSON file/text including SocIoTal Capability token - see example at 3.1.1]
Error Messages (Status codes):		

errorCode element (Response body)	Message/Additional info
200 "OK"	The request has been properly executed and the result has been included in the response payload. (Initial versions won't show error code in this case)
404 "NOT FOUND"	No context elements found (Check the Entity_ID param)
401 "UNAUTHORIZED"	The Capability-token is either corrupted, not valid or does not contain the appropriated credentials
500 "INTERNAL SERVER ERROR"	Error accessing Context Broker element of the Context Manager
Status Code (returned by the server)	Message/Additional info
405 "METHOD NOT ALLOWED"	Check you're using POST
500 "INTERNAL SERVER	Error accessing Context Manager server/service

ERROR”	

Call	<sociotal_server>:<port>/SocIoTal_Context_UC_REST/NGSI10_API/queryContext/SocIoTal:IoTWeek:WeatherStation:Dev_001	
HTTP Headers	Content-type	application/json
	Accept	application/json
	Capability-token	

```
{
      "de": "http://193.144.201.50:3500",
      "id": "geqe3k0pl1oj4i14idhuqg8am4",
      "is": "capabilitymanager@um.es",
      "na": 1563534433,
      "nb": 1433534333,
      "su":
"QKkWEGvhwkn4wubbkASz6DT04ukliJbOXkDGCcqCLdk=TX+91sWv/3eZP5fw
jO7wv0x4+FD6uRtOcBRGLwjkWCo=",
      "ii": 1433534333,
      "ar": [{
   "ac": "GET",
   "re":
"/SocIoTal_CM_REST_V2/NGSI10_API/queryContext/SocIoTal:SAN:We
atherStation:Dev_001"
                  }],
      "si":
"MEUCIQDCJDKXp9RkYZLkmge/vFfzFTcjtTobVi2ypSwkmW+t/QIgBpWRaL61
Ya6LFOhhZ0QyUjCvAxiPBuLAX6yLbEVeh40="
}
```

Response JSON:

```
{
   "errorCode": {
      "details": "Count: 1",
      "reasonPhrase": "OK",
      "code": "200"
   },
   "contextResponses": [{
      "statusCode": {
         "reasonPhrase": "OK",
         "code": "200"
      },
      "contextElement": {
         "id": "SocIoTal:IoTWeek:WeatherStation:Dev_001",
         "attributes": [{
            "name": "AmbientTemperature",
            "value": "25.44",
            "type": "http://sensorml.com/ont/swe/property/AmbientTemperature",
            "metadatas": [{
               "name": "DateTimeStamp",
               "value": "20141030T113343Z",
               "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
            },
            {
               "name": "Unit",
               "value": "celsius",
               "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
            },
            {
               "name": "accuracy",
               "value": "0,5",
              "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
           },
                                                                       {
              "name": "DataDescription",
              "value": "float",
              "type": "http://sensorml.com/ont/swe/property/DataDescription"
           }]
        },
        {
           "name": "HumidityValue",
           "value": "59",
           "type": "http://sensorml.com/ont/swe/property/HumidityValue",
           "metadatas": [{
              "name": "DateTimeStamp",
              "value": "20141030T113343Z",
              "type": "http://sensorml.com/ont/swe/property/DateTimeStamp"
           },
           {
              "name": "Unit",
              "value": "percentage",
              "type": "http://purl.oclc.org/NET/ssnx/qu/qu#Unit"
           },
           {
              "name": "accuracy",
              "value": "1",
              "type":
"http://sensorml.com/ont/swe/property/QuantitativeAttributeAccuracy"
           },
                                                                       {
              "name": "DataDescription",
              "value": "integer",
              "type": "http://sensorml.com/ont/swe/property/DataDescription"
           }]
        },
        {
           "name": "Location",
           "value": "43.472057, -3.800156",
           "type": "http://sensorml.com/ont/swe/property/Location",
           "metadatas": [{
              "name": "WorldGeographicReferenceSystem",
              "value": "WSG84",
              "type":
"http://sensorml.com/ont/swe/property/WorldGeographicReferenceSystem"
           }]
        }],
        "type": "urn:x-org:sociotal:resource:device",
        "isPattern": "false"
     }
  }]
}
```

Section 3. Security & Privacy

3.1. Authorization and Authentication API

The SocIoTal access control system is designed as a combination of different authorization technologies and tools in order to enable a suitable solution for IoT environments. Such system is based on the use of XACML access control policies, which are employed to generate authorization credentials in the form of capability tokens. Then, such tokens are used by smart objects to get access to services being provided by other IoT entities. It should be noticed that the required exchange to get and use such tokens is carried out through the use of a certificate-based mutual authentication by using DTLS for authentication purposes. Additionally, this process could be made in a privacy-preserving way by using the API that is explained in Section 3.3. For additional information about the process, see SocIoTal D2.2.

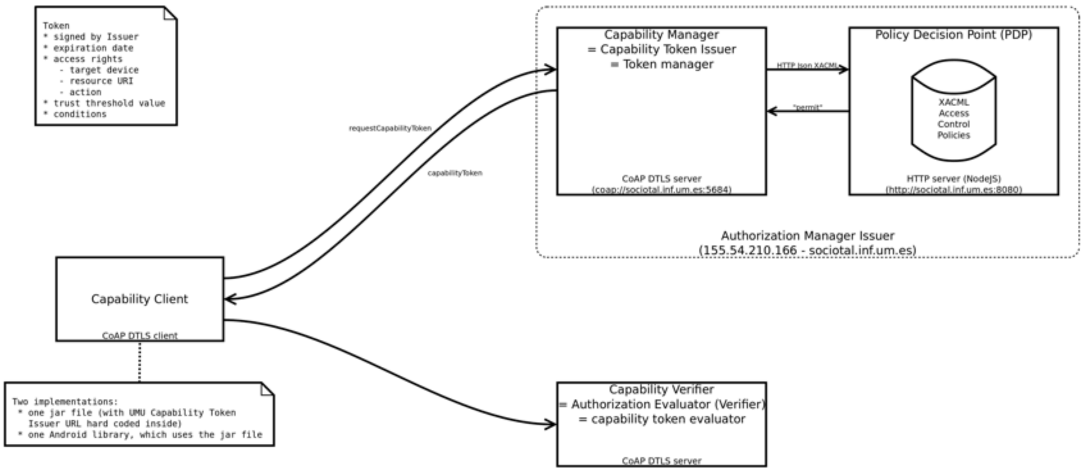
Token
* signed by Issuer
* expiration date
* access rights
- target device
- resource URI
- action
* trust threshold value
* conditions
requestCapabilityToken
capabilityToken
Capability Manager
= Capability Token Issuer
= Token manager
CoAP DTLS server
(coap://sociotal.inf.um.es:5684)
HTTP Json XACML
"permit"
Policy Decision Point (PDP)
XACML Access Control Policies
HTTP server (NodeJS)
(http://sociotal.inf.um.es:8080)
Authorization Manager Issuer
(155.54.210.166 - sociotal.inf.um.es)
Capability Client
CoAP DTLS client
Two implementations:
* one jar file (with UMU Capability Token Issuer URL hard coded inside)
* one Android library, which uses the jar file
Capability Verifier
= Authorization Evaluator (Verifier)
= capability token evaluator
CoAP DTLS server

Figure 5. The SocIoTal access control system

3.1.1. Capability Token example

Before the explanation of the main APIs of the authorization components, below, for the sake of clarity, a capability token example is shown:

```
{
  "id": "cvv96ql9ju6bj6m27u68j32a4n",
  "ii": 1436955119,
  "is": "capabilitymanager@um.es",
  "su":
"QKkWEGvhwkn4wubbkASz6DT04ukliJbOXkDGCcqCLdk=TX+91sWv/3eZP5fwjO7wv0x4+FD6uRtOcBRGLw
jkWCo=",
  "de": "http://193.144.201.50:3500",
  "si":
"MEUCIQDZGjJoZLY5TIajlSdGmOoxrE6fcJt0oGfUE9YEMSWPLwIgTzgDydsxe52UmX8oON5BSJYsr2Rw2G
XVh53lTOqDbTA=",
  "ar": [
{
      "ac": "GET",
      "re":
"/SocIoTal_CM_REST_V2/NGSI10_API/queryContext/SocIoTal:SAN:WeatherStation:Dev_001"
}
  ],
  "nb": 1436955119,
  "na": 1436955219
}
```

In addition, a brief description of each field is provided:

- **Identifier** (id). This field is used to unequivocally identify a capability token. A random or pseudorandom technique will be employed by the issuer to ensure this identifier is unique.
- **Issued-time** (ii). It identifies the time at which the token was issued as the number of seconds from 1970-01-01T0:0:0Z.
- **Issuer** (is). The entity that issued the token and, therefore, the signer of it.
- **Subject** (su). It makes reference to the subject's public key to which the rights from the token are granted.
- **Device** (de). It is a URI used to unequivocally identify the device to which the token applies.
- **Signature** (si). It carries the digital signature (ECDSA) of the token encoded through Base64.
- **Access Rights** (ar). This field represents the set of rights that the issuer has granted to the subject.
  - Action (ac). Its purpose is to identify a specific granted action. Its value makes reference a CoAP/HTTP method (e.g. GET, POST).
  - Resource (re). It represents the resource in the device for which the action is granted.
- **Not Before** (nb). The time before which the token must not be accepted. Its value cannot be earlier than the II field and it implies the current time must be after or equal than NB.
- **Not After** (na). It represents the time after which the token must not be accepted.

3.1.2. *Capability Client API*

The *Capability Client* library is a CoAP-DTLS client making requests to the *Capability Manager* to obtain capability tokens, which are used to get access to resources hosted in a *Capability Verifier*. .

The basic configuration of the library is included in the CapabilityClient_config.txt file:

- **capability_tokens_folder**: the path of the folder where capability tokens are stored
- **certs**: the path of the folder where certificates and cryptographic material (e.g. java keystores) are stored

Name	**tokenIsValid**
Description	This method checks if a capability token is valid (i.e. it is not expired)
Parameters	- *CapabilityToken capability_token*. The capability token to be checked
Return value	Boolean. *true* if the token is valid, *false* if it is not

Name	**ownToken**
Description	It checks if the capability client already has a capability token for the action and resource being specified
Parameters	- *String coap_http_method*. The requested action. This parameter can take the value of a CoAP method (e.g. GET) - *String resource_uri*. The URI of the resource on which the action is requested. For example: coap://localhost:5684/bubble
Return value	The Capability Token object in case it exists. Otherwise, it returns a null value

Name	**requestCapabilityToken**
Return value	The Capability Token object in case it is obtained. Otherwise, it returns a null value

Description	It requests a capability token for a specific action and resource to the Capability Manager. In case of affirmative authorization decision, a capability token is received and stored in the capability tokens folder. This folder is specified in the CapabilityClient_config.txt file
Parameters	- *File keystore*. The path where the keystore is. Currently, two keystores are included under the certificates folder. This folder is specified in the CapabilityClient_config.txt file - *String coap_http_method*. The action for which the capability token is requested. For the time being, this parameter can take

	the values "GET" or "POST" - *String resource_uri*. The URI related to the resource for which the capability token is requested. For example: coaps://localhost:5684/bubble - *String ipCapabilityManager*. The IP address of the Capability Manager to get a capability token.
Name	**requestCapabilityToken**
Description	It requests a capability token for a specific action and resource to the Capability Manager. In case of affirmative authorization decision, a capability token is received and stored in the capability tokens folder. This folder is specified in the CapabilityClient_config.txt file
Parameters	- *String keystore*. The path where the keystore is. Currently, two keystores are included under the certificates folder. This folder is specified in the CapabilityClient_config.txt file - *String coap_http_method*. The action for which the capability token is requested. For the time being, this parameter can take the values "GET" or "POST" - *String resource_uri*. The URI related to the resource for which the capability token is requested. For example: coaps://localhost:5684/bubble - *String ipCapabilityManager*. The IP address of the Capability Manager to get a capability token.
Return value	The Capability Token object in case it is obtained. Otherwise, it returns a null value

Name	**getAccess**
Description	It makes a CoAPS request to perform a specific action on a particular resource, by making use of capability token
Parameters	- *CapabilityToken token*. The capability token to be used to get access - *String keystore*. The path where the keystore is. Currently, two keystores are included under the certificates folder. This folder is specified in the CapabilityClient_config.txt file. - *String coap_http_method*. The requested action. For the time being, this parameter can take the values "GET" or "POST" - *String resource_uri*. The URI of the resource on which the action is requested. For example: coaps://localhost:8888/bubbleA - *String payload*. The payload of the request in case it is needed.
Return value	String. The message from the Capability Verifier component

Name	getAccess
Description	It makes a CoAPS request to perform a specific action on a particular resource, by making use of capability token
Parameters	- *CapabilityToken token*. The capability token to be used to get access - *File keystore*. The file path where the keystore is. Currently, two keystores are included under the certificates folder. This folder is specified in the CapabilityClient_config.txt file. - *String coap_http_method*. The requested action. For the time being, this parameter can take the values "GET" or "POST" - *String resource_uri*. The URI of the resource on which the action is requested. For example: coaps://localhost:8888/bubbleA - *String payload*. The payload of the request in case it is needed.
Return value	String. The message from the Capability Verifier component

3.1.2.1. Usage example

You can find below an example code about the usage of the library:

```
try{
   File sdcard = Environment.getExternalStorageDirectory();
   File file = new File(sdcard,"CapabilityClient_config.txt");

   CoAPSCapabilityClient cc = new CoAPSCapabilityClient(file.getAbsolutePath());

   CapabilityToken ct = cc.ownToken("GET", "coap://localhost:5684/bubbleA");

   File keystore = new File ("resources/umu.bks");

   if (ct == null || !cc.tokenIsValid(ct))
      ct = cc.requestCapabilityToken(
            keystore,
              "GET",
            “coap://localhost:5684/bubbleA",
            “155.54.210.166);

   else{
      cc.getAccess(
            ct,
            keystore,
            "GET", " coap://localhost:5684/bubbleA"",
            null);
   }
}
```

In this example, firstly the client checks if it has a capability token for the action “GET” and resource “coap://localhost:5684/bubbleA”. If not (or the token that is found is not valid (i.e. it is expired), it requests a token to the Capability Manager, which is hosted at 155.54.210.166.

In case the client already has a valid token, it uses it through the getAccess method to perform the action “GET” over the resource “coap://localhost:5684/bubbleA”.

3.1.3. Capability Verifier API

The *Capability Verifier* is a CoAP-DTLS server library receiving access requests from Capability Clients. Such access requests contain a capability token, which is evaluated by the Capability Verifier in order to deny or grant the requesting action. The library of the Capability Verifier also provides support for Android devices. It should be noticed that this library contains the functionality of the Capability Verifier entity and Capability Evaluator (see Section 3.2.3). That is, on the one hand, it can be used as a Capability Server which receives access requests with capability tokens, and evaluates this credential. On the other hand, it can be employed for evaluating capability tokens, by making use of the CapabilityEvaluator class.
The basic configuration of the library is included in the CapabilityServer_config.txt file:

- **certs**. It contains the cryptographic material required for secure communications with certificates examples.

Name	**actionInCaseActionNotPermitted**
Description	Abstract method to specify the process in case the token evaluation result was CapabilityVerifierCode.ACTION_NOT_PERMITTED
Parameters	- *String action*. The action that was requested by a Capability Client - *String resource*. The resource that was requested by a Capability Client
Return value	String. The message response from the Capability Verifier that is included in the payload of the CoAP response

Name	**actionInCaseAuthorized**
Description	Abstract method to specify the process in case the token evaluation was successful
Parameters	- *String action*. The action that was requested by a Capability Client - *String resource*. The resource that was requested by a Capability Client
Return value	String. The message response from the Capability Verifier that is included in the payload of the CoAP response

Name	**actionInCaseNotValidSignature**
Description	Abstract method to specify the process in case the token evaluation result was CapabilityVerifierCode.SIGNATURE_NOT_VALID
Parameters	- *String action*. The action that was requested by a Capability Client - *String resource*. The resource that was requested by a Capability Client

Return value	String. The message response from the Capability Verifier that is included in the payload of the CoAP response

Name	**actionInCaseNotValidToken**
Description	Abstract method to specify the process in case the token evaluation result was CapabilityVerifierCode.TOKEN_NOT_VALID
Parameters	- *String action*. The action that was requested by a Capability Client - *String resource*. The resource that was requested by a Capability Client
Return value	String. The message response from the Capability Verifier that is included in the payload of the CoAP response

Name	**Initialize**
Description	It initializes the CoAPS Capability Server with a set of resurces
Parameters	- *String [] resources*. The set of resources that are going to be hosted in this Capability Server

3.1.3.1. Usage example

You can find below an example code about the usage of the library:

```
package org.umu.capabilityverifier.tests;

import java.io.IOException;
import org.umu.capabilityverifier.CoAPSCapabilityServer;

public class CoAPSCapabilityVerifierTest {
   public static void main(String[] args) {
      CoAPSCapabilityServer cv = new
CoAPSCapabilityServer("CapabilityServer_config.txt") {

         @Override
         public String actionInCaseNotValidToken(String action, String resource) {
            // TODO Auto-generated method stub
            return "Token is not valid";
         }

         @Override
         public String actionInCaseNotValidSignature(String action, String resource)
{
            // TODO Auto-generated method stub
            return "Signature is not valid";
         }

         @Override
         public String actionInCaseAuthorized(String action, String resource) {
            return "You are authorized";
         }

         @Override
         public String actionInCaseActionNotPermitted(String action, String resource)
{
            return "Action is not permitted";
         }
      };
      String [] resources = {"bubbleA"};
      cv.initialize(resources);
      try {
         System.in.read();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```
In this example, the Capability Verifier is initialized with the resources “bubbleA”, so it will listen only requests for that resource.

3.1.4. Capability Evaluator API

This class is used by the Capability Verifier to evaluate a capability token. In addition, it can be used independently.

Name	evaluateCapabilityToken
Description	It evaluates a capability token taking into account the action and resource requested
Parameters	- *String capability_token*. The capability token to be evaluated - *String action*. The action that was requested by a Capability Client - *String resource*. The resource that was requested by a Capability Client
Return value	A CapabilityVerifierCode indicating the result of the evaluation

3.1.4.1. Usage example

You can find below an example code about the usage of the library:

```
package org.umu.capabilityverifier.tests;

import java.io.IOException;
import org.umu.capabilityverifier.CapabilityEvaluator;

public class CapabilityEvaluatorTest {
   public static void main(String[] args) {
      String certs_folder = "certs";
      CapabilityEvaluator ce = new CapabilityEvaluator(certs_folder);
      ce.evaluateCapabilityToken(capability_token, "GET", "temperature");
   }
}
```

According to the example, it should be noticed that it is necessary to extract the capability token from the request and using that as a parameter of the function. In addition, please note that it is required to indicate the folder where the cryptogtaphic material is stored when you construct the Capability Evaluator. This folder is the same that is indicated in the CapabilityServer_config.txt file

3.2. Group Manager API

The SocIoTal Group Manager component is based on the CP-ABE cryptographic as a flexible scheme to enable a secure group data sharing mechanism. The Group Manager client API allows obtaining cryptographic material, encrypt, decrypt as well as sharing encrypted information through the Context Manger. A detailed overview of this process can be found in SocIoTal D3.3.

The basic configuration of the library is included in the GroupManagerClient_config.txt file:

- **sharing_material_folder**: the path of the folder where CP-ABE keys are other parameters required for the Group Manager functionality are stored
- **certs**: the path of the folder where certificates and cryptographic material (e.g. java keystores) are stored

Name	getSharingKey
Description	It requests a CP-ABE key to an Attribute Authority. In case of a successful authentication, a new CP-ABE key is received and stored in the sharing keys folder. This folder is specified in the GroupManagerClient_config.txt file
Parameters	- *File keystore*. The file path where the keystore is. Currently, two keystores are included under the certificates folder. This folder is specified in the GroupManagerClient _config.txt file - *String ipAttributeAuthority*. The IP address of the Attribute Authority to get a CP-ABE key.
Return value	The CP-ABE object in case it is obtained. Otherwise, it returns a null value

Name	encryptData
Description	It encrypt a piece of data according to the CP-ABE policy being specifed
Parameters	- *String public_parameters_file*. The file path where the public parameters required by CP-ABE are stored. keystore is. This folder is specified in the GroupManagerClient _config.txt file - *String policy*. It indicates the combination of identity attributes that must be satisfied by data consumers. The format to specify CP-ABE policies is: (attributevalue (attributevalue)+ **mofn**)+, where **m<n** means **OR** operation and **m=n** means **AND** operation between attributes - String data_value. The data to be encrypted
Return value	A String representing the encrypted data encoded with Base64. It will return a null value in case there was a problem to encrypt data.

Name	decryptData
Description	It tries to decrypt a piece of data by using a specific CP-ABE key
Parameters	- *String public_parameters_file*. The file path where the public parameters required by CP-ABE are stored. keystore is. This folder is specified in the GroupManagerClient _config.txt file. - *String private_key*. The file where the CP-ABE key to decrypt the information is stored inside the sharing_material_folder. This folder is specified in the GroupManagerClient _config.txt file. - String encrypted_data. A String representing the encrypted data encoded with Base64
Return value	A String representing the decrypted data. It will return a null value in case the private_key does not satisfy the policy that was used to encrpyt.

Name	updateContextEncrypted	
Description	It updates the value of an entity in the Context Manager by encrypting the value through the use of the encryptData method	
SocIoTal Server	<sociotal_server>:<port>	
Method	POST	
	/SocIoTal_Context_UC_REST/NGSI10_API/updateContext	
Headers		
	Content-type	application/json
	Accept	application/json

Payload:

```
{
   "contextElements": [
   {
      "type": "urn:x-org:sociotal:resource:device",,
      "isPattern": "false",
      "id": "SocIoTal:UMU:VirtualDeviceContext",
      "attributes": [
      {
         "name": "temperature",
         "type": "string",
         "value": "encrypted_value",
         "metadatas": [
         {
            "name": "cph",
            "type": "string",
            "value":
"w7FoZXVieTJvOGZ5MzQ5OHlmbnBmw7FodWk0Znk4MzR5bmYwOHkzZmh1aTNodnVpaGh2aW9qZm05d2U5dWZ
tOTgzdTk4ZnkzOG5meSAzNGpmbzM0w7JpZmhkd3V5ZmQ3NjJyZTg3NzIww7JwZGtqa2NuZ3Y4NzN5cDg5Zmh
1bzN2aGdvdXlnOHRnNzk4eWAzdWYyw61oMmJweWNnOHAyY2d1dnBiM3Vicm5qa25scG8wOXU4eTM3dDgyNjV
yMjY3Mzg3eTk4dWRpb2hqa25samhjYnZnanlmYWdzb2hwb8Oxd3BpYHJ1OTA4eTlnZWZ5aXZoYjNranJsbnZ
icHUzb2d2dWliM2prcmxudsOxY21ybGVmMzR1aXVoZmpvaTNqcG8zNCxmM3Yz"
         },
         ]
      }
      ]
   }
   ],
   "updateAction": "UPDATE"
}
```

3.3. Identity Manager

The Identity Management (IdM) system follows a claim-based approach with Attribute Based Credentials (ABC). The IdM relies on the Idemix cryptographic library providing additional means to deal with IoT scenarios where consumers and providers' can be not only traditional

computers, but also smart objects (e.g. smartphones). The IdM endows users and smart objects with means to control and manage their private data in their smartphone, defining partial identities over their whole identity, which is derived from the credential obtained from the Issuer. The usage of partial identities ensures a privacy-preserving solution with minimal disclosure of personal information. Unlike in traditional IdMs, the subject smart object is not redirected to its online Identity Provider (IdP) during the transaction, so that the IdP is not involved when the target device verifies the smart object's attributes. A detailed description of the IdM can be found in deliverable D2.1.
The IdM is composed of three main components.

1. SocIoTal-IdM-Android-Client: It is an android application that allows obtaining Idemix credentials from the Issuer server. It also allows interact with the Verifier server which can validate the partial identity derived from the credential.

2. SocIoTal-Issuer-Server: It is a web application implemented with Java servlets and XML-RPC which allows generating Idemix credentials for clients. Communications are done by https. The client must be authenticated against the Issuer using a valid certificate. The Issuer also support the verification functionality.

3. SocIoTal-Verifier-Server: It is a web application, also implemented with Java servlets and XML-RPC, which is able to validate partial identities presented by the client application.

Notice that the IdM provides means for privacy-preserving authentation, since users can employ their partial identities to authenticate against the verifier, following the Idemix verficiation protocol.

The following figure shows the main components and their interactions.

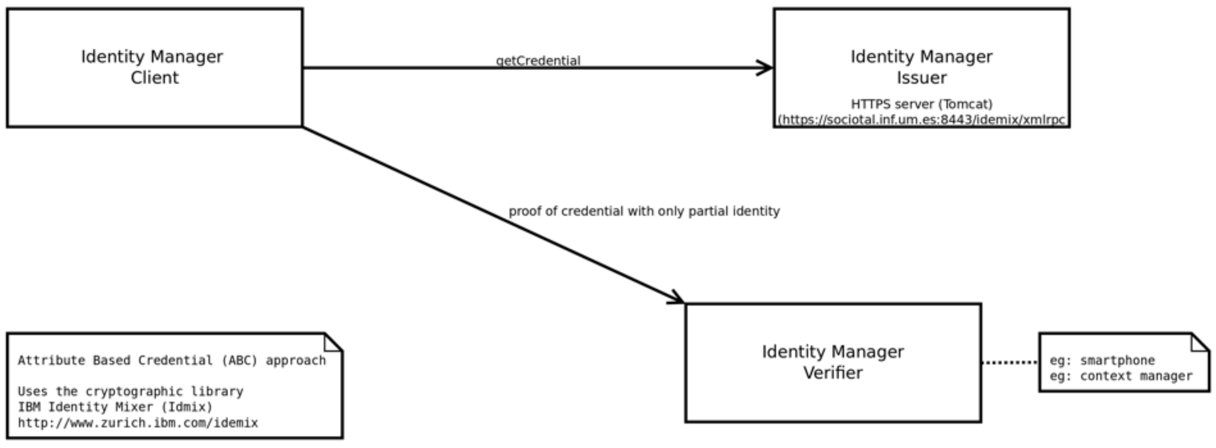
Identity Manager Client
getCredential
Identity Manager Issuer
HTTPS server (Tomcat)
(https://sociotal.inf.um.es:8443/idemix/xmlrpc
proof of credential with only partial identity
Attribute Based Credential (ABC) approach
Uses the cryptographic library
IBM Identity Mixer (Idmix)
http://www.zurich.ibm.com/idemix
Identity Manager Verifier
eg: smartphone
eg: context manager

Figure 6. Identity Manager components and interactions

3.4. Client API

The IdM client API is provided by java libary that can be used mainly for:

1. Obtain an Idemix credential from the Issuer.
2. Validate a partial identity (Idemix proof) derived from the obtained Idemix credential

Idemix cryptographic library requires the recipient (the client user) to share the same Idemix system parameters and group parameters with the Issuer Server. These parameters are public and accessible by two configuration files. The SocIoTal IdM relies on the Idemix cryptographic library v2.3.0. For further information about Idemix and the library, please, refer to [7]

Name	**IssuanceCredential**
Description	It allows obtaining an Idemix credential from the Issuer server. The crendential follows an data model structure given by the first parameter. The attributes values to be included in the credential are passed in the second parameter
Method	BigInteger nonce issuanceCredential(String credStructLocation, Values values)
Parameters	-*credStructLocation*→ the credential structure - *values*→the attributes values to be included in the credential
Return value	The cryptographic credential from the issuer (a binary file), which is given a common name to be stored in the client side to be employed later on.

Request Credential structure example	`<?xml version="1.0" encoding="UTF-8"?>`
	`<ProofSpec xmlns="http://www.zurich.ibm.com/security/idemix"`
	`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`
	`xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix`
	`xsd/ProofSpec.xsd">`
	`<Declaration>`
	`<AttributeId name="id1" proofMode="unrevealed" type="int" />`
	`<AttributeId name="id2" proofMode="unrevealed" type="int" />`
	`</Declaration>`
	`<Specification>`
	`<Credentials>`
	`<Credential name="someRandomName"`
	`credStruct="http://sociotal.inf.um.es/idemix/files/credstruct`
	`1a.xml" >`
	`<Attribute name="attr1">id1</Attribute>`
	`<Attribute name="attr2">id2</Attribute>`
	`</Credential>`
	`</Credentials>`
	`<EnumAttributes />`
	`<Inequalities />`
	`<Commitments />`
	`<Representations />`
	`<Pseudonyms />`
	`<VerifiableEncryptions />`

	`<Messages />`
	`</Specification>`
	`</ProofSpec>`

Name	**VerifyProof**
Description	It allows verifying an Idemix cryptographic proof (partial identity) generated by the client based on a credential obtained previouly from the Issuer server. The proof follows an data model structure given by the first parameter
Method	Boolean verifyProof(String proofSpecification, String credentialName)
Parameters	-*proofSpecification*→ the proof specification structure (partial identity) with the attributes to be verified by the Verifier entity - *credentialName*→the credential name among the ones obtained (and stored) previously the client.
Return value	Indicates whether the proof is valid or not according to the proof specification and the credential passed as parameters

Request proof structure example	`<?xml version="1.0" encoding="UTF-8"?>`
	`<ProofSpec xmlns="http://www.zurich.ibm.com/security/idemix" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix xsd/ProofSpec.xsd">`
	
	`<Declaration>`
	`<AttributeId name="id1" proofMode="unrevealed" type="int" />`
	`<AttributeId name="id2" proofMode="unrevealed" type="int" />`
	`</Declaration>`
	
	`<Specification>`
	`<Credentials>`
	`<Credential name="someRandomName" credStruct="http://sociotal.inf.um.es/idemix/files/credstruct1a.xml" >`
	`<Attribute name="attr1">id1</Attribute>`
	`<Attribute name="attr2">id2</Attribute>`
	`</Credential>`
	`</Credentials>`
	`<EnumAttributes />`
	`<Inequalities />`
	`<Commitments />`
	`<Representations />`
	`<Pseudonyms />`
	`<VerifiableEncryptions />`
	`<Messages />`
	`</Specification>`
	`</ProofSpec>`

3.5. Issuer API

The Issuer server is a web application implemented with Java Servlets and XML-RPC which allows generating Idemix credentials for clients. Communications are done by https.
The Issuer requires of a Web container like Apache Tomcat. The Tomcat must be configured to trust the CA.

The Issuer server is release as a java war application that has to be deployed in the /webapps folder of the Tomcat server for its installation.

Name	**InitiateIssuanceCredential**
Description	Given a credential structure and a set of attributes provided by the client, the Issuer validates that the client satisfices the attribute, if so it generates a nonce that is send back to the client. To validate the client attributes, the current version requires that the client includes the attributes values in its certificate that is signed by the CA. This method is invoked by the method IssuanceCredential of the IdM Client API. The credential structure passed in the request is based on the XML Schema of the Idemix credential structure.
SocIoTal Server	https://sociotal.inf.um.es:8443/idemix/xmlrpc
Method	BigInteger initiateIssueCredential(**byte**[] values, String *credStructLocation*)
Parameters	-*credStructLocation*→ the credential structure location used to know the attributes and their types to be include in the credential - *values*→the attributes values to be included in the credential
Return value	a random value that is created by the issuer for each incoming request to avoid reply attacks.

Request Credential structure example	`<CredentialStructure xmlns="http://www.zurich.ibm.com/security/idemix/credentialStructure" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix/credentialStructure ./xsd/CredentialStructure.xsd">`
	`<References>`
	`<IssuerPublicKey>http://sociotal.inf.um.es/idemix/files/ipk.xml</IssuerPublicKey>`
	`</References>`
	`<Attributes>`
	`<Attribute issuanceMode="known" name="attr1" type="int" />`
	`<Attribute issuanceMode="known" name="attr2" type="int" />`
	
	`</Attributes>`
	
	`<Features/>`

	`<Implementation>`
	`<AttributeOrder>`
	`<Attribute name="attr1">1</Attribute>`
	`<Attribute name="attr2">2</Attribute>`
	`</AttributeOrder>`
	`</Implementation>`
	`</CredentialStructure>`

Name	**issuanceComputeRound2**
Description	It receives as input an Idemix cryptographic message which has been computed previously by the client. Such a message is constructed based on the once obtained previously invoking the initiateIssuanceCredential method. As a result it generates a cryptographic message representing the credential.
SocIoTal Server	https://sociotal.inf.um.es:8443/idemix/xmlrpc
Method	**byte**[] computeRound2(**byte**[] message)
Parameters	-*message*→ cryptographic message created by the Idemix client (or recipient) based on the credential structure, the attribute values and the nonce previously obtained
Return value	The cryptographic credential (binary file) with the Issuer CL signature, which will allow the client to demonstrate that he is in possession of a valid credential with certain attributes specified in the credential structure.

3.6. Verifier API

The verifier server is a web application based on XML-RPC that allows validating partial identities, i.e. validate Idemix proofs related to an Idemix credential obtained from the Issuer server.

Idemix requires agreeing a particular proof specification, between the client and the Verifier. In other words, both entities must agree on a specific structure of the partial identity, with the attributes from the full credential that are going to be shown and verified in the verification process.

The proof specification is described following the Idemix ProofSpec XML schema. The SocIoTal provides a predefined set of credential specifications, which are available at http://sociotal.inf.um.es/idemix/files/

Name	initiateVerifier
Description	This method allows initiate the Idemix verification protocol. It returns a nonce from the verifier server given a particular set of system parameters.
SocIoTal Server	https://sociotal.inf.um.es:8443/idemix/xmlrpc
Method	BigInteger initiateVerifier()
Return value	A nonce from the verifier server given a particular set of system parameters.

Name	verifyProof
Description	This method allows verifying an Idemix proof of having a credential issued by the Issuer. It requires as input a proof cryptographic message with the CL proof and the proof specification. The proof cryptographic message can contain a set of identity attributes with their values. Notice that this method cannot been invoked unless the initiate verifier method has been already invoked. This is controlled by the Verifier by means of the HTTP Session. Moreover, the nonce is different for each verification.
SocIoTal Server	https://sociotal.inf.um.es:8443/idemix/xmlrpc
Method	**boolean** verifyProof(**byte**[] proof, **byte**[] proofSpecification)
Parameters	-*proof*→ the cryptographic proo generated by the client (recipient) given the proof specification, the nonce and the credential obtained before. -*proofSpecification*→ the proof specification indicating the structure of the partial indentity, that is, the attributes and their values (from the credential) which are going to be demonstrated.

Request proof structure example	`<?xml version="1.0" encoding="UTF-8"?>`
	`<ProofSpec xmlns="http://www.zurich.ibm.com/security/idemix" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix xsd/ProofSpec.xsd">`
	`<Declaration>`
	`<AttributeId name="id1" proofMode="unrevealed" type="int" />`
	`<AttributeId name="id2" proofMode="unrevealed" type="int" />`
	`</Declaration>`
	`<Specification>`
	`<Credentials>`
	`<Credential name="someRandomName" credStruct="http://sociotal.inf.um.es/idemix/files/credstruct1a.xml" >`
	`<Attribute name="attr1">id1</Attribute>`
	`<Attribute name="attr2">id2</Attribute>`
	`</Credential>`
	`</Credentials>`
	`<EnumAttributes />`
	`<Inequalities />`

	`<Commitments />`
	`<Representations />`
	`<Pseudonyms />`
	`<VerifiableEncryptions />`
	`<Messages />`
	`</Specification>`
	`</ProofSpec>`

3.7. Trust & Reputation

Trust and Raputation Management is a process enabled by Trust Manager - mixture of the REST webservice and logic with a set of different rules for building a reputation score. Generic model for rules enables mapping between provided JSON format and relational database for mining and extraction of rules previously added over a registration API. The crucial component that Trust Manager utilize to continuously maintain the updated version of score in respect to last attribute value changes is the SocIoTal Context Manager.

3.7.1. Trust Manager Rule registration

The API described in this section provides functions for trust and reputation management for the algorithm developed within SocIoTal WP2. It includes methods for compiling new rules for the algorithm.

Name	**registerRules**	
Description	Register NEW set of rules. *THIS should be the first operation that initiates the start of reputation computation*. Trust Manager is comparing two attribute values or attribute value against custom value and assigns 5 if condition is true and 1 if not. Final score is a mean of all conditions.	
SocIoTal Server	*<trust_manager_server>*	
Method	**POST**	
	/TrustManagerSociotal/rest/api/registerRules/	
Headers		
	Content-Type	application/json
Request Payload		
Payload Description: The payload includes a list of ***rules*** elements, each one with the following information: ● **appicationId**: unique ID of the application that starts the reputation computation ● **attribute**: an attribute with value that is going to be compared ● **operandKeyword:** keyword used for comparing two attribute, if one attribute value IS, GRATER, LESS or ISNOT equal to another attribute value (or custom provided		

value). Only **IS** is implemented right now

- **compareAttr**: value used for comparision. If this is not an attribute then its type should be "type":"String"

Payload Example:

```
[
  {
    "applicationId":"application_1",
    "attribute":{
      "deviceId":"SocIoTal:UNIS:Smartphone:Dev:002",
      "name":"DiscoveredDevice"
    },
    "operandKeyword":"IS",
    "compareAttribute":{
     "value":"ELSON-THINK",
      "type":"String"
    }
  },
  {
    "applicationId":"application_1",
    "attribute":{
      "deviceId":"SocIoTal:UNIS:Smartphone:Dev:002",
      "name":"SocialRelation"
    },

   "operandKeyword":"IS",
    "compareAttribute":{
      "value":"PRIVATE",
     "type":"String"
    }
  }
]
```

Status Codes
200 – Status Ok 401 - *"Unauthorized"*
Error Messages
401 - *"Unauthorized"* 400 - *Custom error messages for bad requests*

3.7.2. Reputation Query

Reputation score can be queried by the application_Id used during rule registration process. Query context is described in Section 2.5. Example of the final payload that application gets after queryContext is given below.

Name	queryContext (by Context Entity ID)	
Description	this method (not NGSI native) will retrieve the whole current context of the Entity ID passed as a http parameter	
SocIoTal Server	<sociotal_server>:<port>	
Method	GET	
	/SocIoTal_CM_REST_V2/NGSI10_API/queryContext/<APPLICATION_ID>	
Headers		
	Content-type	application/json
	Accept	application/json
Status Codes:		
<All the HTTP status codes returned by API>		
Response:		

```
{
   "contextElements": [{
      "id": "applicationID",
      "type": "SocIoTal_Resource:Reputation",
      "isPattern": "false",
      "attributes": [{
         "name": "Reputation",
         "value": "4",
         "type": "float"
      }]
   }],
   "updateAction": "APPEND"
}
```

Error Messages	
Section 2.5 queryContext <messages>	

3.8. Face-to-face interactions

The following subsections provide the specification for the REST communication with SocIoTal Context Manager (Section 2) in order to query or update information related to F2F interactions.

3.9. Query F2F Interaction Information

Retrieve a user's F2F interactions and additional contextual information such as social relation. Each retrieved entity consists of one attribute:

- **Name**: "F2FInteration". It specifies if a face-to-face interaction is detected or not.
- **Value**: "false". The value describes if a face-to-face interaction occurred or not. The non-face-to-face interactions are logged in order to detect also the social relations with people-devices that do not belong to SocIoTal platform. In that situation, only their interpersonal distance and social relation is considered.

- **Metadatas**. Included some additional information about the detected face-to-face interaction. In particular, it logs the discovered device with which it perform a face-to-face interaction, the social relation based on their interpersonal distance, the timestamp when it occurred and the location

3.9.1. Query F2F Interaction Information by ID

Query F2F Interaction Information for a specific entity ID (i.e. device)

F2F Interaction Information can be queried for **a specific entity ID** (i.e. device) using the API described in2.5. For Example:

Call	**<Sociotal_server>:<port>**//SocIoTal_CM_REST_V2/NGSI10_API/queryContext/**SocIoTal:UNIS:SmartphoneContext:VirtualSmartphoneContext_002**

Response JSON:

```
{
    "errorCode": {
        "details": "Count: 1",
        "reasonPhrase": "OK",
        "code": "200"
    },
    "contextResponses": [
        {
            "statusCode": {
                "reasonPhrase": "OK",
                "code": "200"
            },
            "contextElement": {
                "id":
"SocIoTal:UNIS:SmartphoneContext:VirtualSmartphoneContext_002",
                "attributes": [
                    {
                        "name": "F2FInteraction",
                        "value": "false",
                        "type": "boolean",
                        "metadatas": [
                            {
                                "name": "DiscoveredDevice",
                                "value": "Nick?s MacBook Air",
                                "type":
"http://sensorml.com/ont/swe/property/pseudonym"
                            },
                            {
                                "name": "SocialRelation",
                                "value": "PUBLIC",
                                "type": "string"
                            },
                            {
                                "name": "DateTimeStamp",
                                "value": "20150317T134409Z",
                                "type":
"http://sensorml.com/ont/swe/property/DateTimeStamp"
                            },
                            {
                                "name": "Location",
                                "value": "-0.58823666, 51.24346692",
                                "type":
"http://sensorml.com/ont/swe/property/Location"
```

```
                                }
                            ]
                        }
                    ],
                    "type": "urn:x-org:sociotal:resource:device",
"isPattern": "false"
            }
        }
    ]
}
```

3.9.2. Query F2F Interaction Information by payload

Regular expression. The API provides the ability to the developer to perform a query using a regular expression in the name of the device. The attribute "isPattern" should be set to true. For Example:

SocIoTal SERVER	<Sociotal_server>:<port>	
POST	/ SocIoTal_CM_REST_V2/NGSI10_API /queryContext	
HEADERS		
	Content-type	application/json
	Accept	application/json
Payload:		

```
{"entities":[
    {"type":"",
     "isPattern":"true",
 "id":"SocIoTal:UNIS:SmartphoneContext:VirtualSmartphoneContext*"
    }
  ]
}
```

The response, in JSON format, is the query F2F Interaction Information by payload response described in 2.5

3.9.3. Query F2F Interaction Information by attributes

The API provides also the ability to the developer to specify the particular attributes to retrieve, instead of acquiring the complete entity.

SocIoTal SERVER	<Sociotal_server>:<port>	
POST	/ SocIoTal_CM_REST_V2/NGSI10_API /queryContext?limit=1	
HEADERS		
	Content-type	application/json

	Accept	application/json

Payload:

```
{"entities":[
    {"type":"",
     "isPattern":"false",
 "id":"SocIoTal:UNIS:SmartphoneContext:VirtualSmartphoneContext_002"
    }
  ],
"attributes":["F2FInteraction"]
}
```

The response, in JSON format, is the query F2F Interaction Information by attributes response described in 2.5

3.10. Update F2F Interaction Information

Update a user's F2F interactions and additional contextual information such as social relation. In particular, updating a user's F2F interactions consists of three main functionalities:

- ***APPEND***. Creates a F2F Interaction entity with the given information as attributes, if the entity for a particular device does not exist. If the F2Finteraction entity exists for a particular device, then it appends the given attribute(s).
- ***UPDATE***. Updates a F2F Interaction entity. Given the arguments, it updates - overrides the attributes of a particular F2F interaction entity. The update operation requires the entity that will be updated, already exists.
- ***DELETE***. Deletes a F2F Interaction entity based on the attributes provided as arguments. The delete operation requires the entity that will be deleted, already exists.

3.10.1. Update a F2F Interaction entity: "APPEND"

This action should be called in order to create an entity in the SocIoTal Context Manager or to append attributes at an existing entity. If the entity does not exist, the SocIoTal Context Manager will create a new entity described by the information specified in the payload of the post operation. If the entity exists, the SocIoTal Context Manager will append the existing entity with the specified attributes of the payload.
Please refer to the paragraph 2.5 of this document to read how to create a F2F Interaction entity.

3.10.2. Update a F2F Interaction entity: "UPDATE"

This action should be called in order to update an existing entity in the SocIoTal Context Manager. The developer should specify the information that should be updated in the payload, and in particular in the attributes. Any previous information will be lost and be replaced by newly specified information.

Please refer to the paragraph 2.5 of this document to read how to update a F2F Interaction entity.

3.10.3. Deletes a F2F Interaction entity: "DELETE"

This action should be called in order to delete an existing entity in the SocIoTal Context Manager. The developer should specify the entity ID to delete and additional attributes.
Please refer to the paragraph 2.5 of this document to read how to delete a F2F Interaction entity.

Section 4. Sociotal User and Developer Environment API

The SocIoTal User and Developer environments are aimed to lower the barrier of IoT adoption for citizens.

UserEnv is designed as an API broker that consumes data from the Sociotal Context Manager and services that are actually storing the data collected from the devices.
This approach assumes that physical devices connect to Sociotal Context Manager that virtualize these, making them accessible via API calls. Full specification are described in D4.1 [4].

This design allows paving the ground for the high level features of the mobile and web user interface tools, without facing devices low-level details.

In a similar way, the DevelEnv is assumed to be decoupled as much as possible from external entities, providing connectors to IoT platforms as a way to interact with SocIoTal, 3rd party components and platforms are not detailed in this document.

In UserEnv API, every endpoint is authenticated by token through an *access_token* URL parameter.
That token is unique and it is associated to a particular user and can be obtained from the user profile page inside the User Environment website. That token is represented by the AUTH_TOKEN placeholder in the API documentation that follows.

The supported format is only JSON, for every endpoint.

4.1. Channel Management API

A channel **is an** API connector that allows interfacing the UserEnv to a SocIoTal API (eg. A SocIoTal Context Manager) or to a 3rd party device API (eg. Xively), or a social/online service (eg. Facebook). Each category of device or services to be managed in the UserEnv will have a corresponding Channel. [D4.2][3]

This section describes how to manage a channel in the sociotal web environment.

Name	**createChannel**
Description	Create a new Channel inside the User Environment. This channel is created following a set of templates. One of this template is the Sociotal Channel.
URL parameters	

[3] Sociotal D4.2 - Alpha release of intuitive user and developer environment

	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page.

SocIoTal Server	http://sociotal.crs4.it/api	
Method	**POST**	
	/channels? access_token= AUTH_TOKEN	
Headers		
	Content-Type	application/json
Request Payload		

Payload description:

- "channel_type" : STRING. The type of the channel
- "title": STRING. Title
- "context": STRING. Name of the context
- "unit": STRING. Unit of measure of the sensor
- "description": STRING. Channel description
- "tags": [STRING] comma separated tags

Payload example:

```
{
 "channel_type": "GenericDeviceChannel",
 "title": "MyTemperature",
 "context": "bed room",
 "unit": "centigrade",
 "description": "this is my temperature sensor in my bed room",
 "tags": "bed, room, sens"
}
```

Status Codes

201 Created in case of success
401 in case of missing or not valid token

Error Messages

An error string.
For Example:
"Unauthorized" for 401

Name	**updateChannel**
Description	Update the channel of a user inside the User Environment.
URL parameters	
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it

	is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user	
SocIoTal Server	http://sociotal.crs4.it/api	
Method	**PUT**	
	/channels/:CHANNEL_ID?access_token= AUTH_TOKEN	
Headers		
	Content-Type	application/json
Request Payload		

Payload description:

- "channel_type" : STRING. The type of the channel
- "title": STRING. Title
- "context": STRING. Name of the context
- "unit": STRING. Unit of measure of the sensor
- "description": STRING. Channel description
- "tags": [STRING] comma separated tags

Payload example:

```
{
 "channel_type": "GenericDeviceChannel",
 "title": "MyTemperature",
 "context": "bed room",
 "unit": "centigrade",
 "description": "this is my temperature sensor in my bed room",
 "tags": "bed, room, sens"
}
```

Status Codes
200 OK in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param
Error Messages
An error string. For Example: *“Unauthorized”*

Name	**deleteChannel**
Description	Delete the channel of a user inside the User Environment.
URL parameters	
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it

	is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user	
SocIoTal Server	http://sociotal.crs4.it/api	
Method	**DELETE**	
	/channels/:CHANNEL_ID?access_token=AUTH_TOKEN	
Headers		
	Content-Type	application/json
Status Codes		
200 OK in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param		
Error Messages		
An error string. For Example: *"Unauthorized"*		

Name	**getChannelList**	
Description	Returns the list of the channel owned by the user.	
SocIoTal Server	http://sociotal.crs4.it	
Method	**GET**	
	/channels/?access_token=AUTH_TOKEN	
URL parameters		
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page.	
Headers		
	Content-Type	application/json
Response Body		

Response example:

```
{
    "channels": [
        {
            "_id": "54e4bd424f38c75d3047fe46",
            "_type": "GenericDeviceChannel",
            "user": {
                      "_id": "543e77c7814100d1110479e3"
            },
            "context": "12",
            "unit": "centigrade",
            "__v": 0,
            "connections": [],
            "interval_sec": 60,
            "createdAt": "2015-02-18T16:26:42.593Z",
            "image": {
                "files": []
            },
            "isRunning": false,
            "outbox": [],
            "inbox": [],
            "tags": [
                "bed",
                " room",
                " sens"
            ],
            "description": "this is my temperature sensor in my bed room",
            "channel_type": "GenericDeviceChannel",
            "title": "MyTemperature"
        }
}
```

Status Codes
200 OK in case of success 401 in case of missing or not valid token
Error Messages
An error string. For Example: *"Unauthorized"*

Name	**getChannelInformation**	
Description	Returns the channel information in json format.	
SocIoTal Server	http://sociotal.crs4.it/api	
Method	**GET**	
	/channels/:CHANNEL_ID?access_token= AUTH_TOKEN	
URL parameters		
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user	
Headers		
	Content-Type	application/json
Response Body		

Response example:

```
{
  "channel": {
    "_id": "55b9e3f53debaf1c313140ec",
    "_type": "GenericDeviceChannel",
    "unit": "centigrade",
    "__v": 1,
    "context": "bed BOH ",
    "connections": [],
    "interval_sec": 60,
    "createdAt": "2015-07-30T08:44:37.720Z",
    "image": {
      "files": []
    },
    "isRunning": false,
    "outbox": [],
    "inbox": [],
    "tags": [
      "bed",
      " room",
      " sens",
      " temperature"
    ],
    "description": "this is my temperature sensor in my bed room",
    "channel_type": "",
    "title": "MyTemperature"
  }
}
```

Status Codes
200 OK in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param
Error Messages
An error string. For Example: *"Unauthorized"*

4.2. Process Modelling: Composition (Trigger/Action) API

These APIs allow a user to create compositions between triggers and actions of different Channels. [D4.2] [4]
A **trigger**, registered in a business logic composition, is a condition to be evaluated on the incoming data on the source Virtual Entity. An **action** is the set of operations to be executed on the destination Virtual Entity when the trigger is valid.

First three methods aim to handle the management of **Composition** objects. A Composition object is composed by a trigger for the source VE and an action for a destination VE.
Data is a document that contains the information about the creation of a composition, in particular: ve_id of the destination (where the action will be executed) and the trigger logic to be executed.
The method **getCompositionList** returns all the compositions for a specific virtual entity and **getComposition(composition_id): composition** returns information about a single composition.

[4] Sociotal D4.2 - Alpha release of intuitive user and developer environment

Name	addComposition
Description	To connect a source Channel that produces data, to a target channel able to receive and read the data. This data is the result of a triggered composition.
SocIoTal Server	http://sociotal.crs4.it/api
Method	POST
	/channels/:CHANNEL_ID/compositions?access_token= AUTH_TOKEN
URL parameters	
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user
Headers	
	Content-Type — application/json

Request Payload

Payload description:

- "trigger " : JSON.
  - o Attribute: the attribute to trigger
  - o Name: name of the compare operation
  - o Arg: the argument for the comparizon
- "action": JSON:
  - o targetChannelID: STRING. Id of the target channel
  - o "actionName": STRING. Name of the action
- “label”: the message to send to the trigger channel

Payload example:

```
{
"trigger": { "attribute": "Speed", "name": "greaterThan", "check": "56" },
 "action":
  { "targetChannelId": "54d38e928ae5cd360fc3ec23",
    "actionName": "consoleAction",
    "arg": "you are driving nicely!!!" },
 "label": "WHEN Speed greater than 56 DO mydebug.consoleAction('you are driving
nicely!!!')"
}
```

Response Example

```
{
   "compositions": [
       {
           "label": "WHEN Speed greater than 56 DO mydebug.consoleAction('you are
driving nicely!!!')",
           "_id": "54d8832245d0ed612953fcda",
           "active": false,
           "action": {
               "targetChannelId": "54d38e928ae5cd360fc3ec23",
               "actionName": "consoleAction",
               "arg": "you are driving nicely!!!"
           },
           "trigger": {  "attribute": "Speed",  "name": "greaterThan",  "check":
"56",  "negation": false}
       },
       {
           "label": "WHEN Fuel lower than 10 DO mydebug.consoleAction('WARNING:
search a gas station!!!')",
           "_id": "54d8b52805bd390136359e3c",
           "active": false,
           "action": {
               "targetChannelId": "54d38e928ae5cd360fc3ec23",
               "actionName": "consoleAction",
               "arg": "WARNING: search a gas station!!!"
           },
           "trigger": { "attribute": "Fuel",  "name": "lowerThan",  "check": "10",
"negation": false}
       }
   ]
}
```

Status Codes
201 Created in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param
Error Messages
An error string. For Example: *"Unauthorized"*

Name	**getCompositionList**
Description	Returns all the channel compositions of the user.
SocIoTal Server	http://sociotal.crs4.it/api
Method	**GET**
	/channels/:CHANNEL_ID/compositions?access_token= AUTH_TOKEN
URL parameters	
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. CHANNEL_ID: is the unique id for the channel of the user
Headers	

	Content-Type	application/json

Response Example

```
{
   "compositions": [
       {
           "label": "WHEN Speed greater than 56 DO mydebug.consoleAction('you are
driving nicely!!!')",
           "_id": "54d8832245d0ed612953fcda",
           "active": false,
           "action": {
               "targetChannelId": "54d38e928ae5cd360fc3ec23",
               "actionName": "consoleAction",
               "arg": "you are driving nicely!!!"
           },
           "trigger": {
               "attribute": "Speed",
               "name": "greaterThan",
               "check": "56",
               "negation": false
           }
       },
       {
           "label": "WHEN Fuel lower than 10 DO mydebug.consoleAction('WARNING:
search a gas station!!!')",
           "_id": "54d895f705bd390136359e3a",
           "active": false,
           "action": {
               "targetChannelId": "54d38e928ae5cd360fc3ec23",
               "actionName": "consoleAction",
               "arg": "WARNING: search a gas station!!!"
           },
           "trigger": {
               "attribute": "Fuel",
               "name": "lowerThan",
               "check": "10",
               "negation": false
           }
       }
   ]
}
```

Status Codes

200 OK in case of success
401 in case of missing or not valid token
404 if no Channel corresponds to the specified CHANNEL_ID param

Error Messages

An error string. For Example:
"Unauthorized"

Name	**getComposition**
Description	Get the composition representation
SocIoTal Server	http://sociotal.crs4.it/api
Method	**GET**

	/channels/:CHANNEL_ID/compositions/:COMPOSITION_ID?access_token=AUTH_TOKEN	
URL parameters		
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user. **COMPOSITION_ID:** is the unique id for the composition	
Headers		
	Content-Type	application/json

Response Example

```
{ "composition": {
           "label": "WHEN Fuel lower than 10 DO mydebug.consoleAction('WARNING:
search a gas station!!!')",
           "_id": "54d895f705bd390136359e3a",
           "active": false,
           "action": {
               "targetChannelId": "54d38e928ae5cd360fc3ec23",
               "actionName": "consoleAction",
               "arg": "WARNING: search a gas station!!!"
           },
           "trigger": {
               "attribute": "Fuel",
               "name": "lowerThan",
               "check": "10",
               "negation": false
           }
       }
   ]
}
```

Status Codes

200 in case of success
401 in case of missing or not valid token
404 if no Channel corresponds to the specified CHANNEL_ID param or no Composition exists for specified COMPOSITION_ID param

Error Messages

An error string. For Example:
“Unauthorized”

Name	**updateComposition**	
Description	Update a Composition	
SocIoTal Server	http://sociotal.crs4.it/api	
Method	**PUT**	
	/channels/:CHANNEL_ID/compositions/:COMPOSITION_ID?access_token=AUTH_TOKEN	
URL parameters		
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user. **COMPOSITION_ID:** is the unique id for the composition .	
Headers		
	Content-Type	application/json

Request Payload

Payload description:

- "trigger " : JSON.
  - o Attribute: the attribute to trigger
  - o Name: name of the compare operation
  - o Arg: the argument for the comparizon
- "action": JSON:
  - o targetChannelID: STRING. Id of the target channel
  - o "actionName": STRING. Name of the action
- “label”: the message to send to the trigger channel

Payload example:

```
{
"trigger": { "attribute": "Speed", "name": "greaterThan", "check": "56" },
 "action":
  { "targetChannelId": "54d38e928ae5cd360fc3ec23",
    "actionName": "consoleAction",
    "arg": "you are driving nicely!!!" },
 "label": "WHEN Speed greater than 56 DO mydebug.consoleAction('you are driving
nicely!!!')"
}
```

Response Example

```
{
  "composition": [
    {
      "label": "WHEN AmbientTemperature greater than 24 DO
TestDebug.consoleAction(>24)",
      "_id": "55b77a6296c543151c427078",
      "active": false,
      "action": {
        "targetChannelId": "55b77a4f96c543151c42704d",
        "actionName": "consoleAction",
        "arg": ">24"
      },
      "trigger": {
        "attribute": "AmbientTemperature",
        "name": "greaterThan",
        "check": "24",
        "negation": false
      }
    }
  ]
}
```

Status Codes
200 OK in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param or no Composition exists for specified COMPOSITION_ID param
Error Messages
An error string. For Example: *"Unauthorized"* { "composition": "Not found!!!"}

Name	**deleteComposition**	
Description	Delete a Channel composition.	
SocIoTal Server	http://sociotal.crs4.it/api	
Method	**DELETE**	
	/channels/:CHANNEL_ID/compositions/:COMPOSITION_ID?access_token=AUTH_TOKEN	
URL parameters		
	AUTH_TOKEN: is an unique token associated to a UserEnv user and it is contained in the user profile web page. **CHANNEL_ID:** is the unique id for the channel of the user. **COMPOSITION_ID:** is the unique id for the composition .	
Headers		
	Content-Type	application/json
Status Codes		
200 OK in case of success 401 in case of missing or not valid token 404 if no Channel corresponds to the specified CHANNEL_ID param or no Composition exists for specified COMPOSITION_ID param		

Error Messages
An error string. For Example: *"Unauthorized"*   { "composition": "Not found!!!"}

4.3. Developer Environment API

The interaction between the Studio and the gateway relies on the HTTP protocol, using REST (REpresentational State Transfer) architectural style. The communication is always initiated by the studio, which can query the gateway to obtain the available devices, services, resources and associated values. The studio can also register a callback, to be notified by the gateway as soon as a resource value evolves.

Name	getDevicesList	
Description	Gets the list of available devices	
SocIoTal Server	<uri:port>/sensinact/devices	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body		

Request example
<uri:port>/sensinact/devices

Associated Response
```
[
{"ID":"SensiNact_Gateway"},
{"ID":"AppManager"}
]
```

Name	getDeviceDescription	
Description	Describes a device	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway
 	
Associated Response
`{`
`    "status": "active",`
`    "location": "null",`
`    "ID": "SensiNact_Gateway"`
`}`

Name	getServicesList	
Description	Gets the list of available services, for a given device with id {device_ID}	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway/services
 	
Associated Response
`[`
`{"ID":"Date"},`
`{"ID":"System"},`
`{"ID":"AdminService_SensiNact_Gateway"}`
`]`		

Name	getServiceDescription	
Description	Describes a service	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway/services/System
 	
Associated Response
`{`
`    "ID": "System",`
`    "type": "smart-object-service"`
`}`		

Name	**getResourcesList**	
Description	Gets the list of available resources, for a given device with id {device_ID} and a given service with id {Service_ID}	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body		

Request example

<uri:port>/sensinact/devices/SensiNact_Gateway/services/System/resources

Associated Response

```
[
{"name":"last_event"},
{"name":"location"}
]
```

Name	**getResourceDescription**	
Description	Describes a resource	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources/{Resource_ID}	
Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example

<uri:port>/sensinact/devices/SensiNact_Gateway/services/System/resources/last_event

Associated Response

```
{
    "accessMethods": [
        {
            "parameters": [
                {
                    "name": "subscriberListener",
                    "type":
"fr.cea.sensinact.gateway.device.api.listener.DataResourceListener"
                }
            ],
            "type": "SUBSCRIBE"
        },
        {
            "parameters": [
                {
                    "name": "subscriberListener",
                    "type":
"fr.cea.sensinact.gateway.device.api.listener.DataResourceListener"
                },
                {
                    "name": "condition",
                    "type": "fr.cea.sensinact.gateway.device.api.Condition"
                },
                {
                    "name": "lifeTime",
                    "type": "java.lang.Long"
                }
            ],
            "type": "SUBSCRIBE"
        },
        {
            "parameters": [
                {
                    "name": "subscriptionID",
                    "type": "java.lang.String"
                }
            ],
            "type": "UNSUBSCRIBE"
        },
        {
            "parameters": [],
            "type": "GET"
        }
    ],
    "name": "last_event",
    "attributes": [],
    "type": "SensorData"
}
```

Name	**getResource**
Description	Performs a GET action on a resource
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources/{Resource_ID}/GET

Method	**GET**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway/services/System/resources/last_event/GET

Associated Response

```
{
    "name": "last_event",
    "value": {
        "event": 1,
        "device": "AppManager",
        "type": "fr.cea.sensinact.gateway.device.api.Device"
    },
    "type": "org.json.JSONObject",
    "metadata": []
}
```

Name	**subscribeResource**	
Description	Performs a SUBSCRIBE action on a resource	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources/{Resource_ID}/SUBSCRIBE	
Method	**POST**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway/services/System/resources/last_event/SUBSCRIBE

Associated Request Body

```
{"callback":"http://132.168.88.214:8081"}
```

Associated Response

```
{"message":"18854081501437569503459"}
```

Name	**unsubscribeResource**
Description	Performs an UNSUBSCRIBE action on a resource
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources/{Resource_ID}/UNSUBSCRIBE
Method	**POST**

Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/SensiNact_Gateway/services/System/resources/last_event/UNSUBSCRIBE

Associated Request Body
```
{"usid":"18854081501437569503459"}
```

Associated Response
```
{"message":"Unsubscribe done"}
```

Name	**actResource**	
Description	Performs an ACT action on a resource. This resource must be an action	
SocIoTal Server	<uri:port>/sensinact/devices/{device_ID}/services/{Service_ID}/resources/{Resource_ID}/ACT	
Method	**POST**	
Headers		
	Content-type	application/json
	Accept	application/json
Response Body Example		

Request example
<uri:port>/sensinact/devices/Lightning_system/services/Light_002/resources/DIM/ACT

Associated Request Body
```
[
    {
        "name": "BRIGHTNESS",
        "value": 75,
        "type": "java.lang.Double"
    }
]
```

Associated Response
```
{
    "message": "Dim action executed",
    "status": "SUCCESS",
    "modified resources": [
        "state"
    ]
}
```

Section 5. API evaluation and feedback

Following some best practices, like the API Usability Testing maintained by the US Government 5, it has been defined a simple workflow for all the SocIoTal API Evaluation meetups.

In order to receive feedback from developers, some SocIoTal partners started to test this evaluation procedure internally, involving their collegues: CRS4 with a session held in Pula (Cagliari), DNET in Novi Sad and all the partners involved in Lisbon for the IoT Week 2015. The main objective was to collect a first feedback on the whole procedure and an evaluation of the API about their functionalities, the privacy factor and the developer experience.

The first clear feedback came from the IoT Week in Lisbon, where colleagues from City Pulse, SMARTIE and FIWARE Association evaluated the API following our evaluation workflow. Details about the developer evaluation sessions can be found in D6.3 [5].

The evaluation workflow includes a methodology to follow in order to drive the evaluation sessions with target developer. Moreover, it defines a card schema targeted to supervisors to collect feedback from developers. Next paragraphs show in detail the overall evaluation workflow and how it was used in SocIoTal. Finally, this section reports a summary of the results of the first evaluation sessions done.

5.1. Assumptions and Current Status

Assumption 1) The entry point of the whole evaluation workflow is represented by the API Documentation Hub, The API Documentation Hub could be in an electronic format accessible via web (i.e. a web site or a wiki page where the API are described) or a simple collection of paper documents delivered by hands to developers.

The hub is the central resource for developers in order to:

- Access to the API documentation and to an overview of the SocIoTal Architecture;
- discover and access documentation for every single API endpoint;

Ideally, the hub should be an exhaustive and well-designed website by which developers start the API evaluation.

To date, given the continuous evolution of the status of the SocIoTal development, the hub is exceptionally represented by an excerpt of the API Specification document and it will be replaced as soon as possible as development goes ahead and produces a dedicated API documentation website, which is an ongoing work and available at URL: https://github.com/sociotal/SOCIOTAL/wiki

5 http://18f.github.io/API-Usability-Testing

Assumption 2) Each evaluation session should be composed by 3 to a maximum of 5 participant developers. The number of SocIoTal supervisors is not limited but should be fair. Evaluation environment should be friendly, not crowded, noiseless and evaluation conducted in an informal way.

5.2. The Workflow

1. Select Developers
2. Let Developers get access to the API Documentation Hub
3. Developers navigate through general documentation and specific APIs sections; they MUST be not guided unless they ask, and they talk out loud about evaluation listing what is good and what is bad or they communicate about their expectations and issues; SocIoTal Supervisors collect the whole feedback: taking notes or eventually recording audio notes. The final goal is to create a collection of Evaluation Cards (see paragraph 5.3 of this Section)
4. (related to API maturity and stage of development) Developers are invited to try the APIs using the tool/language they prefer (e.g., curl, JavaScript, Python, …). SocIoTal Supervisors collect the feedback
5. SocIoTal Supervisors summarize and fill the Evaluation Cards
6. Workflow should be iterated after APIs improvement.

The following picture shows the described workflow.

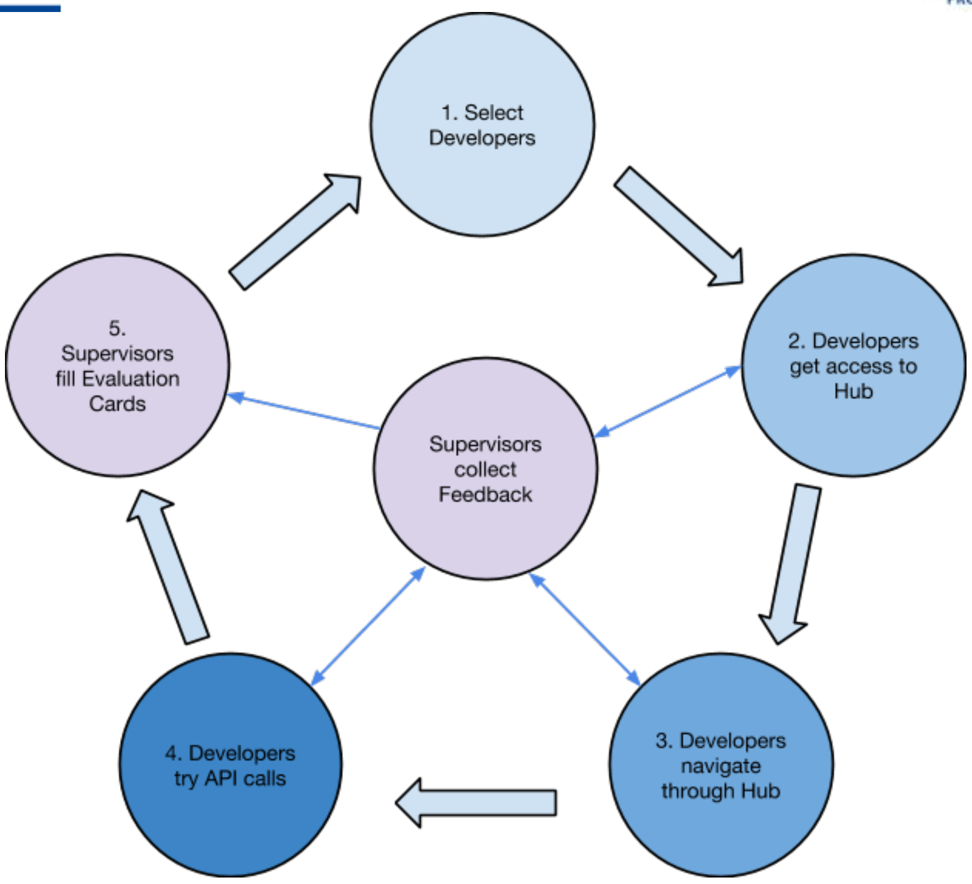
1. Select Developers
2. Developers get access to Hub
3. Developers navigate through Hub
4. Developers try API calls
5. Supervisors fill Evaluation Cards
Supervisors collect Feedback

Figure 7. SocIoTal API Evaluation Workflow

5.3. The Evaluation Card

The Evaluation Cards are defined and used by sessions supervisors in order to collect and summarize the feedback given by developers about the APIs.

The Card template is defined as in the following model:

Date	
API Group	The SocIoTal API macro-group under evaluation
API	Specific SocIoTal API (i.e., endoint) under evaluation

Good	A description of what's defined as good about the API
Bad	A description of what's defined as bad about the API
Missing	A set of missing features in the API
What would you do differently?	A collection of developers' advices about improving the API
What adjunctive API would you like to have?	A list of suggested new APIs or new features
What adjunctive resources/tools/documentation would you like to have?	A list of further tolos developers' remarked as better to have and provide
NOTES	Free notes about the overall evaluation

5.4. Evaluation results and summary

This evaluation results and summary are related to three evaluation sessions held in Pula (CA) at CRS4, in Novi Sad at DNET and in Lisbon for the IoT Week 2015.

The general feedback resulted from the sessions has been good. There are also some gaps that should be managed in future to have a more developer friendly APIs and documentation. First of all, the developers want to have a clear idea of what is SocIoTal and all its aspects before use and evaluate the APIs.
Some developers said that the session was too short (about one hour of average time per session), so that more time for our explanations and their testing would be welcome.

An introduction scenario could help a developer to use and evaluate better the APIs. This scenario can also introduce what exactly is expected for the session and all the methods and tools that will be used (e.g., RESTEasy, POSTMAN etc)
For example, for the next sessions a suitable scenario that tries to cover all the SocIoTal's APIs could include these steps:

1) Create an identity;
2) Create/Obtain a valid token;
3) Register an entity;
4) Update values
5) Check security;
6) Discover available entities;

7) Manage the trust; etc.

Then, once guided the developers through this guided stage, they will be able to play with all available methods.

Developers also leave some comments about the API documentation:

- They would appreciate an electronic format of the API documentation
- Could be better a friendly way of the method description
- To explain methods main goals and functionalities
- Endpoint URLs are hard to memorize and should be uniformed: too long, ugly path names
- For each endpoint documentation, having a summary of all supported operations could increase readability
- Sometimes real API responses are different from the responses described in documentation

Specific comments about each API group can be summarized as follows:

- **Authorization Client**
    - Java library available can be easily integrated
    - Improve Authentication process and describe how to get certificates
- **Sociotal Context Manager API (NGSI 9 and NGSI10):**
    - Documentation is good and easy to use
    - Developers with a really background in context broker find methods and tutorial quite complete. Those with no expertise need to be assisted
    - Bad usage of POST, in same cases PUT is the way
    - A JSON with errorMessage also for 200 OK operations
    - ID is not a real/unique ID
    - Possible issue with Update/append action for registering context and updating a context. It is not clear what will be updated and what appended if the same ID is used and partially different attributes
    - Methods for NGSI10 example contains invalid JSON payload
    - Error handling. If user sends wrong request to the endpoint that does not accept that request html is returned instead of json response (error not handled properly).
    - Subscription when value is >, < or = something
    - When the answer is OK from the webservice, body can be misleading because of the “errorcode” part that is always returned.
    - When deleting context why payload is required when this can be done only using ID?

- **User Env:**
    - Easy to use once you know what you are doing
    - Access token in the headers
    - To add some client libraries
    - Check date formats
    - To update one parameter and not all the data
    - Less user data details on reponses
    - Add metadata for pagination
    - Payload description on POST, types, etc...

- **Trust Manager:**
    - What is been compared in JSON rule example. It is not clear
    - Provide more user-friendly example of rules with explanation possibly real-world example
    - Reputation score in queryContext to be marked red
    - Explain what is returned by the TrustManager during rule registration at the first time? It would be better to put JSON response with Ok message and not the current value of the reputation score
    - Explain better what is returned by the TM and what by the context broker
    - Mention that application_id used in queryContext to obtain reputation score is one used during registration
    - Register rules: Error handling. When JSON payload is not good return message is not handled properly

The overall feedback was very good and, as a result of this, developers did a lot of comments and suggestions that will help to complete the SocIoTal APIs for Month 36.

Section 6. Conclusion

This document reports the updates in specifications of the Application Programming Interface (API) of the SocIoTal platform and it addresses the objective **O1.4: To specify required application programming interfaces (API)** related to task *T1.3: Open APIs for service development and IoT device integration (M7- M36)*, that will enable development of services on top of the SocIoTal architecture, having in mind that the targeted users are public at large.

The API updates follows the best practices in specification of such interfaces. A REST style API is the envisioned approach (but not limited to) to be used, when suitable and convenient.

This deliverable, through its Section 1, proposes a semi-formal notation for the API specification description, which is carried out in Sections 2, 3 and 4, grouped by envisioned modules of the SocIoTal architecture and those provided by a selected base platform as prerequisite for SocIoTal.
Each API description includes the supported functionalities list (i.e., CRUD operations in case of a REST-based API), data types and their representation as well as returned data and parameters; moreover, they include the scope of the API, i.e.: internal or public, and the provider as the architecture module, WP or external tool/framework which expose the particular API.
The deliverable highlights two main aspects worth noting:

- The basic functionalities desired from a platform that must be the core module of the SocIoTal architecture and providing the lower-level infrastructure of an IoT platform and framework.
- The functionalities exposed by SocIoTal to support people who want to develop new applications based on it.

The deliverable also defines a model for the API evaluation process. It is described as a workflow to adopt and follow during API evaluation sessions with developers. The workflow includes the definition of the required steps and of the evaluation card model, filled by sessions' supervisors and designed to collect developers' feedback.
A summary of the results of the first evaluation sessions done is included in Section 5.

Advances in API specification drove to a better design of SocIoTal modules, resulting in an improved architecture and integration of the tools.

Section 7. References

[1] D3.2.1 – Privacy-aware context-sensing device discovery
[2] D2.2 – Framework Specification for Privacy and Access Control
[3] Publish/Subscribe Context Broker – Orion Context Broker, http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker
[4] D4.2 - Feature specification of intuitive user and developer environment
[5] D6.3 - Y2 Report on first year community interactions and detailed dissemination strategy

Section 8. Glossary

API: Application Programming Interface
CRUD: Set of operations for a resource, Create-Read-Update-Delete.
F2F: Face-to-face, referred to personal interactions that occur in proximity
HTTP: Hypertext Transfer Protocol
REST: Representational State Transfer
IoT: Internet of Things
IoT-A: an architectural reference model for IoT achieved by the IOT-A project
URL: Uniform Resource Locator
UserEnv: The SocIoTal End User Environment
VE: see Virtual Entity
Virtual Entity: Computational or data element representing a Physical Entity in IOT-a model
WS: Web Service
WP: Work Package
XACML: eXtensible Access Control Markup Language
XML: eXtensible Markup Language
JSON: JavaScript Object Notation