IPSO Alliance
Interop Committee

Z. Shelby
Sensinode
C. Chauvenet
Watteco
August 24, 2012

The IPSO Application Framework
draft-ipso-app-framework-04

Abstract

This document defines a RESTful design for use in IP smart object systems such as Home Automation, Building Automation and other M2M applications. This design defines sets of REST interfaces that may be used by a smart object to represent its available resources, interact with other smart objects and backend services. This framework is designed to be complementary to existing Web profiles including SEP2 and oBIX. The document is not and may never be a standard of any kind. It should be considered only as work in progress.

Table of Contents

1. Overview

The IPSO Application Framework makes use of IETF standards as building blocks for a simple and efficient RESTful design model for IP smart objects. The framework may be used over either HTTP [RFC2616] or CoAP [I-D.ietf-core-coap] web transfer protocols. This section describes the overall design principles of the framework.

HTTP, REST, XML, JSON, COAP and other key components of web technology are powerful mechanisms in an Internet of Things application. However, they leave a lot of room for the application implementers to choose particular ways to represent data and operations on it.

This document specifies a particular template for using these mechanisms to represent certain classes of typical Internet of Things applications. The document is not and may never be a standard of any kind. It should be considered only as work in progress. The communication patterns described herein do not represent the only way to implement applications. Indeed, we expect applications to be developed with the full richness of the web communications model, including many standardized application models.

However, we have found these simple patterns helpful in a number of cases and they may make it easier to develop basic applications without having to re-create the communications patterns and data formats every time. The IPSO interoperability committee sees this document as helpful in testing practical interoperability among a number of specific devices, for instance.

1.1. Function Sets

The framework is organized into groups of resource types called Function Sets. A Function Set has a recommended root path, under which its sub-resources are organized. Each Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. A Function Set SHOULD be located at its recommended root path on a web server, however it MAY be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.).

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values.

1.1.1. Path Template

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root resource of a function set (e.g. for /sen/{#} the root is /sen) may optionally support the CoRE Batch (core#b) interface definition [I-D.shelby-core-interfaces]. If supported, this allows a single request on the parent to manipulate the values of the sub-resources at once (e.g. a GET on /sen would return a SenML payload with the values of all the sub-resources of /sen).

1.1.2. Resource Type

The Resource Type parameter defines the value that MUST be included in the rt= field of the CoRE Link Format when describing a link to this resource. This value enables resources to be discovered. Values in this field are in the form "namespace:type", where namespace references the specification where the type is defined. Currently supported namespaces are "ipso." refering to resource types defined in this specification, and "ucum:" used for generic sensor units from the Unified Code for Units of Measure (UCUM) specification [unitsofmeasure.org]. It is expected that other namespaces will be defined in the future.

1.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. This specification makes use of the basic CoRE resource types defined in [I-D.shelby-core-interfaces]. The Interface Description MAY be elided from link descriptions of resource types defined in the framework, but SHOULD be included for custom extensions to the framework.

1.1.4. Data Type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The actual format of this data is returned in either text/plain (MUST be supported) or application/senml+json (optional) formats as defined in Section 3.

1.2. Interaction Model

This framework is designed for a simple client-server interaction model on atomic resources, in order to minimize complexity. An IP smart object runs a simple web server (HTTP or CoAP) and exposes resources that conform to this framework. Other IP smart objects or

backend services that want to interact with that IP smart object, act as a client and make requests to interact with those resources. When using HTTP, polling is used to request values. When using CoAP both polling and the use of Observation is supported including the query string parameters defined in [I-D.shelby-core-interfaces]. The use of Observation is highly recommended when there is a need for continuous readings from a resource.

1.3. Discovery

The resource semantics defined in this framework are compatible with Web Linking [RFC5988] and the CoRE Link Format [I-D.ietf-core-link-format]. A device using to this framework SHOULD make those resources discoverable by providing links to the resources on the path /.well-known/core as defined in [I-D.ietf-core-link-format].

In addition, a device using this framework MAY in addition register its resources to a Resource Directory using the registration interface provided by a Resource Directory is defined in [I-D.shelby-core-resource-directory] if such a directory is available.

2. Function Sets

Function Set	Root Path	Resouce Type
Device	/dev	ipso.dev
General Purpose IO	/gpio	ipso.gpio
Power	/pwr	ipso.pwr
Load Control	/load	ipso.load
Sensors	/sen	ipso.sen
Light Control	/lt	ipso.lt
Message	/msg	ipso.msg
Location	/loc	ipso.loc
Configuration	/cfg	ipso.cfg

2.1. Device

This function set is used to represent information about the IP smart object device itself, including but not limited to its manufacturer, serial number and name. This function set MAY be extended with custom resource types.

Type	Path	RT	IF	Type	Unit
Manufacturer	/dev/mfg	ipso.dev.mfg	rp	s	
Model	/dev/mdl	ipso.dev.mdl	rp	s	
Hardware Revision	/dev/mdl/hw	ipso.dev.mdl.hw	rp	s	
Software Version	/dev/mdl/sw	ipso.dev.mdl.sw	rp	s	
Serial	/dev/ser	ipso.dev.ser	rp	s	
Name	/dev/n	ipso.dev.n	p,rp	s	
Power Supply	/dev/pwr/{#}	ipso.dev.pwr	rp	e	
Power Supply Voltage	/dev/pwr/{#}/v	ipso.dev.pwr.v	s	d	V
Time	/dev/time	ipso.dev.time	p, rp	i	s
Uptime	/dev/uptime	ipso.dev.uptime	s	i	s

Manufacturer: The manufacturer of the device as a string.

Model: The model of the device as a string.

Hardware Revision: The version of the hardware of the device as a string.

Software Version: The version of the software embedded in the device as a string.

Serial: The serial number of the device as a string.

Name: The descriptive or functional name of the device as a string.

Power Supply: The type of power supply as an enumeration Table 1.

Power Supply Voltage: The supply level of the device in Volts.

Time: POSIX time as the number of seconds that have elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970.

Uptime: The number of seconds that have elapsed since the device was turned on.

Enumeration	Name
0	Line
1	Battery
2	Harvester

Table 1: Power Supply Enumerations

2.2. General Purpose IO

This function set is used to represent general purpose IO related resources of a device, including press buttons, switches, digital inputs and outputs, and analogue inputs and outputs. The following table defines the kinds of sub-resource defined for the function set.

Type	Path	RT	IF	Type	Unit
Button	/gpio/btn/{#}	ipso.gpio.btn	s	i	
Digital Input	/gpio/din/{#}	ipso.gpio.din	s	b	
Digital Output	/gpio/dout/{#}	ipso.gpio.dout	a	b	
Analog Input	/gpio/ain/{#}	ipso.gpio.ain	s	d	V
Analog Output	/gpio/aout/{#}	ipso.gpio.aout	a	d	V
Dimmer Input	/gpio/dimin/{#}	ipso.gpio.dimin	s	i	0-100 %

Button: This resource type represents a button, and is modeled as an integer counter that is increased by one each time the button is pressed. When first initialized, this resource MUST be set to 0.

Digital Input: A digital input resource represents the boolean digital value (1,0) of a GPIO line set to input mode.

Digital Output: A digital output resource represents the boolean digital value (1,0) of a GPIO line set to output mode. A GET on this resource returns the current output value, and a PUT sets a new output value.

Analog Input: An analog input resource represents the decimal value in Volts of a GPIO line set to input mode.

Analog Output: An analog output resource represents the decimal value in Volts of a GPIO line set to output mode. A GET on this resource returns the current output value, and a PUT sets a new output value.

Dimmer Input: A variable digital input resource with possible values 0-100 as an integer in %.

2.3. Power

This function set is used to represent power measurement and control related resources, such as power meters, relays and loads. The following table defines the kinds of sub-resource defined for the function set. Not all instances of the power function set are required to support all sub-resources. For example a simple power measurement sensor might only support the Instantaneaous Power sub-resouce.

Type	Path	RT	IF	Type	Unit
Instantaneous Power	/pwr/{#}/w	ipso.pwr.w	s	d	W
Cumulative Power	/pwr/{#}/kwh	ipso.pwr.kwh	s	d	kWh
Load Relay	/pwr/{#}/rel	ipso.pwr.rel	a	b	
Load Dimmer	/pwr/{#}/dim	ipso.pwr.dim	a	i	0-100 %

Instantaneous Power: This resource type returns the instantaneous power of a load as a Decimal value in W.

Cumulative Power: This resource type returns the cumulative power of a load as a Decimal value in kWh. The value SHOULD be set to zero on initialization, however the value may be saved and retrieved from non-volatile memory.

Load Relay: This resource represents a relay attached to the load, which can be controlled, the setting of which is a Boolean value (1,0). A GET on the resource returns the current state of the relay, and a PUT on the resource sets a new state.

Load Dimmer: This resource represents a power controller attached to the load, which can be controlled as a % between 0-100. A GET on the resource returns the current state, and a PUT on the resource sets a new state.

2.4. Load Control

This function set is used for demand-response load control and other load control in automation application (not limited to power).

Type	Path	RT	IF	Type	Unit
Event Identifier	/load/id	ipso.load.id	p	s	
Start Time	/load/time	ipso.load.time	p	i	s
Duration In Min	/load/dur	ipso.load.dur	p	i	min
Criticality Level	/load/crt	ipso.load.crt	p	e	
Avg Load Adj Pct	/load/lap	ipso.load.lap	p	d	%
Duty Cycle	/load/dc	ipso.load.dc	p	d	%

Event Identifier: The event identifier as a string.

Start Time: Time when the load control event will start stated as the number of seconds that have elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970. If set to 0, the load control event starts immediately.

Duration in Min: The duration of the load control event.

Criticality Level: The criticality of the event. The device receiving the event will react in an appropriate fashion for the device.

Avg Load Adj Pct: (optional) Defines the maximum energy usage of the receivng device, as a percentage of the device's normal maximum energy usage.

Duty Cycle: (optional) Defines the duty cycle for the load control event, i.e, what percentage of time the receiving device is allowed to be on.

Enumeration	Name
0	Green
1	Level_1
2	Level_2
3	Level_3
4	Level_4
5	Level_5
6	Emergency
7	Planned_Outage
8	Service_Disconnect

Table 2: Criticality Level Enumerations

2.5. Sensors

This function set is used to represent simple types of sensors. This function set MAY be extended with custom resource types.

Type	Path	RT	IF	Type	Unit
Motion	/sen/{#}	ipso.sen.mot	s	i	
Motion Status	/sen/{#}/status	ipso.sen.mot.status	s	b	
Contact	/sen/{#}	ipso.sen.con	s	i	
Contact Status	/sen/{#}/status	ipso.sen.con.status	s	b	
Generic	/sen/{#}	ucum.{unit}	s	d	

Motion: This resource type represents a motion sensor, and is modeled as an integer counter that is increased by one each time an event occurs. When first initialized, this resource MUST be set to 0.

Motion Status: This resource type represents the status of a motion sensor as a boolean value, with 0 indicating no presence and 1 indicating presence.

Contact: This resource type represents a contact or other kind of proximity sensor, and is modeled as an integer counter that is increased by one each time an event occurs. When first initialized, this resource MUST be set to 0.

Contact Status: This resource type represents the status of a contact sensor as a boolean value, with 0 indicating no no contact and 1 indicating contact.

Generic: This resource type allows sensors with generic units to be represented with a decimal value. The resource type is set to "ucum:{unit}" where {unit} is replaced by one of the units in the Unified Code for Units of Measure (UCUM) specification [unitsofmeasure.org].

2.6. Light Control

This function set is used to control a light source, such as a LED or other light. It allows a light to be turned on or off and its dimmer setting to be control as a % between 0 and 100. Not all lights are expected to have a dimmer control (for example simple LEDs).

Type	Path	RT	IF	Type	Unit
Light Control	/lt/{#}/on	ipso.lt.on	a	b	
Light Dimmer	/lt/{#}/dim	ipso.lt.dim	a	i	0-100 %

Light Control: This resource represents a light, which can be controlled, the setting of which is a Boolean value (1,0) where 1 is on and 0 is off. A GET on the resource returns the current state of the light, and a PUT on the resource sets a new state.

Light Dimmer: This resource represents a light dimmer setting, which has an Integer value between 0 and 100 as a percentage. A GET on the resource returns the current state of the dimmer, and a PUT on the resource sets a new state.

2.7. Message

This function set is used by a smart object to make messages available to others, for example status, alarms, user display strings and other logs. This function set MAY be extended with custom resource types.

Type	Path	RT	IF	Type	Unit
Status	/msg/status	ipso.msg.status	p,rp	s	
Alarms	/msg/alarms	ipso.msg.alarms	p,rp	s	
Display	/msg/disp	ipso.msg.disp	p,rp	s	

Status: The current status of the device as a string.

Alarms: Alarm messages of the device as a string. This resource returns the latest alarm.

Display: This resource is used to provide an interface to a text display.

2.8. Location

This function set allows the location of a device and related meta-data to be made available, for example in map based or tracking applications. This function set is designed to be useable by both GPS coordinate and local coordinate based positioning.

Type	Path	RT	IF	Type	Unit
GPS Location	/loc/gps	ipso.loc.gps	s	s	Lon,Lat
XY Location	/loc/xy	ipso.loc.xy	s	s	X,Y
Semantic Location	/loc/sem	ipso.loc.sem	s	s	
Fix	/loc/fix	ipso.loc.fix	s	b	
Period	/loc/per	ipso.loc.per	p	i	s

GPS Location: This resource represents the current global position of the device as a String with the format {Lon,Lat}.

XY Location: This resource represents the current local position of the device as a String with the format {X,Y} where X and Y are in meters. It is assumed that the anchor position for the application or deployment is known.

Semantic Location: This resource can contain the semantic location of a device, such as "room5", "Corner Cafe", "High Street 24" or "Building A".

Fix: This resource indicates if there is currently a valid fix on the location as a Boolean value (1,0).

Period: This resource is the period that the location values are updated as an Integer in seconds. A GET on this resouce retrieves the current period, and a PUT sets a new period.

2.9. Configuration

This function set is used to configure an end-point, or discover configuration parameters from another end-point.

Type	Path	RT	IF	Type	Unit
Services	/cfg/services	ipso.cfg.services		s	
Stack	/cfg/stack	ipso.cfg.stack	p	s	
Stack PHY	/cfg/stack/phy	ipso.cfg.stack.phy	p	s	
Stack MAC	/cfg/stack/mac	ipso.cfg.stack.mac	p	s	
Stack NET	/cfg/stack/net	ipso.cfg.stack.net	p	s	
Stack RTG	/cfg/stack/rtg	ipso.cfg.stack.rtg	p	s	

Services: The services resource type contains a list of links in CoRE Link Format (application/link-format) that describe the services supported by an end-point, and the service configurations of the end-point. An example service that an end-point could be configured for is a Resource Directory. A service is identified by its Resource Type which is included in the rt= field of a link (a Resource Directory has rt="core.rd" for example). A GET to this resource returns the list of supported services and services currently configured as links. A PUT to this resource replaces the configured services with the links included in the body of the request. A DELETE to this resource removes the configured services.

Stack: The Stack set of resources describes and configures the set of communication stack protocols supported by this endpoint.

The Services resource returns its list of supported services by including the following link in its Services links where the list of supported services is included in the Resource Type attribute:

```
</cfg/services>;rt="core.rd core.mp foo"
```

An example of a configured Resource Directory service is the following:

```
<coap://{IP address of RD}/rd>;rt="core.rd";d="default";ep="sensor693"
```

3. Data Formats

This section descibes the resource representations supported by this framework in more detail.

3.1. text/plain

When the resource reprsentation of a sensor, configuration parameter or other information is fully atomic and does not require any meta-data to be included with it, then plain text is an ideal way to represent the data. In this framework, the resource representations of all interface types MUST be available in this format, indicated by the Media Type text/plain. In this data format, all data types are represented in ASCII using XSD data type definitions.

3.2. SenML

The Sensor Markup Langauge (SenML) is an IETF specification that provides a simple way to represent sensor and other parameter information typically available from constrained embedded devices [I-D.jennings-senml]. SenML provides an object model where an array of measurement or parameter values along with common base information can be modeled. This object can then be serialized in JSON, XML or EXI formats. By default, this framework makes use of the JSON format of SenML. Resource representations in this framework MAY also be provided in application/senml+json format in addition to text/plain.

In this data format, data types are represented using SenML value fields.

Data Type	text/plain Format	SenML Field
b (boolean)	xsd:boolean	Boolean Value (v)
s (string)	xsd:string	String Value (sv)
e (enum)	xsd:integer	Value (v)
i (integer)	xsd:integer	Value (v)
d (decimal)	xsd:decimal	Value (v)

4. Examples

An example design of a theoretical plug socket sensor, with two plugs (index of 0 and 1) a button and a LED, along with temperature and CO2 sensors in CoRE Link Format. The following examples make use of this resource design.

```
</dev/mfg>;rt="ipso.dev.mfg",
</dev/ser>;rt="ipso.dev.ser",
</dev/mdl>;rt="ipso.dev.mod",
</pwr/0/w>;rt="ipso.pwr.w",
</pwr/0/kwh>;rt="ipso.pwr.kwh",
</pwr/0/rel>;rt="ipso.pwr.rel",
</pwr/1/w>;rt="ipso.pwr.w",
</pwr/1/kwh>;rt="ipso.pwr.kwh",
</pwr/1/rel>;rt="ipso.pwr.rel",
</gpio/btn/0>;rt="ipso.gpio.btn",
</lt/led0/on>;rt="ipso.lt.on",
</sen/temp>;rt="ucum.Cel";obs,
</sen/co2>;rt="ucum.ppm"
```

In this example the resources in the Device Function Set are requested.

```
Req: GET /dev/mfg (Accept: text/plain)
Res: 2.05 Content (text/plain)
     Body: IPSO Alliance

Req: GET /dev/ser (Accept: text/plain)
Res: 2.05 Content (text/plain)
     Body: 3450232

Req: GET /dev/mdl (Accept: text/plain)
Res: 2.05 Content (text/plain)
     Body: SuperPlug300
```

In this example the power of plug 0 is read (123 W), the relay shut off (0), and the power read again (0 W).

```
Req: GET /pwr/0/w (Accept: text/plain)
Res: 2.05 Content (text/plain)
     Body: 123

Req: PUT /pwr/0/rel (Accept: text/plain)
     Body: 0
Res: 2.04 Changed (text/plain)

Req: GET /pwr/0/w (Accept: text/plain)
Res: 2.05 Content (text/plain)
     Body: 0
```

This example shows the use of CoAP Observe to push a new value of the /sen/temp sensor from the server to the client whenever the resource changes.

```
Req: GET /sen/temp (Accept: text/plain) (Observe:0)
Res: 2.05 Content (text/plain) (Observe:0)
     Body: 23

Res: 2.05 Content (text/plain) (Observe:1)
     Body: 23.5

Res: 2.05 Content (text/plain) (Observe:2)
     Body: 24

Res: 2.05 Content (text/plain) (Observe:3)
     Body: 24.6

...
```

This example shows the use of CoAP Observe to push a new value of the /sen/temp sensor from the server to the client whenever the resource changes by at least 1 degree.

```
Req: GET /sen/temp?st=1 (Accept: text/plain) (Observe:0)
Res: 2.05 Content (text/plain) (Observe:0)
     Body: 23

Res: 2.05 Content (text/plain) (Observe:1)
     Body: 24

...
```

5. Security Considerations

The security considerations discussed in [I-D.ietf-core-coap], [I-D.ietf-core-link-format] and [I-D.shelby-core-interfaces] apply to this specification.

Writable parameter resources and actuator resources SHOULD be protected with DTLS or TLS including access control, unless a sufficient underlying security mechanism is available.

6. Acknowledgments

Acknowledgement is given to members of the IPSO Alliance where the initial idea and further improvements to this document have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle, Robert Assimiti, Jari Arkko and Milt Roselinsky who have provided useful discussion and input to the concepts in this document.

7. Changelog

Changes from -03 to -04:

- Added the load control function set
- Added the configuration function set
- New device function set resources
- New GPIO dimmer input resource
- Improved the table structure
- Added an enumeration data type

8. References

8.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-11 (work in progress), July 2012.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.

[I-D.jennings-senml]
Jennings, C., Shelby, Z., and J. Arkko, "Media Types for Sensor Markup Language (SENML)", draft-jennings-senml-09 (work in progress), July 2012.

[I-D.shelby-core-interfaces]
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-shelby-core-interfaces-03 (work in progress), July 2012.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

8.2. Informative References

[I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-04 (work in progress), July 2012.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Cedric Chauvenet
Watteco
1766, Chemin de la Planquette
La Garde 83130
France

Phone:
Email: c.chauvenet@watteco.com