Smart Home
Technical Specification

Version 1.0

Status:	Final
Version:	1.0
Date:	4th September 2014
Author:	Smart TV Alliance, Inc.
Category:	Released

CONTENTS

Change History

Version	Date	Changes
1.0	2014-09-04	Final version 1.0 for publication

1. Introduction

1.1. Overview

This document sets out version 1.0 of the Smart TV Alliance Smart Home specification. It is intended primarily for manufacturers but also for application developers, and describes the common technical features in the form of a data model and common transport protocol. Version 1.0 of the specification is focused on four basic functional use cases of appliances and gateways in the local area network:

- Discover appliances in the local area network within an application running on a Smart TV
- Monitoring appliances from an application running on a Smart TV
- Controlling appliances from an application running on a Smart TV
- Notification from appliances to an application running on a Smart TV

These basic functional use cases enable applications to create business use cases, as these basic functionalities can be freely composed together as needed.

Smart TV Alliance will also release a Software Development Kit and developer documentation. This will provide a user-friendly environment for developers to create Smart Home applications that run on Smart TV Alliance platform.

While a lot of care has been taken to ensure the correctness of the information in this document, errors cannot be completely prevented. The latest version of this document, with possible corrections, is always available online. If you have questions and/or remarks regarding these guidelines, please post them through the designated support channels.

1.2. Definitions

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
mDNS	Multicast Domain Name System
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transport Protocol (Secure)
JSON	JavaScript Object Notation
OBIX	Open Building Information Exchange
SSDP	Simple Service Discovery Protocol
UPnP	Universal Plug and Play
WS(S)	WebSocket (Secure)
XML	Extensible Markup Language
Zeroconf	Zero Configuration Networking

1.3. References

[1] Smart TV Alliance Specification Version 4
https://developers.smarttv-alliance.org/specification

[2] HTML5 Working Draft 29 March 2012
http://www.w3.org/TR/2012/WD-html5-20120329/

[3] W3C Network Service Discovery API
http://www.w3.org/TR/2013/WD-discovery-api-20130404/#service-discovery

[4] Bindings for OBIX: Web Socket Bindings Version 1.0
http://docs.oasis-open.org/obix/obix-websocket/v1.0/obix-websocket-v1.0.html

[5] The application/json Media Type for JavaScript Object Notation (JSON), July 2006
http://tools.ietf.org/html/rfc4627

[6] HTML5 WebSockets – Candidate Recommendation September 2012
http://www.w3.org/TR/2012/CR-websockets-20120920/

[7] RFC2782
https://www.ietf.org/rfc/rfc2782.txt

[8] RFC1035
https://www.ietf.org/rfc/rfc1035.txt

[9] RFC5198
https://www.ietf.org/rfc/rfc5198.txt

[10] RFC6763
https://www.ietf.org/rfc/rfc6763.txt

[11] RFC6762
https://www.ietf.org/rfc/rfc6762.txt

1.4. Trademarks and copyrights

2. Technical Specification

2.1. Introduction

This version of the specification contains standardization for an application running on a TV and how it can discover home appliances as well as receive and send data to them. It does explicitly not contain any standardization regarding how an application and an appliance connect itself to a cloud infrastructure.

Following entities are considered in this specification:

- *Application*: an HTML5 Smart TV application compatible with Smart TV Alliance technical specifications [1] running on a Smart TV
- *Home appliance/device*: a smart appliance which is capable to run a server component and can be addressed within a home area network
- *Home Gateway*: a component which abstracts various devices using non-IP or proprietary connection methods, e.g. ECHONET Lite[1], EnOcean[2] or ZigBee[3] devices, and exposes them over an IP network, it is usually located in a private home and connected to a home area network that is connected to the internet via broadband gateway
- *Cloud Server*: a component located in a data center and reachable via the internet
- *Common Interface*: a protocol definition which abstracts a generic home appliance/device concept and provides methods to control, monitor and notification, optionally it also provides a discovery mechanism
- *Endpoint*: abstract notion of an home appliance/device, a TV, a home gateway or a cloud server implementing the common interface

There are three communication paths to look at in respect of this specification:

- Application (on a Smart TV) to home appliance/device using home network connection like Ethernet/Wi-Fi
- Application (on a Smart TV) to home gateway using home network like Ethernet/Wi-Fi
- Application (on a Smart TV) to cloud server in the cloud through broadband gateway

All of these will have a common interface and they are all providing an endpoint for WebSocket [6] connections. See Figure 1 below:

[1] http://www.echonet.gr.jp/english/
[2] http://www.enocean-alliance.org/
[3] http://www.zigbee.org/

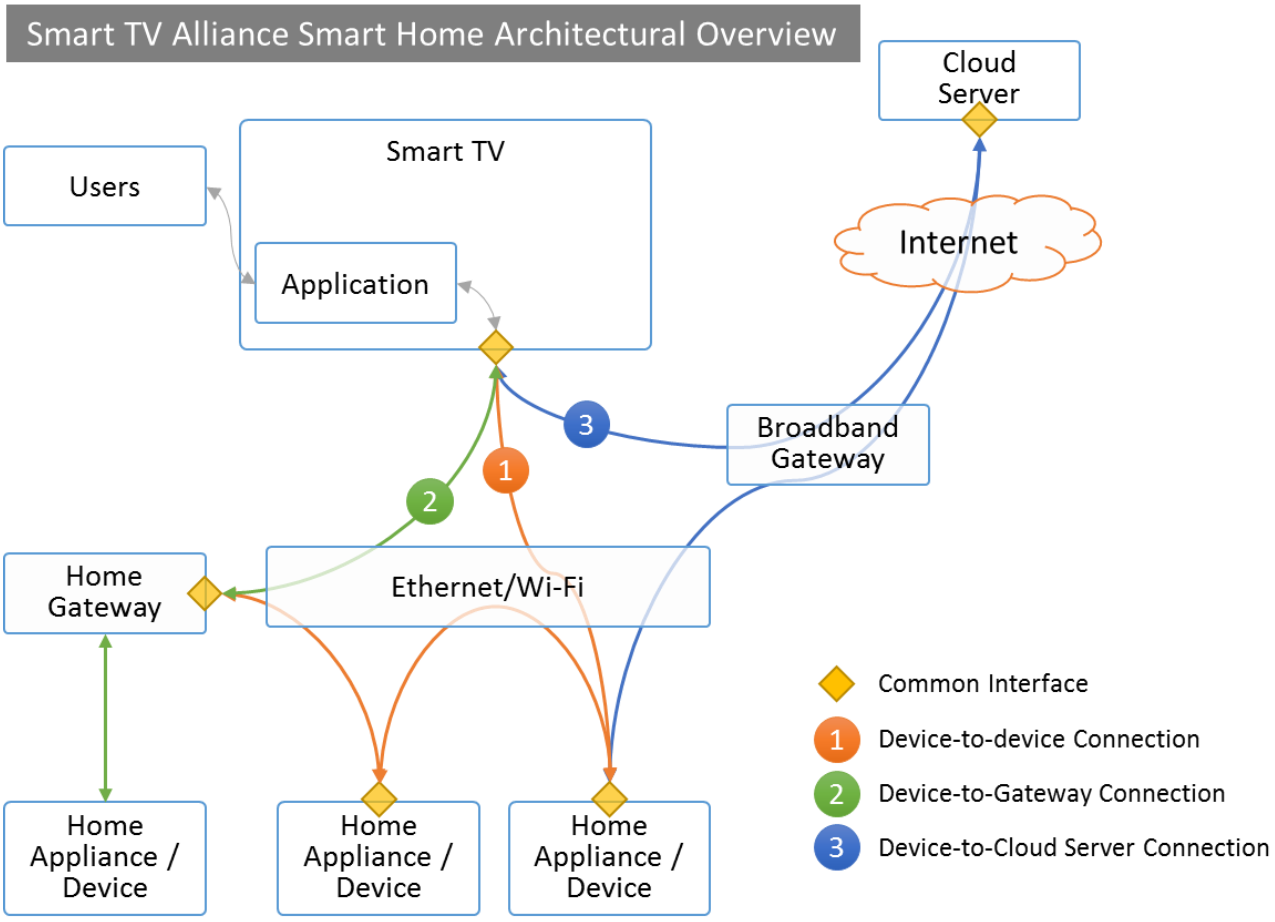

Figure 1 Smart TV Alliance Smart Home Architectural Overview

The home appliances/devices and gateways implement a common interface, which is a WebSocket server interacting with the protocol defined below. For implementation, an endpoint can chose which protocol version it will implement. An endpoint can also support discovery in the local area network.

2.2. Discovery

In this section it is described how an application can detect home appliances/devices within the local area network using standard technologies. There are some assumptions for the discovery as this is done within the application running on a TV:

- Prerequisite is that the browser supports a technology like Network Service Discovery API [3], which is a part of TV device specification [1]
- The user initiates / allows the search for home appliances/devices

For the Discovery mechanism, SSDP and Zeroconf on the device layer can be used. The discovery via SSDP and Zeroconf is optional to conform to this specification.

2.2.1. SSDP

Appliances/devices supporting WebSocket Server can be discovered by SSDP. The client can get WebSocket Server URL via M-SEARCH request according to the following procedure.

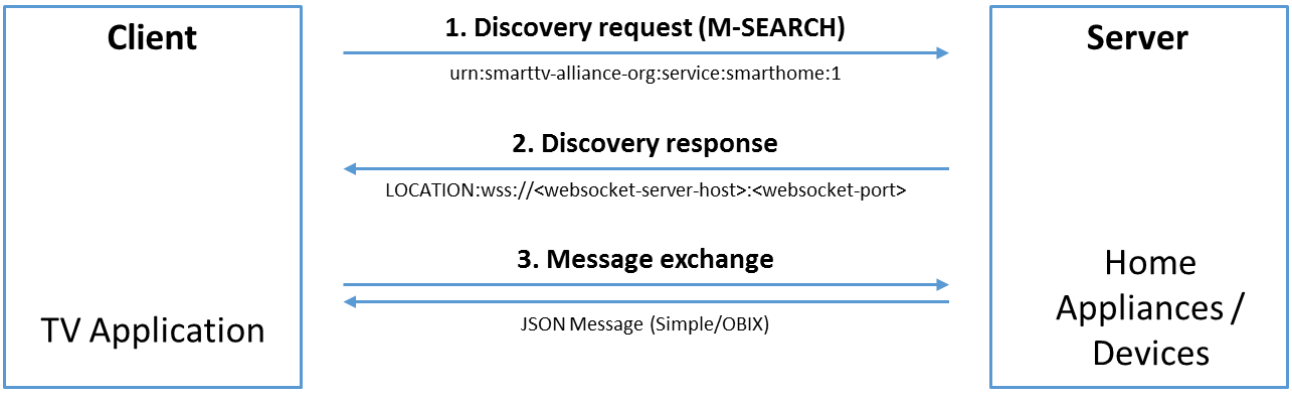

Figure 2 Device Discovery Mechanism using SSDP

Please note that the references to UPnP do not imply that an entire UPnP stack is required to support STASH device discovery specification. Only the SSDP portion of UPnP is required.

Device discovery based on SSDP consists of three messages:

1. Discovery request message (M-SEARCH)
   Appliances that support this specification can be discovered through the SSDP M-SEARCH method. The search target header should be

   ST: urn:smarttv-alliance-org:service:smarthome:<Device Contract>:<STASH Spec Version>

   All multicast messages are sent to the reserved address and port 239.255.255.250:1900, which is a default multicast address and port for UPnP specification.

   To enhance readability, "-" is used instead of "." in the value of URN.

   <Device Contract> refers to the devices that clients want to discover. If clients want to discover air conditioners only, <Device Contract> should be "sta:AirConditioner." If clients want to discover all the compatible appliances in the network, <Device Contract> should be "sta:Device." In this version of specification, clients can use these kinds of <Device Contract>:
   -sta:Device
   -sta:AirConditioner
   -sta:Refrigerator
   -sta:Washer
   -sta:Dryer
   -sta:WasherDryer
   -sta:Cleaner
   -sta:Light

   <STASH Spec Version> refers to the version of the STASH specification.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: <seconds to delay response e.g., 3>
ST: urn:smarttv-alliance-org:service:smarthome:<Device Contract>:<STASH Spec Version>
USER-AGENT: OS/version product/version
```

2. Discovery response message

Appliances/devices that support this specification will respond with a unicast response with IP address and port of an appropriate WebSocket server. The value of ST header is same with the one in the M-SEARCH request. The value of encoding parameter refers to the preferred message format and version for the device and the value of version refers to the version of the preferred message format, see section 3.4 and 3.5 below. The unique service name can be delivered through the value of USN header.

```
HTTP/1.1 200 OK
LOCATION:wss://<websocket-server-host>:<websocket-port>?encoding=<simple/obix>&version=<message format version>
CACHE-CONTROL: max-age= <seconds until advertisement expires e.g., 1800>
EXT:
SERVER: OS/version UPnP/1.0 product/version
ST: urn:smarttv-alliance-org:service:smarthome:<Device Contract>:<STASH Spec Version>
USN: <Unique Service Name>
```

3. Notify message
   When an appliance/device is added to the network, NOTIFY discovery message can be used to advertise itself. The value of NT (Notification Type) header must be same with Search Target of M-SEARCH request message.

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age= <seconds until advertisement expires e.g., 1800>
LOCATION:wss://<websocket-server-host>:<websocket-port>?encoding=<simple/obix>&version=<message format version>
NT: urn:smarttv-alliance-org:service:smarthome:<Device Contract>:<STASH Spec Version>
NTS: ssdp:alive
SERVER: OS/version UPnP/1.0 product/version
USN: <Unique Service Name>
```

2.2.2. Zeroconf (mDNS+DNS-SD)

Appliances/devices supporting WebSocket Server can be discovered by using complementary standards mDNS and DNS-SD. In small networks, such as home networks, mDNS can be used for the name resolution and DNS-SD can be used to allow clients to discover the services by type using standard DNS queries.
A particular service instance can be described by using a DNS SRV [7] and DNS TXT [8] record. A client discovers the list of available instances of a given service type using a query for a DNS PTR [8] record with a name of the form "<servicename>.<servicedomain>". This returns a set of instances (if available), which contains DNS SRV/TXT record pairs.

a) <u>The DNS SRV record:</u>

PTR Service Instance Name: "**<Instance>.<servicename>.<servicedomain>.<parentdomain>.**"
<Instance>: User-friendly name consisting of arbitrary Net-Unicode text [9]. e.g.: MyFridge (may be up to 63 bytes [10])
<servicename>: _smarthome._<Device Contract>._tcp
<Device Contract>: refers to the devices that clients want to discover. In this version of specification, clients can use the following kinds of <Device Contract>:

- AirConditioner
- Refrigerator
- Washer
- Dryer
- WasherDryer
- Cleaner
- Light

<servicedomain>: smarttv-alliance-org

<parentdomain>: .local.

b) The key/value pairs in DNS TXT record should be
encoding=<simple/obix>
version=<message format version>

Whenever an mDNS responder starts up, it sends a Multicast DNS query to probe the network for the service resource conflicts. After being sure that there is no conflict of resources it announces its services in order to notify the others [11].

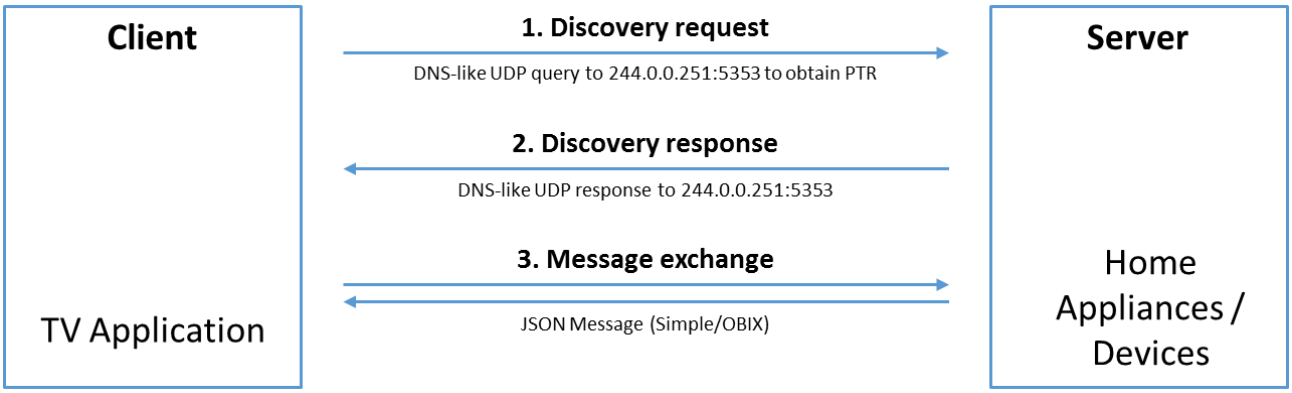

Figure 3 Device Discovery Mechanism using mDNS

2.3. Connection

To establish a connection to the appliance, a client needs to connect to the URL using the WebSocket protocol. A WebSocket connection is established with first establishing an HTTP connection and then doing a change to the WebSocket protocol.

The discovered URL should contain a query parameter called "encoding" to denote which transport protocol variant the endpoint supports. The value can be either "simple" or "obix" referring to the two variants described below.

2.4. Transport Protocol: Simple Variant

The "Simple" transport protocol defines a message format and operations on top of the WebSocket protocol. This protocol is designed for home appliances/devices that have only limited resources. The operations can be used after successfully having established a WebSocket session. The JSON data format is used as data format, see [5].

The current value of <message format version> is v1. If there are any updates of message format, the exact version number will be specified in the tables below.

2.4.1. Transaction Message Format

In each message, there are fields containing header information and a payload field containing the request, response, notification or error data:

Name	Type	Description
type	String	Is either "request", "response", “notify” or "error"
tid	Integer	Transaction id to correlate the requests Range: 0-99 0 is reserved for notify / error type
payload	JSON Object	Contains the request, response, notification or error data

2.4.2. Device Message Format

Below is a JSON object model for a device:

Name	Type	Description
deviceId	String	Unique ID to identify a home appliance/device. ASCII and less than 256 characters.
is	String	Device class (e.g. “AirConditioner”) as found in the data model section below
displayName	String	Appliance/device name (end-user readable and editable), UTF-8 and less than 32 characters.
location	String	Place name (end-user readable and editable), UTF-8 and less than 32 characters.
status	JSON Object	Pairs of property name and value
writable	String array	Name of writable properties
vendorCode	String	Vendor identifier

2.4.3. Operations

Following operations issued from the client to the server are supported:

Get devices

Request: JSON object containing "getDevices" property in the payload object. The "getDevices" property can be null or the exact deviceId as string.

Response: JSON object containing "devices". The "devices" property is a JSON array of device as defined above.

Set status properties

Request: JSON object containing "deviceId" and "setStatus" property in the payload object. The "deviceId" denotes the unique identifier of the device, "setStatus" is a JSON object containing property name and value.

Response: JSON object containing "deviceId" and "accepted". The "deviceId" denotes the unique identifier of the device, "accepted" is a JSON array containing the property names of which the status change was executed.

Get status properties

Request: JSON object containing "deviceId" and "getStatus" in the payload object. The "deviceId" denotes the unique identifier of the device, "getStatus" is a JSON array containing the status property names which are requested – it can be empty to retrieve all properties.

Response: JSON object containing "deviceId" and "status". The "deviceId" denotes the unique identifier of the device, "status" is a JSON object containing property name value.

Set device attributes

Request: JSON object containing "deviceId" and "setAttributes" in the payload object. The "deviceId" denotes the unique identifier of the device, "setAttributes" is a JSON object containing the attribute names and values to be updated.

Response: JSON object containing "deviceId" and "accepted". The "deviceId" denotes the unique identifier of the device, "accepted" is a JSON array containing the attribute names of which the status change was executed.

Get device attributes

Request: JSON object containing "deviceId" and "getAttributes" in the payload object. The "deviceId" denotes the unique identifier of the device, "getAttributes" is a JSON array containing the attribute names which are requested – it can be empty to retrieve all attributes.

Response: JSON object containing "deviceId" and "attributes". The "deviceId" denotes the unique identifier of the device, "attributes" is a JSON object containing property name value.

Notification and error reporting

Notify: JSON object containing "deviceId" and "message" in the payload object. The "deviceId" denotes the unique identifier of the device, "message" is a JSON object containing text of what the device wants to notify a client.

Error: JSON object containing "deviceId" and "error" or "errorStatus". The "deviceId" denotes the unique identifier of the device, "error" is a JSON object containing text describing the error. "errorStatus" is a JSON object containing a value of error code.

2.4.4. Vendor Dependent Extension

If a vendor wants to add a new type of home appliance/device, additional definitions of Device class, attributes and values are available for this Simple Transport Protocol. It is preferable for a server and client to ignore unknown Device class, attributes and values.

2.5. Transport Protocol: OBIX Variant

The OBIX transport protocol is specified in [4]. After establishing a WebSocket connection the server returns the OBIX Lobby object and then it can be navigated from there.

The current value of <message format version> is v1. If there are any updates of message format, the exact version number will be specified in the tables below.

The server need not to comply fully as OBIX Server supporting WebSocket, but it must comply with following:

- After successful establishment of a connection it must respond with the OBIX Lobby object
- A "device" object with the obix:href must be specified to denote all the home appliances/devices
- The home appliances/devices themselves are obix:obj and the obix:is facet denotes the class name from the data model defined below
- The obix:Read and obix:Write must be supported for the device obix:obj
- The contracts for the devices must comply with the given contracts in this specification

Additionally, the server may provide a WatchServer to receive updates continuously. For the notification use case there is no separate message type but it is handled as obix:Update message and pushed using the WatchService.

2.6. Orthogonal Component: Security

Security consists of three parts: transport layer security, authentication and authorization. All of them are considered within the specification and they are defined what level of support is required to comply with this specification.
In addition, home appliances/devices may support CORS, see section 3.3.2.3 in [1].

2.6.1. Transport Layer Security

A server component implementing the common interface is recommended to support secure WebSocket wss:// connections.

2.6.2. Authentication

A server component implementing the common interfaces may support authentication with username and password (usually referred to as “Basic Authentication”) or client certificates.

2.6.3. Authorization

Each home appliance/device may have two levels of authorization:

- on the device level, defined by manufacturers: read/update (this disallows/allows an application to read properties of the home appliance/device or update the home appliance/device using capabilities)
- on the properties/capabilities level, defined by applications: read/update (this disallows/allows a user to read or update a home appliance/device)

For that the endpoint needs to support authorization and provides e.g. a role-based authorization framework (which is usually not a single appliance but a gateway or the cloud). The common interface does not require the integration of such an authorization framework but it strongly recommends that the appliance/device itself supports this.
For example, it could be an appliance-based mode where the access credentials for remote access can be directly set on the display on the device or a cloud-based approach where the credentials can be set on the cloud and then get pushed back to the appliance.

3. Data Model

OBIX [4] uses the notion of a contract to group a set of properties; this concept is also used in this specification. The contracts defined below are specified to have a common data model among the CE manufacturers and must be supported by compliant endpoints. These contracts define the basic set of properties and operations, but as the object structure itself is not restricted additional vendor specific properties and operations can be used ad libitum. A contract inherits its parent contract properties and operations and thus it is a device type hierarchy on contracts built.

The data model comprises the contract hierarchy as well as the contract definition with properties and operations. Please find in the figure 4 below, a graphical overview of the defined contracts, the details regarding mandatory and optional attributes are described further below:

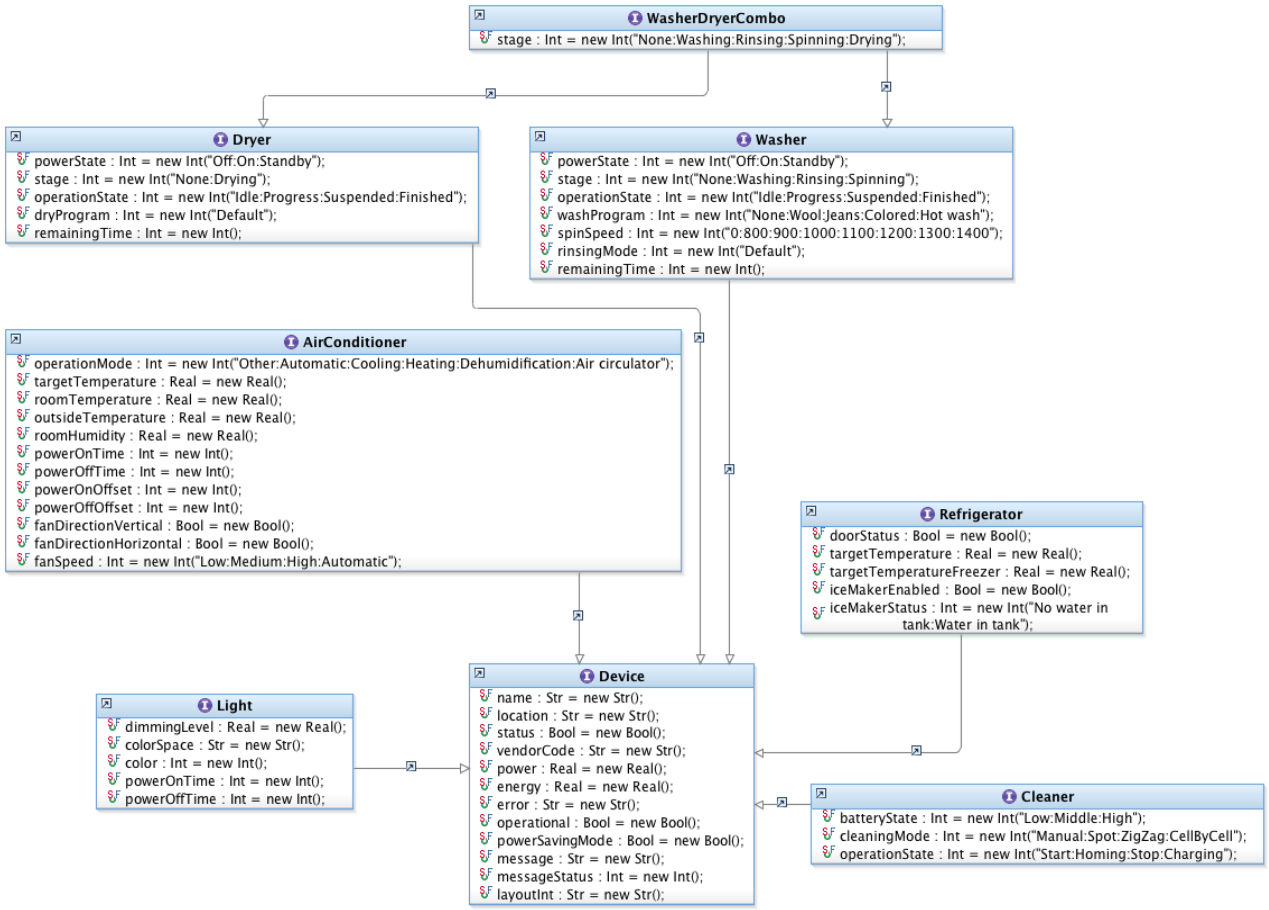

Figure 4 Device Type Hierarchy

Remarks regarding the contract definitions below:

- The type of a property can be the simple type “Boolean”, “String”, “Integer”, “Double”. There is also a more complex type specified, called “String Value List”, which means the “display” attribute contains the possible values as colon separated string list and the value attribute itself contains the index of the selected value within the list (starting at 0). The “String Value List” type could change in future versions.
- Empty or unavailable properties can be omitted completely to save transmission data.
- If “Values” contains a numeric range, set as “min” and “max” attribute, this also easily allows home appliance/device or vendor specific ranges
- An asterisk “*” marks that the property and the value of this property must be present
- Other properties are optional

For examples of the JSON messages in the various transport protocol encodings see the annex.

3.1. Device Contract

Device denotes that this entry is a device; it is the top-level element in the hierarchy.

Property	Type	Writeable	Values	Unit
status*	Boolean	(vendor and device type specific)	true / false	-
Denotes that the status of the device, if it is switched on or off				
name*	String	(vendor and device type specific)	text (255 chars)	
Name of the device, unique at least within the application. Encoded in ASCII.				
displayName	String	true	text (255 chars)	
Localized name of the device. Encoded in UTF-8.				
location*	String	(vendor specific)	text (255 chars)	
User specified name of the location the device is in. Encoded in UTF-8				
layoutHint	String	False	Text (? chars)	
A layout hint given by the device how to render it				
Power	Integer	False		W
Instantaneous power consumption				
Energy	Integer	False		Wh
Cumulative power consumption				
operational	Boolean	False	true / false	
Denotes that the device can be used and is in a controllable state, it can be omitted if value is "true"				
Error	String	False	text (255 chars)	
Textual description if error is present (i.e. operational == false)				
errorStatus	Integer	False	If no error is present, this is -1	
Denotes an integer value for machine processing in case of an error (HTTP Status Codes can be reused)				
powerSavingMode	Boolean	(vendor specific)	true / false	
Denotes if the device is in power saving mode				
Message	String	false	text (2048 chars)	
Content of an end-user notification message, containing title and message text, can be formatted				
messageStatus	Integer	false		
HTTP Status Code				
vendorCode*	String	false	text (255 chars)	
Company name				

3.2. AirConditioner Contract

Contract Hierarchy: "sta:AirConditioner" → "sta:Device"

Property	Type	Writeable	Values	Unit
operationMode*	String Value	read (optional: true)	Basic values:	-

	List		”Automatic”: 0x01, ”Cooling”: 0x02, ”Heating”: 0x03, “Drying”: 0x04, “Ventilating”: 0x05 other values are possible but vendor specific	
Denotes the air conditioners operation mode				
roomTemperature	Integer	false	-127 to 125	°C or °F
Denotes the current room temperature				
targetTemperature*	Integer	true	0 to 50	°C or °F
Denotes the target temperature				
outsideTemperature	Integer	false	-127 to 125	°C or °F
Denotes the outside temperature				
fanSpeed	String Value List	(vendor specific)	Basic values: “Low”: 1, “Medium”: 2, “High”: 3, "Automatic": 4, other values are possible but vendor specific	
Denotes the speed of the fan				
fanDirectionVertical	Boolean	(vendor specific)	true / false	-
Denotes if the fan moves vertical				
fanDirectionHorizontal	Boolean	(vendor specific)	true / false	-
Denotes if the fan moves horizontal				
powerOnTime	Integer	(vendor specific)	0000 to 2359	HHmm
An optional property denoting when to power on the air conditioner specified as absolute time				
powerOffTime	Integer	(vendor specific)	0000 to 2359	HHmm
An optional property denoting when to power off the air conditioner specified as absolute time				
powerOnOffset	Integer	(vendor specific)	0000 to 2359	HHmm
An optional property denoting when to power on the air conditioner specified as relative time				
powerOffOffset	Integer	(vendor specific)	0000 to 2359	HHmm
An optional property denoting when to power off the air conditioner specified as relative time				

3.3. Refrigerator Contract

Contract Hierarchy: “sta:Refrigerator” → “sta:Device”

Property	**Type**	**Writeable**	**Values**	**Unit**
targetTemperature*	Double	(vendor specific)	0.0 to 10.0	°C
Property to denote the refrigerator compartment target temperature				
compartment###	Double	(vendor	-50.0 to 50.0	

		specific)		
Property to denote the compartment target temperature, a display name can be used to describe the compartment. ### denotes the compartment number.				
mode	String Value List	true	Not predefined, could be "Turned off" : 0, "Power mode": 1, "Standby": 2, "Eco mode": 3	
Returns all modes the refrigerator supports within the "values" attribute				
iceMakerEnabled	Boolean	false	true / false	
Returns true if the ice maker is available and enabled				
iceMakerStatus	String Value List	false	Basic values: "No water in tank": 0, "Water in tank": 1	
Returns the status of the ice maker				
doorStatus	Boolean	false	true / false	
Returns true if the door is open and false if closed				

3.4. Washer Contract

Contract Hierarchy: "sta:Washer" → "sta:Device"

Property	**Type**	**Writeable**	**Values**	**Unit**
powerState	String Value List	false	Not predefined, could be: On: 0, Off: 1, Standby: 2	
Denotes the power state of the washing machine				
washingStage	String Value List	false	Not predefined, could be: None: 0, Washing: 1, Rinsing: 2: Spinning: 3, Drying: 4, Completed: 5	
Denotes the current washing stage				
operationState	String Value List	(vendor specific)	Basic values: "Idle": 0, "Progress" : 1, "Suspended": 2, "Completed": 3	
Denotes the operation of the washing machine				
washingProgram	String Value List	(vendor specific)	Not predefined, could be "Weak", "Normal", "Strong" or a program like "Jeans", "Colored", ...	
Denotes the strength or type of the selected washing mode				
spinSpeed	String Value List	false	Not predefined, could be "Weak", "Normal", "Strong", "Off" or "0", "1000", "1200", etc.	
Denotes the speed of spin				
rinsingMode	String Value List	false		
Denotes the strength level of rinsing				
remainingTime*	Integer	false	Hours and minutes	HHmm
Denotes the remaining washing time				

3.5. Dryer Contract

Contract Hierarchy: “sta:Dryer” → “sta:Device”

Property	Type	Writeable	Values	Unit
powerState	String Value List	(vendor specific)	Not predefined, could be: On: 0, Off: 1, Standby: 2	
Denotes the power state of the washing machine				
dryingProgram	String Value List	(vendor specific)		
Denotes the strength or type of the selected drying mode				
dryingStage	String Value List	false	Not predefined, could be: None: 0, Drying: 1	
Denotes the current drying stage				
remainingTime	Integer	false	Hours and minutes	HHmm
Denotes the remaining washing time				
operationState	String Value List	(vendor specific)	Basic values: “Idle”: 0, “Suspended”: 1, “Drying in progress”: 2, “Finished”: 3	
Denotes the operation of the dryer				

3.6. WasherDryer Contract

Contract Hierarchy: “sta:WasherDryer” → “sta:Washer”, "sta:Dryer"

Stage property does not have pre-defined values and differs from Washer and Dryer contract.

No additional properties or operations

3.7. Cleaner Contract

Contract Hierarchy: "sta:Cleaner" → "sta:Device"

Property	Type	Writeable	Values	Unit
operationState	String Value List	true	Basic values: “Start”:0,“Homing”:1, “Stop”:2, “Charging”:3	
Denotes the operation state of the robot cleaner				
batteryState	String Value List	false	Basic values: “Low”:0, “Middle”:1, “High”:2	
Denotes the battery state of robot cleaner				
cleaningMode	String Value List	true	Not predefined, could be: “Manual”:0, “Spot”:1, “ZigZag”:2, “CellByCell”:3	
Denotes the cleaning mode of robot cleaner				

3.8. Light Contract

Contract Hierarchy: “sta:Light” → "sta:Device"

Property	Type	Writeable	Values	Unit
dimmingLevel	Integer	(vendor specific)	0 to 100	%
Denotes the level of dimming if supported				
colorSpace	String	(vendor specific)	Default is "RGB"	
Denotes the color space which is used (usually RGB)				
color	Integer	(vendor specific)	In RGB (0,0,0) to (255,255,255)	
Denotes the color the light currently has				
powerOnTime	Integer	true	Hours and minutes	HHmm
Denotes the on timer				
powerOffTime	Integer	true	Hours and minutes	HHmm
Denotes the off timer				

4. STASH JavaScript Interface

A JavaScript interface is defined ease application development and to abstract the transport protocol variants defined in this specification. Please find the architectural overview in**Error! Reference source not found.** Figure 5 below.

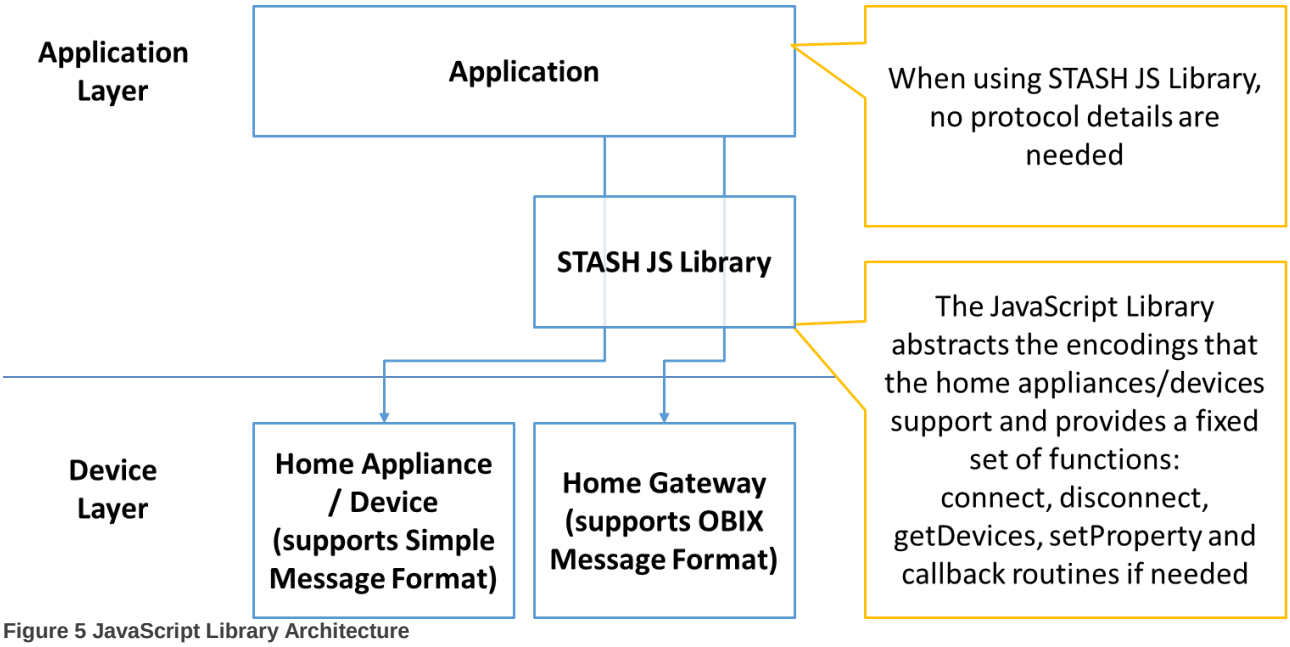

Figure 5 JavaScript Library Architecture

In the following public methods, properties and classes are defined, for examples see Annex G. STASH JavaScript Library Example.

4.1. Main methods and properties

Methods:

- addEndpoint(name, endpointAddress): creates a new endpoint with the given name and address, does return the endpoint object
- getEndpoints(): returns an array of all endpoint objects
- getEndpoint(name): returns the endpoint object with the given name or null
- removeEndpoint(name): removes the endpoint object with the given name if it does exist
- getDevices(): returns an array of device objects from all endpoints
- discoverEndpoints(): returns a list of discovered endpoints, these can be added to the endpoint list with addEndpoint

Properties:

- version: String which contains the version of the current library
- debug: Boolean which can be set to "true" to debug the library, defaults to false

4.2. Endpoint object methods and properties

Methods:

- connect(): connects to this endpoint
- disconnect(): disconnects from this endpoint
- poll(): retrieves the device list if polling is supported
- getDevices(): returns an array of device objects
- setProperty(name, propName, value): sets the property with name "propName" of a device with name "name" to the value specified in "value"

4.3. Device object methods and properties

Methods:

- setProperty(propName, value): sets the value of the property specified by "propName" to the given

Properties:

- name: String which denotes the name (unique attribute) of the device, needs to be set
- is: String which denotes the type of the device as specified in the data model (e.g. "sta:AirConditioner"), needs to be set
- location: String which denotes the location, can be empty
- properties: Array of Property objects (can be of String, Int, Bool, Real subclass)

4.4. Property object properties

Properties:

- name: String which denotes the name (unique attribute)
- value: Object which denotes the value
- display: if a display is set a String of possible displays for this attributes (especially important for integer values)
- writable: Boolean which is set to true if the property can be set, false else
- min: Object which denotes the minimum possible value, is optional and therefore can be null
- max: Object which denotes the maximum possible value, is optional and therefore can be null
- unit: String which denotes the unit (e.g. "°C")

4.5. Device Classes

Following device classes are implemented:

- Device: this is the base device class
- AirConditioner: for the contract "sta:AirConditioner"
- Washer: for the contract "sta:Washer"
- Dryer: for the contract "sta:Dryer"
- WasherDryer: for the contract "sta:WasherDryer"
- Refrigerator: for the contract "sta:Refrigerator"
- Light: for the contract "sta:Light"
- Cleaner: for the contract “sta:Cleaner”

Annex A. Use Case: Discovery

On a television, a user may want to get notified from nearby home appliances/devices, to monitor and control them. To support those use cases, home appliances/devices should be able to be discovered first. For this version of specification, following discovery use cases can be supported:

- If a home appliance/device is connected to the network, it advertises itself to the nearby devices that it supports Smart TV Alliance Smart Home specification
- An application on a TV can search for Smart TV Alliance Smart Home specification compatible devices among nearby devices

Annex B. Use Case: Notification

On a television a user wants to get notified on important or urgent events from the home appliances/devices in the home. Following notification model is stated:

- A home appliance/device can emit notifications which are directly pushed to connected clients
- A home appliance/device can store notifications for an arbitrary timeframe to allow newly connecting clients to receive this notification
- A notification may contain following content:
    - Title
    - Message

While the application is running, incoming notifications can be displayed, which means this version of specification only supports in-app notifications.

Annex C. Use Case: Monitoring

In this section, it is described how an application can retrieve and fetch data from a home appliance/device. The client can use polling or, if the endpoint supports pushing updates, it can retrieve data continuously. Following steps need to be executed:

1. Open a WebSocket connection to the endpoint. The endpoint URLs can be discovered by the method in Section 2.2.
2. For the two protocols, there is a different handling of monitoring home appliances/devices:
    1. The Simple variant in Section 2.4 allows polling to retrieve the latest home appliance/device information
    2. The OBIX variant in Section 2.5 allows update messages to be pushed from the endpoint to the client once the state on the endpoint does change using the concept of watches
3. Close WebSocket connection if no other request will be made or if listening is no longer required. To prevent early closes by the default WebSocket server time (usually 30s), empty packets can be sent.

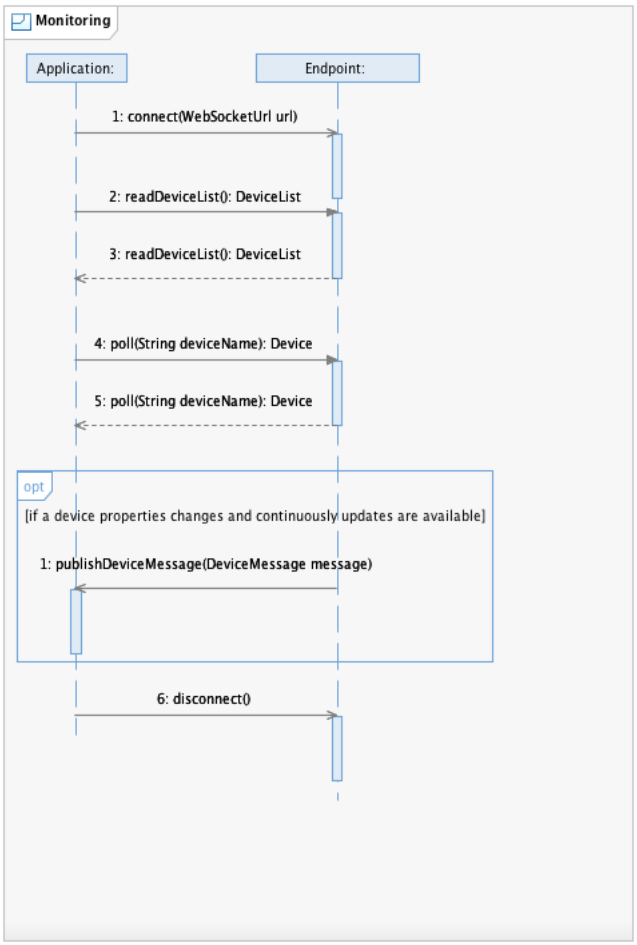

Figure 6 Monitoring Use Case Flow

Annex D. Use Case: Control

In this section it is specified how an application can control a home appliance/device. To send control messages for a home appliance/device to the server endpoint, providing access to this home appliance/device a client must execute following steps:

1. Open a WebSocket connection to the endpoint. The endpoint URLs can be discovered by the method in section 2.2.
2. To control a device, a JSON formatted message needs to be send adhering to the protocol variant the endpoint supports. For example, using the OBIX variant there must be an OBIX message sent over the WebSocket session containing the unique identifier of the device to control (like "/device/WashingMachineBasement") and the new state of a property or an operation to trigger an update of the home appliance/device.
3. Close WebSocket connection if no other request will be made.

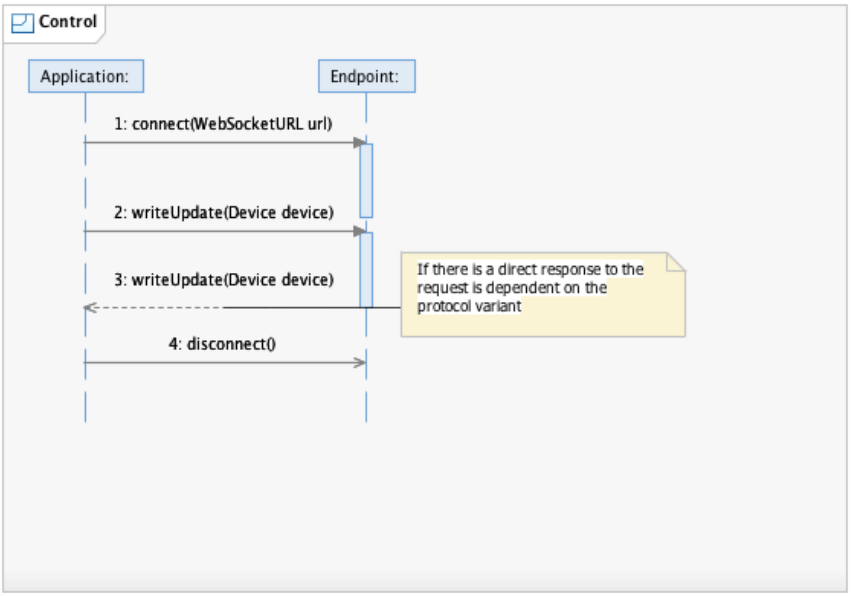

Figure 7 Control Use Case Flow

Note: Multiple monitoring and control messages can be sent during a single WebSocket connection.

Annex E. Message Format Examples: Simple Variant

Below a message exchange example is given:

Client	Server
Client initiates action on its own timing Connect to WebSocket server: wss://myhome?encoding=simple	→
X	*No response on successful connection*

Client	Server

Client initiates action on its own timing
{
"type": "request",
"tid" : 1,
"payload" : {
"getDevices": null
}
}
	→

←	*Server sends message in response to request from Client*
	{
	"type": "response",
	"tid" : 1,
	"payload" : {
	"devices": [
	{
	"deviceId": "abc123",
	"is": "AirConditioner",
	"vendorCode": "foo",
	"displayName": "My AirConditioner",
	"location": "Living Room"
	}
	]
	}
	}

Client	Server

Client initiates action on its own timing
{
"type": "request",
"tid" : 2,
"payload" : {
"deviceId": "abc123",
"getAttributes": [
"status",
"targetTemperature"
]
}
}
	→

←	*Server sends message in response to request from Client*
	{
	"type": "response",
	"tid" : 2,
	"payload" : {

```
    "deviceId": "abc123",
    "attributes": {
      "status": false,
      "targetTemperature": 28
    }
  }
}
```

Client	Server

Client initiates action on its own timing

```
{
  "type": "request",
  "tid" : 3,
  "payload" : {
    "deviceId": "abc123",
    "setAttributes": {
      "status": true
    }
  }
}
```

→

Server sends message in response to connection from Client

←

```
{
  "type": "response",
  "tid" : 3,
  "payload" : {
    "deviceId": "abc123",
    "accepted": [
      "status"
    ]
  }
}
```

Client	Server

Server sends message when it has some notifications

←

```
{
  "type": "notify",
  "tid" : 0,
  "payload" : {
    "deviceId": "abc123",
    "message": "Please check the air filter. It may need to be cleaned."
  }
}
```

Client	Server

Server sends message when it has some errors

←

```
{
  "type": "error",
  "tid" : 0,
  "payload" : {
    "deviceId": "abc123",
    "error": "Please make a phone call to technical support."
  }
}
```

Annex F. Message Format Examples: OBIX Variant

In this annex sample JSON messages are shown for all appliance types and also for all requests.

- Device Contract (sta:Device):

```
{
 "obix" : "obj",
 "href" : "/devices/WashingMachineBasement",
 "is" : "sta:Device",
 "location" : "basement",
 "children" : [ {
        "obix" : "bool",
        "name" : "operational",
        "value" : "true"
 } ]
};
```

- AirConditioner Contract (sta:AirConditioner):

```
{
      "obix" : "obj",
      "href" : "/devices/AirConditionerLivingRoom",
      "is" : "sta:AirConditioner",
      "location" : "living room",
      "children" : [
                  {
                        "obix" : "int",
                        "name" : "operationMode",
                        "val" : 1,
                        "display" : "Something:Automatic:Cooling:Heating:Drying:Ventilating"
                  }, {
                        "obix" : "int",
                        "name" : "fanSpeed",
                        "val" : 1,
                        "display" : "Weak:Normal:String"
                  }, {
                        "obix" : "real",
                        "name" : "roomTemperature",
                        "val" : 25.8,
                        "unit" : "obix:units/celsius",
                        "min" : -127.0,
                        "max" : 125.0
                  }, {
                        "obix" : "int",
                        "name" : "targetTemperature",
                        "val" : 22,
                        "min" : 0,
                        "max" : 50,
                        "unit" : "obix:units/celsius"
                  }, {
                        "obix" : "bool",
                        "name" : "status",
                        "val" : true
                  }, {
                        "obix" : "int",
                        "name" : "powerOffTime",
                        "val" : 2230,
                        "min" : 0,
                        "max" : 2359
                  } ]
}
```

- Refrigerator Contract (sta:Refrigerator):

```
{
   "obix" : "obj",
   "href" : "/devices/RefrigeratorKitchen",
   "is" : "sta:Refrigerator",
```

```
   "location" : "kitchen",
   "children" : [
         {
            "obix" : "real",
            "name" : "targetTemperature",
            "val" : 2.50,
                "min" : 0.0,
            "max" : 10.0,
            "unit" : "obix:units/celsius"
         }, {
            "obix" : "int",
            "name" : "compartment1",
            "val" : -18,
                 "min" : -50,
                 "max" : 50,
            "unit" : "obix:units/celsius"
         }, {
            "obix" : "int",
            "name" : "mode",
            "val" : 0,
            "display" : "Turned off:Power mode:Standby:Eco mode"
         }, {
            "obix" : "bool",
            "name" : "iceMakerEnabled",
            "val" : false,
         }, {
            "obix" : "int",
            "name" : "iceMakerStatus",
            "val" : 1,
            "display" : "No water in tank:Water in tank"
         }, {
            "obix" : "int",
            "name" : "mode",
            "val" : 0,
            "display" : "Turned off:Power mode:Standby:Eco mode"
         } ]
}
```


- Washer Contract (sta:Washer):


```
{
 "obix" : "obj",
 "href" : "/devices/WashingMachineBasement",
 "is" : "sta:Washer",
 "location" : "basement",
 "children" : [
                {
                        "obix" : "int",
                        "name" : "powerState",
                        "val" : 1,
                        "display" : "Off:On:Standby"
                }, {
                        "obix" : "int",
                        "name" : "washingStage",
                        "val" : 0,
                        "display" : "None:Washing:Rinsing:Spinning"
                }, {
                        "obix" : "int",
                        "name" : "operationState",
                        "val" : 0,
                        "display" : "Idle:Suspended:Finished"
                }, {
                        "obix" : "int",
                        "name" : "washingProgram",
                        "val" : 0,
                        "display" : "None:Wool:Jeans:Colored:Hot wash"
                }, {
                        "obix" : "int",
                        "name" : "spinSpeed",
                        "val" : 0,
                        "display" : "0:800:900:1000:1100:1200:1300:1400"
                }, {
                        "obix" : "int",
```

```
                "name" : "remainingTime",
                "val" : 0,
                "min" : 0,
                "max" : 2359
        } ]
}
```

- Dryer Contract (sta:Dryer):

```
{
 "obix" : "obj",
 "href" : "/devices/DryerBasement",
 "is" : "sta:Dryer",
 "location" : "basement",
 "children" : [
        {
                "obix" : "int",
                "name" : "powerState",
                "val" : 1,
                "display" : "Off:On:Standby"
        }, {
                "obix" : "int",
                "name" : "operationState",
                "val" : 2,
                "display" : "Idle:Suspended:Drying in progress:Finished"
        }, {
                "obix" : "int",
                "name" : "remainingTime",
                "val" : 40,
                "min" : 0,
                "max" : 2359
        } ]
}
```

- Cleaner Contract (sta:Cleaner):

```
{
 "obix" : "obj",
 "href" : "/devices/CleanerBasement",
 "is" : "sta:Cleaner",
 "location" : "basement",
 "children" : [
        {
                "obix" : "int",
                "name" : "operationState",
                "val" : 2,
                 “display” : “Start:Homing:Stop:Charging”
        }, {
                "obix" : "int",
                "name" : "batteryState",
                "val" : 1,
                "display" : "Low:Middle:High"
        }, {
                "obix" : "int",
                "name" : "cleaningMode",
                "val" : 0,
                "display" : "Manual:Spot:ZigZag:CellByCell"
        }]
}
```

Following an exemplary message flow is denoted:

Client	**Server**
Client initiates action on its own timing Connect to WebSocket server: wss://myhome?encoding=obix	→

←	*Server sends message in response to connection from Client*
	Returns the Lobby:
	{
	"obix" : "obj",
	"is" : "obix:Lobby",
	"children" : [{
	"obix" : "ref",
	"name" : "about",
	"is" : "obix:About",
	“href” : “/about”
	}, {
	"obix" : "op",
	"name" : "batch",
	"in" : "obix:BatchIn",
	"out" : "obix:BatchOut"
	}, {
	"obix" : "ref",
	"name" : "watchService",
	"is" : "obix:WatchService"
	“href” : “/watchService”
	}, {
	"obix" : "ref",
	"name" : "device",
	“href” : “/device”,
	"is" : "gateway:Device"
	}]
	}

Client	**Server**

Client reads an about object:
{
"obix" : "obj",
“is” : “obix:Read",
“rid” : 1,
“href" : “/about“
}
	→

←	*Server sends message in response to read request from Client*
	{
	“obix” : “obj”,
	“is” : “obix:Response”,
	“rid” : 1, “children” : [{
	"obix" : "obj",
	"href" : "/about",
	"name" : "about",
	"children" : [{
	"obix" : "str",
	"name" : "stashVersion",
	"val" : "1.0“
	},{
	"obix" : "str",
	"name" : "deviceId",
	"val" : "uniqueId”
	},{
	"obix" : "str",
	"name" : "vendorCode",
	"val" : "uniqueVendorCode“
	},{
	"obix" : "str",
	"name" : "productName",
	"val" : "productName”
	},{
	"obix" : "str",
	"name" : "productVersion",
	"val" : "productVersion”
	},{
	"obix" : "str",
	"name" : "productContract“,
	"val" : "sta:Washer”

<table>
<tr><td></td><td>

<pre>
        },{
                "obix" : "uri",
                "name" : "supportUrl",
                "val" : "http://www.smarttv-alliance.org“
        }]
 }]
}
</pre>

</td><td></td></tr>
<tr><td colspan="2">

Client reads a device object

<pre>
{
  "obix" : "obj",
  “is" : “obix:Read",
  “rid” : 2,
  “href” : “/device”
}
</pre>

</td><td>→</td></tr>
<tr><td>←</td><td colspan="2">

Server sends message in response to read request from Client

<pre>
{
 "obix" : "obj",
 "is" : "obix:Response",
 "rid" : "1",
 "children" : [{
        "obix" : "list",
        "of" : "obj",
        "name" : "device",
        “href” : “/device”
        "children" : [{
                "is" : "sta:Washer",
                "location" : "home",
                "name" : “washer",
                “href” : “/device/washer”
                "children" : [{
                        "name" : "status",
                        "value" : false,
                        "writeable" : false,
                        "display" : "",
                        "obix" : "bool"
                }, {
                        "name" : "power",
                        "value" : 0.0,
                        "writeable" : false,
                        "display" : null,
                        "obix" : "real"
                }, {
                        "name" : "energy",
                        "value" : 0.0,
                        "writeable" : false,
                        "display" : null,
                        "obix" : "real"
                }, {
                        "name" : "error",
                        "value" : "",
                        "writeable" : false,
                        "display" : null,
                        "obix" : "str"
                }, {
                        "name" : "powerState",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "Off:On:Standby",
                        "obix" : "int"
                }, {
                        "name" : "stage",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "None:Washing:Rinsing:Spinning",
                        "obix" : "int"
                }, {
                        "name" : "operationState",
                        "value" : 0,
</pre>

</td></tr>
</table>

<table>
<tr><td></td><td>

<pre>
                        "writeable" : true,
                        "display" : "Idle:Progress:Suspended:Finished",
                        "obix" : "int"
                }, {
                        "name" : "washProgram",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "None:Wool:Jeans:Colored:Hot wash",
                        "obix" : "int"
                }, {
                        "name" : "spinSpeed",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "0:800:900:1000:1100:1200:1300:1400",
                        "obix" : "int"
                }, {
                        "name" : "remainingTime",
                        "value" : 0,
                        "writeable" : false,
                        "display" : null,
                        "obix" : "int"
                }]
        }]
 }]
</pre>

</td><td></td></tr>
</table>

<table>
<tr><th>Client</th><th>Server</th></tr>
<tr><td colspan="2"><u>Client creates a watch:</u>

<pre>
{
 "obix" : "obj",
 "is" : "obix:Invoke",
 "rid" : 1,
 "href" : "/watchService/make"
}
</pre>

</td><td>→</td></tr>
<tr><td>←</td><td colspan="2"><u>Server sends message in response to watch request from Client</u>

<pre>
{
 "obix" : "obj",
 "is" : "obix:Response",
 "rid" : 1,
 "children" : [{
        "obix" : "obj",
        "href" : "/watch/1",
        "is" : "obix:Watch"
 }]
}
</pre>

</td></tr>
<tr><td colspan="2"><u>Client adds /device to the watch:</u>

<pre>
{
 "obix" : "obj",
 "is" : "obix:Invoke",
 "href" : "/watch/1/add,
 "rid" : 2,
 "children" : [{
        "obix" : "obj",
        "is" : "obix:WatchIn",
        "children" : [{
                "obix" : "list",
                "names" : "hrefs",
                "children" : [{
                        "obix" : "uri",
                        "val" : "/device/"
                }]
        }]
 }]
}
</pre>

</td><td>→</td></tr>
</table>

←	*Server sends message in response to watch request from Client*
	
	{
	"obix" : "obj",
	"is" : "obix:Response",
	"rid" : "1",
	"children" : [{
	“obix” : “obj”,
	“is” : “obix:WatchOut”,
	“children” : [{
	"obix" : "list",
	"of" : "obj",
	"name" : "device",
	“href” : “/device”
	"children" : [{
	"is" : "sta:Washer",
	"location" : "home",
	"name" : "washer",
	“href” : “/device/washer”
	"children" : [{
	"name" : "status",
	"value" : false,
	"writeable" : false,
	"display" : "",
	"obix" : "bool"
	}, {
	"name" : "power",
	"value" : 0.0,
	"writeable" : false,
	"display" : null,
	"obix" : "real"
	}, {
	"name" : "energy",
	"value" : 0.0,
	"writeable" : false,
	"display" : null,
	"obix" : "real"
	}, {
	"name" : "error",
	"value" : "",
	"writeable" : false,
	"display" : null,
	"obix" : "str"
	}, {
	"name" : "powerState",
	"value" : 0,
	"writeable" : true,
	"display" : "Off:On:Standby",
	"obix" : "int"
	}, {
	"name" : "stage",
	"value" : 0,
	"writeable" : true,
	"display" : "None:Washing:Rinsing:Spinning",
	"obix" : "int"
	}, {
	"name" : "operationState",
	"value" : 0,
	"writeable" : true,
	"display" : "Idle:Progress:Suspended:Finished",
	"obix" : "int"
	}, {
	"name" : "washProgram",
	"value" : 0,
	"writeable" : true,
	"display" : "None:Wool:Jeans:Colored:Hot wash",
	"obix" : "int"
	}, {
	"name" : "spinSpeed",
	"value" : 0,
	"writeable" : true,
	"display" : "0:800:900:1000:1100:1200:1300:1400",
	"obix" : "int"

<table>
<tr><td></td><td>

<pre>
                }, {
                        "name" : "remainingTime",
                        "value" : 0,
                        "writeable" : false,
                        "display" : null,
                        "obix" : "int"
                }]
        }]
    }]
 }]
}
</pre>

</td></tr>
</table>

<table>
<tr><td>Client</td><td>Server</td></tr>
<tr><td colspan="2"><u>Client sends property update of property "powerState" of device "Washer"</u>

<pre>
{
 "obix" : "obj",
 "is" : "obix:Write",
 “href” : “/device/washer”
 "rid" : "2",
 "children" : [{
        "obix" : "int",
        "name" : "washingProgram",
        "val" : "2"
 }]
}
</pre>

→</td></tr>
<tr><td>←</td><td><u>Server sends message response to the write request</u>

<pre>
{
 "obix" : "obj",
 "is" : "obix:Response",
 "children" : [{
        "is" : "sta:Washer",
        "location" : "home",
        "name" : "washer",
        "children" : [{
                "name" : "status",
                "value" : false,
                "writeable" : false,
                "obix" : "Bool"
        }, {
                "name" : "power",
                "value" : 0.0,
                "writeable" : false,
                "obix" : "real"
        }, {
                "name" : "energy",
                "value" : 0.0,
                "writeable" : false,
                "obix" : "real"
        }, {
                "name" : "error",
                "value" : "",
                "writeable" : false,
                "obix" : "str"
        }, {
                "name" : "powerState",
                "value" : 2,
                "writeable" : true,
                "display" : "Off:On:Standby",
                "obix" : "int"
        }, {
                "name" : "washingStage",
                "value" : 0,
                "writeable" : true,
                "display" : "None:Washing:Rinsing:Spinning",
                "obix" : "int"
        }, {
                "name" : "operationState",
                "value" : 0,
                "writeable" : true,
</pre>

</td></tr>
</table>

<table>
<tr><td></td><td>

<pre>
                "display" : "Idle:Progress:Suspended:Finished",
                "obix" : "int"
        }, {
                "name" : "washingProgram",
                "value" : 0,
                "writeable" : true,
                "display" : "None:Wool:Jeans:Colored:Hot wash",
                "obix" : "int"
        }, {
                "name" : "spinSpeed",
                "value" : 0,
                "writeable" : true,
                "display" : "0:800:900:1000:1100:1200:1300:1400",
                "obix" : "int"
        }, {
                "name" : "remainingTime",
                "value" : 0,
                "writeable" : false,
                "obix" : "int"
        }]
 }]
}
</pre>

</td></tr>
<tr><td>←</td><td>

Server asynchronously sends message once the device that was added in watch is updated

<pre>
{
 "obix" : "obj",
 "is" : "obix:Update",
 "children" : [{
        “obix” : “obj”,
        “is” : “obix:WatchOut”,
        “children” : [{
                "obix" : "list",
                "of" : "obj",
                "name" : "device",
                “href” : “/device”
                "children" : [{
                        "is" : "sta:Washer",
                        "location" : "home",
                        "name" : "washer",
                        "children" : [{
                                "name" : "status",
                                "value" : false,
                                "writeable" : false,
                                "obix" : "Bool"
                        }, {
                                "name" : "power",
                                "value" : 10.0,
                                "writeable" : false,
                                "obix" : "real"
                        }, {
                                "name" : "energy",
                                "value" : 0.0,
                                "writeable" : false,
                                "obix" : "real"
                        }, {
                                "name" : "error",
                                "value" : "",
                                "writeable" : false,
                                "obix" : "str"
                        }, {
                                "name" : "powerState",
                                "value" : 2,
                                "writeable" : true,
                                "display" : "Off:On:Standby",
                                "obix" : "int"
                        }, {
                                "name" : "washingStage",
                                "value" : 0,
                                "writeable" : true,
                                "display" : "None:Washing:Rinsing:Spinning",
                                "obix" : "int"
                        }, {
</pre>

</td></tr>
</table>

<table>
<tr><td></td><td>

<pre>
                        "name" : "operationState",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "Idle:Progress:Suspended:Finished",
                        "obix" : "int"
                    }, {
                        "name" : "washingProgram",
                        "value" : 2,
                        "writeable" : true,
                        "display" : "None:Wool:Jeans:Colored:Hot wash",
                        "obix" : "int"
                    }, {
                        "name" : "spinSpeed",
                        "value" : 0,
                        "writeable" : true,
                        "display" : "0:800:900:1000:1100:1200:1300:1400",
                        "obix" : "int"
                    }, {
                        "name" : "remainingTime",
                        "value" : 0,
                        "writeable" : false,
                        "obix" : "int"
                    }]
                }]
            }]
        }]
    }
</pre>

</td></tr>
</table>

<table>
<tr><th>Client</th><th>Server</th></tr>
<tr><td colspan="2"><u>Client sends property update of read-only property "powerState"</u>

<pre>
{
  "obix" : "obj",
  "is" : "obix:Write",
  “href” : “/device/washer”
  "rid" : "2",
  "children" : [{
        "obix" : "int",
        "name" : "powerState",
        "val" : "2"
  }]
}
</pre>

→</td></tr>
<tr><td>←</td><td><u>Server sends an error response to the write request</u>

<pre>
{
  "obix" : "obj",
  "is" : "obix:Response",
  “rid” : “5”
  "children" : [{
        "obix" : "PermissionErr",
  }]
}
</pre>

</td></tr>
</table>

<table>
<tr><th>Client</th><th>Server</th></tr>
<tr><td colspan="2"><u>Client sends message on its own timing</u>
Disconnect from wss://myhome/

→</td></tr>
<tr><td></td><td>Server disconnects from Client</td></tr>
</table>

Annex G. STASH JavaScript Library Example

Provide the application developers with a JavaScript API to write an application like below:

```
// load existing appliances/devices from local storage for all endpoints
var devices = stash.getDevices();

// manually add new endpoint
var epName = "Appliance1";
var epAddress = "wss://user:password@192.168.0.2";
var allEndpoints = stash.getEndpoints();
for ( var ep in allEndpoints) {
  if (allEndpoints[ep].name == epName) {
          log("Endpoint name already in use");
          return;
  } else if (allEndpoints[ep].endpointAddress == epAddress) {
          log("Endpoint with address: " + epAddress + " already added");
          return;
  }
}
stash.addEndpoint(epName, epAddress);
var endpoint = stash.getEndpoint(epName);
endpoint.ondeviceupdate = function() {
  // on device updates this function will get called
};

// load an appliance via the device name from the list of existing devices
var washingMachine = stash.getDevice("WashingMachineBasement");

// read / monitor appliance status and properties
if (washingMashine.getProperty("status") == 1
              && washingMachine.getProperty("stage") == 0) {
       // control appliance
       washingMachine.setProperty("washProgram ", 1);
}
```