IP Connected Device Protocol

Version 0.6

February 21, 2018

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Changes** |
| 0.1 | 04-Feb-2014 | Steven Perry | Initial draft |
| 0.2 | 17-Feb-2014 | Steven Perry | Corrected examples, expanded description of persistently connected devices, removed Profiles |
| 0.3 | 24-Oct-2014 | Steven Perry | Added onConnect event. Added requirements for ordered serialization of command attributes. Clarified use of txnid. |
| 0.4 | 21-Nov-2014 | Erik Larson | Clarified behavior when getting the value of a write-only parameter. |
| 0.5 | 24-Nov-2014 | Erik Larson | Added SetDeviceInfo command. |
| 0.6 | 21-Feb-2018 | Mike Finch | Clarified TLS, reporting behavior, added Leave command. |

# Table of Contents

Table of Contents 2

1.0 Introduction 4

1.1 Functional Concepts 4

1.2 Security Goals 5

1.3 Architecture Goals 5

2.0 Architecture 5

2.1 Protocol Components 5

2.2 Security Mechanisms 6

2.3 Parameters 6

2.4 Sessions 8

2.5 File Transfers 10

3.0 Actions 11

3.1 Report 11

3.2 Event 13

4.0 Commands 17

4.1 GetDeviceInfo 20

4.2 SetDeviceInfo 21

4.3 GetParameterValues 22

4.4 SetParameterValues 23

4.5 GetParameterInfo 25

4.6 GetReportConfiguration 27

4.7 SetReportConfiguration 28

4.8 GetEventConfiguration 30

4.9 SetEventConfiguration 32

4.10 Download 34

4.11 Upload 36

4.12 FactoryReset 36

4.13 Leave 37

# Introduction

The IP Connected Device Protocol (IPCD) is intended to support a variety of functions related to the monitoring and control of devices connected to the Internet of Things. It specifies two different methods for communication between devices and a centralized platform. On-demand devices use periodic HTTP POSTs to deliver periodic reports and important events and can receive commands from the platform in response. Persistently connected devices use WebSockets so that the platform can send commands as needed through their always-on connection. For security reasons, and to facilitate IP connected devices that are behind home Internet gateways performing Network Address Translation (NAT), the IPCD Protocol requires that all connections (on-demand or persistent) be initiated by the device.

## 1.1 Functional Concepts

The IPCD Protocol supports the following primary capabilities:

* Periodic reports of device state
* Device events
* Device control
* Diagnostics
* Embedded software management

### 1.1.1 Report of Device State

A primary purpose of the IPCD protocol is to allow devices to periodically report their state to the platform. Delivery of a report is triggered by the passage of time, not by any direct or indirect change in state of the device. The rate at which a device reports and the set of parameters it reports may be configured by the platform.

### 1.1.2 Device Events

In addition to periodic reporting, the IPCD protocol allows devices to notify the platform when specific parameters have changed or when important device lifecycle events have occurred. Lifecycle events include device boot, factory reset, software download complete or failed, and software update. Thresholds can be configured on individual parameters to control how and when these events are raised.

### 1.1.3 Device Control

The IPCD Protocol provides for commands to be sent from the platform to connected devices. These commands are either requests to set specific parameters on the device or are other protocol-level commands that get information from the device or set its configuration. The mechanics of how these commands are sent depends upon the connection model used by the device (on-demand or persistent). Commands may be sent to persistently connected devices asynchronously. On-Demand devices may receive commands in response to the Reports or Events they send.

### 1.1.4 Diagnostics

In addition to Reports and Events, Standard and Advanced IPCD Protocol devices support Commands such as GetParameterValues, which allows the platform to collect diagnostic information.

### 1.1.5 Software Management

IPCD Protocol devices may support the Download command to retrieve a new software image.

## 1.2 Security Goals

The IPCD Protocol is designed to maximize security around the transport of messages between the device and the platform. The security goals are as follows:

* Provide confidentiality for communications between the device and the platform
* Prevent tampering with the management and control functions of the device
* Allow appropriate authentication of the device to the platform

## 1.3 Architecture Goals

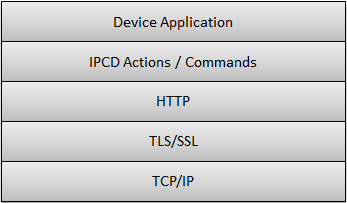
The IPCD Protocol is designed to accommodate different types of IP connected devices with different capabilities and connection models. The protocol is intended to provide the following:

* Allow the device to initiate a secure on-demand or persistent connection to the platform
* Allow the device to periodically report all or a subset of parameter values
* Allow the platform to change the behavior or control the function of a device by setting parameter values
* Allow devices to send events based on thresholds for known parameters
* Allow the platform to control the reporting and eventing capabilities of devices
* Allow the platform to discover newly connected devices
* Allow devices to download and apply software updates

# 2.0 Architecture

## 2.1 Protocol Components

The IPCD Protocol is built on top of several standard protocols to form a stack. A brief description of each layer is provided below. Devices that implement the IPCD Protocol must adhere to the requirements of the underlying standard protocols.



|  |  |
| --- | --- |
| **Layer** | **Description** |
| Device Application | Application code running in the connected device. The application is not specifically defined by the IPCD Protocol. |
| IPCD Actions / Commands | IPCD Actions (Report and Event) and the Commands that can be sent as a response by the connected device platform, are defined by the IPCD Protocol and are defined in section 3.0. |
| HTTP | Hypertext Transfer Protocol 1.1 as defined in RFC 2616. |
| TLS/SSL | Internet transport layer security, TLS 1.2 as defined in RFC 5246 or later. |
| TCP/IP | Standard TCP/IP. |

## 2.2 Security Mechanisms

The IPCD Protocol is designed to allow a high degree of security. It is designed to provide confidentiality for transactions between the device and the platform and to prevent those transactions from being tampered with. It allows for various levels of authentication depending upon the capabilities of the device.

* TLS 1.2 and above is used to secure communication transport between the device and the platform which provides data confidentiality and message integrity and prevents eavesdropping and tampering.
* Client certificates may be used to further authenticate devices.

## 2.3 Parameters

Within the IPCD Protocol, each device represents its state using a set of device parameters. These parameters take the form of name-value pairs. Each device is allowed to specify its own parameters.

### 2.3.1 Parameter Names

The rules for valid parameter names are as follows:

* May contain only
  + Alphabetic characters [A-Z,a-z]
  + Numeric digits [0-9]
  + Underscore, ‘ \_’
  + Period, ‘.’
* Must begin with an alphabetic character
* Must be no longer than 64 characters

There is no limit to the number of parameters a device can provide.

### 2.3.2 Parameter Types

Parameter typing is supported using a subset of JSON types, which are derived from JavaScript as defined in the object literals sections of the ECMAScript Programming Language Standard, Third Edition.

These types are supported:

|  |  |
| --- | --- |
| **Type** | **Description** |
| string | A sequence of zero or more Unicode characters with backslash escaping |
| number | Double precision floating point format |
| boolean | true or false |
| null | The null value |

The following specialized sub-types are also supported:

|  |  |  |
| --- | --- | --- |
| **Type** | **Sub-Type Of** | **Description** |
| enum | string | A specialization of string that accepts only specific statically defined values |

Unlike in JSON, array and object types are not directly supported as parameter types by the IPCD Protocol. JSON arrays and objects are used in JSON representation of the requests and responses for IPCD Commands.

Dates should be represented with the number type as Unix epoch time values (number of milliseconds since midnight, January 1, 1970, UTC).

The use of binary data as parameter values is discouraged, but when used, binary data should be restricted to less than 2 kilobytes and should be represented as a MIME-style Base64 encoded string.

### 2.3.3 Parameter Attributes

The IPCD Protocol provides for attributes on parameters that report whether or not they are write-only, read-only, or read-write enabled.

Parameters should be marked write-only in cases where they store secure information that should not be readable (such as a password). Read-write parameters are often used to directly change the state of a device or to otherwise control device behavior. Read-only parameters often relate static information about the device.

### 2.3.4 Required Parameters

A certain set of parameters is required to be supported so that devices may be properly identified within the protocol. These required parameters are read-only and, because they are included with every Action or Command, they are not included in the normal parameter tree.

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter Name** | **Type** | **Attributes** | **Description** |
| vendor | string | read-only | Human readable name of the device manufacturer or vendor. This may or may not be the same as the Vendor Key referenced in sections 3.1 and 3.2. |
| model | string | read-only | Human readable name of the model of the device. This may or may not be the same as the Model Key referenced in sections 3.1 and 3.2. |
| sn | string | read-only | Serial number of the device. Must be unique across any other devices with the same Vendor and Model string. A MAC address can be used as the value of the sn parameter. This value should be purely alpha-numeric with no whitespace, punctuation, or special characters. |
| ipcdver | string | read-only | Version of the IPCD Protocol supported by the device. |

### 2.3.5 Parameter Namespaces

Because there may be future efforts to standardize the naming of parameters that are common to many devices, it is recommended that vendor-specific parameters be prefixed with a model or vendor specific string to avoid any future collision of parameter names. A period can be used as a separator between the prefix and the parameter name.

For example, if a hypothetical vendor named Connected Things was adding IPCD Protocol support to a new device, it is recommended that they prefix all non-required parameter names with the string, ‘ct’. If they were to introduce a parameter named ‘status’, then the full parameter name would be ‘ct.status’.

## 2.4 Sessions

The IPCD Protocol provides two different interpretations of a session, one for on-demand connections and another for persistent connections.

### 2.4.1 On-Demand Sessions

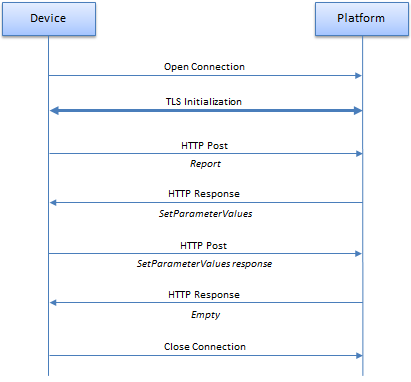
In the IPCD Protocol, an on-demand device is one that does not maintain a persistent connection to the platform. Instead, the device calls into the platform via TLS-wrapped HTTP POST requests to notify the platform of an important event or to deliver a periodic report and receive commands in response.

The benefit of using on-demand sessions is that it requires less sophistication on the part of the device and does not require an always-open socket on the platform. However, the cost of an on-demand device is a sub-optimal user experience.

Because most devices in the home will be behind a router performing NAT and may also be protected by a firewall, there is no way for the platform to reliably and securely reach out to the device when it has a command waiting. Instead, the platform must wait for an on-demand device to initiate a connection to the platform before any queued commands can be sent. The result is that the user must wait before seeing the effect of a control action against the device. As a result, it is recommended that all device vendors implement persistent sessions.

In an on-demand session, the device will call in with a report or event. After delivering the report or event, the device examines the response from the platform to determine if it contains a command. If a command is present in the response, the device sends a response to the command in a new POST request. The platform can then decide whether to continue the process by sending an additional command in the response or sending an empty response. When the device receives the empty response from the platform, it must close the connection.

A sequence diagram of an on-demand IPCD session is below. It illustrates a device calling in with a report followed by the platform sending a SetParameterValues command:



The Report action is performed by a device to provide periodic device status to the platform and poll for any queued commands. Devices that create on-demand sessions execute this action on a regular interval. The *interval* configuration setting (see SetReportConfiguration command in section 4.6.1) controls the interval by specifying the seconds between reports. This command also allows configuration of which parameters are reported on.

On-demand devices that are primarily sensors should use a default reporting rate of 15 minutes, or longer if the phenomenon that they measure changes slowly over time.

A device whose primary purpose is user-driven control should use a persistent connection rather than an on-demand connection for an improved user experience. However, if an on-demand connection is implemented, the device should use a default reporting rate of 1 minute.

On demand devices must support the SetReportConfiguration command.

### 2.4.2 Persistent Sessions

In the IPCD Protocol, a persistent or always-on device maintains a long-term connection to the platform. This is in contrast to on-demand devices, which periodically POST via standard HTTP.

For security reasons, and to avoid NAT traversal issues, the platform does not establish persistent connections to devices. The device must always initiate a persistent session.

Where on-demand devices use HTTP to transport messages to the platform, persistent devices use WebSockets (as defined by RFC 6455). The WebSocket protocol provides for a bi-directional, single-socket communication channel and was designed to interoperate with the existing HTTP infrastructure for authentication, proxy, and filtering. As such, WebSockets work over ports 80 and 443, which are often unblocked for outgoing TCP connections in residential Internet gateway devices.

IPCD requires the use of TLS wrapped WebSockets and devices must be provisioned with a connection URL (see section 4.1, GetDeviceInfo Command) that uses the wss protocol. The mechanism for provisioning the device with a persistent connection URL is not specified in the IPCD Protocol; most likely will be set in the device at the factory.

During the client WebSockets handshake, the relative path for the GET request must be

GET /ipcd-ws/<ipcdver>

Where <ipcdver> is the version of the IPCD protocol supported by the device.

The Host header must be set to the same value as in the wss connection URL. The Origin header should not be provided. The Sec-WebSocket-Protocol header value should be ipcd.

Here is an example:

GET /ipcd-ws/1.0 HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: uaKZJNkjan4kNKLnaw38ZO==

Sec-WebSocket-Protocol: ipcd

Sec-WebSocket-Version: 13

Once a WebSockets connection to the platform has been established, the device must send the Ping control frame every 30 seconds. On receipt of a Ping, the platform must send a corresponding Pong control frame. This mechanism allows both the device and the platform verify that the remote endpoint is responsive. It also works as a TCP keepalive.

Normal IPCD protocol messages are sent over the WebSocket as unmasked text messages.

Whereas on-demand devices send either a report or an event and then wait for commands from the platform, persistently connected devices may receive commands at any time. When a persistent device receives a command, it must hold any pending events or reports until it has sent the appropriate response to the platform. Other than this constraint, persistent devices may send events or reports at any time through the existing session.

## 2.5 File Transfers

IPCD devices may implement the ability to download and apply software updates through the use of the Download command. The device should be able to recognize if a file that it had downloaded constitutes a valid software image. The mechanism for doing so is not part of the IPCD Protocol. If the download constitutes a valid software image, the device should apply it.

All file transfers occur over TLS-wrapped HTTP using GET (this is true for persistently connected devices also). When a file transfer is initiated by the platform, the device is provided with the location of the file as a URL. The device then performs the download and, if it supports the onDownloadComplete and onDownloadFailed events, notifies the platform of download success or failure. If the download is a software image and the device applies it, it should send the onUpdate event before rebooting, then an onBoot and onConnected after applying the update.

# 3.0 Actions

## 3.1 Report

The Report action is used by the device to periodically send the value of its parameters to the platform. Section 4.6 describes the SetReportConfiguration command which can be used to control the periodic reporting interval and which parameters are reported upon. Section 4.5 describes the GetReportConfiguration command which can be used to retrieve the current report configuration from the device.

If a device does not support the SetReportConfiguration command, then it should report on all parameters (especially if it does not also support the GetParameterValues command). A device should submit a report soon after booting. If the device also support the Event action, then it should send the onBoot event first, then immediately send a report.

The exact mechanism for sending a report depends upon whether the device uses an on-demand connection or a persistent connection. On-demand devices will initiate a new session and, after negotiating TLS, issue the Report as an HTTP POST. Persistently connected devices can send a report through their existing asynchronous send channel.

### 3.1.1 Report from On-Demand Connected Devices

On-demand devices do not maintain a persistent connection. In order to send a report the device must create a new on-demand session by submitting the report as an HTTP POST.

The on-demand connection URL (described in section 4.1, GetDeviceInfo Command) is assumed to already be set on the device. The mechanism for provisioning the device with an on-demand connection URL is not specified in the IPCD Protocol; most likely will be set in the device at the factory.

The relative location used to POST the report is different for each device. The table below describes each part of the location.

/ipcd/<ipcdver>/report/<vendor\_key>/<model\_key>/<sn>

|  |  |
| --- | --- |
| **Part** | **Description** |
| ipcd | Path indicating that this is an IPCD connection |
| <ipcdver> | Version of the IPCD protocol supported by the device |
| report | Path component indicating a Report action |
| <vendor\_key> | The vendor string as defined in Section 2.3.4 with any whitespace, punctuation, and special characters removed. |
| <model\_key> | The model string as defined in Section 2.3.4 with any whitespace, punctuation, and special characters removed. |
| <sn> | The serial number as defined in Section 2.3.4. Must be unique across any other devices with the same Vendor and Model string. A MAC address can be used as the value of the sn parameter. This value should be purely alphanumeric with no whitespace, punctuation, or special characters. |

The body of the report is a JSON object with two child objects, device and report.

The device child object identifies the device and protocol version. It has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| vendor | Yes | string | Vendor parameter. |
| model | Yes | string | Model parameter. |
| sn | Yes | string | Serial number parameter. |
| ipcdver | Yes | string | IPCD Protocol version parameter. |

The report child object is a map of parameter name-value pairs.

The application/json Content -Type should be used for the initial report POST and for each response to any subsequent commands sent by the platform. All POSTs must set the Content-Length header to the correct value by inspecting the composed POST body prior to sending.

While the example below shows pretty-printed JSON, devices should consider sending a more concise representation where unnecessary whitespace is removed.

POST /ipcd/1.0/report/BlackBox/Multisensor2/00049B3C7A05 HTTP/1.1  
Content-Type: application/json  
Content-Length: 247  
{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" : "1.0"

},

"report" : {

"ms2.temperature" : 22.8,

"ms2.humidity" : 34.2,

"ms2.luminosity" : 10270

}

}

In response to the POST of a report, the device should expect the platform to send zero or more queued commands (described in Section 4.0 below), followed by an empty POST. The on-demand device should close the connection on receipt of the empty POST from the platform.

### 3.1.2 Report from Persistently Connected Devices

Persistently connected devices do not generally need to send periodic reports, since the platform can get the values of all parameters from the device as needed. However, should a persistently connected device need to send a periodic report, it sends the report as a UTF-8 encoded JSON object in a single unmasked text message. The JSON object has two child objects, device and report.

The device child object identifies the device and protocol version. It has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| vendor | Yes | string | Vendor parameter. |
| model | Yes | string | Model parameter. |
| sn | Yes | string | Serial number parameter. |
| ipcdver | Yes | string | IPCD Protocol version parameter. |

The report child object is a map of parameter name-value pairs.

While the example below shows pretty-printed JSON, devices should consider sending a more concise representation where unnecessary whitespace is removed.

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" : "1.0"

},

"report" : {

"ms2.temperature" : 22.8,

"ms2.humidity" : 34.2,

"ms2.luminosity" : 10270

}

}

## 3.2 Event

The Event action is used by the device to notify the platform when important events occur in its lifecycle or to inform the platform of changes to parameters of interest. Section 4.8 describes the SetEventConfiguration command that can be used to control which lifecycle events will result in notifications and to configure the thresholds used to generate parameter value change events. Section 4.7 describes the GetEventConfiguration command that can be used to retrieve the current event and notification configuration from the device.

If a device does not support the SetEventConfiguration command, then it should send notifications on all lifecycle events supported by the device (onBoot, onDownloadComplete, onDownloadFailed, onUpdate, and onFactoryReset). A device should send at least the onBoot event immediately after booting.

The exact mechanism for sending an event notification depends upon whether the device uses an on-demand connection or a persistent connection. On-demand devices will initiate a new session and, after negotiating TLS, issue the Event as an HTTP POST. Persistently connected devices can send events through their existing asynchronous send channel.

### 3.2.1 Event from On-Demand Connected Devices

On-demand devices do not maintain a persistent connection. In order to send an event, the device must create a new on-demand session by submitting the event as an HTTP POST.

The on-demand connection URL (described in section 4.1, GetDeviceInfo Command) is assumed to already be set on the device. The mechanism for provisioning the device with an on-demand connection URL is not specified in the IPCD Protocol; most likely will be set in the device at the factory.

The relative location used to POST the events is different for each device. The table below describes each part of the location.

/ipcd/<ipcdver>/event/<vendor\_key>/<model\_key>/<sn>

|  |  |
| --- | --- |
| **Part** | **Description** |
| ipcd | Path indicating that this is an IPCD connection |
| <ipcdver> | Version of the IPCD protocol supported by the device |
| event | Path component indicating an Event action |
| <vendor\_key> | The vendor string as defined in Section 2.3.4 with any whitespace, punctuation, and special characters removed. |
| <model\_key> | The model string as defined in Section 2.3.4 with any whitespace, punctuation, and special characters removed. |
| <sn> | The serial number as defined in Section 2.3.4. Must be unique across any other devices with the same Vendor and Model string. A MAC address can be used as the value of the sn parameter. This value should be purely alphanumeric with no whitespace, punctuation, or special characters. |

The body of the event post is a JSON object with three properties, device, events, and valueChanges.

The device child object identifies the device and protocol version. It has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| vendor | Yes | string | Vendor parameter. |
| model | Yes | string | Model parameter. |
| sn | Yes | string | Serial number parameter. |
| ipcdver | Yes | string | IPCD Protocol version parameter. |

The events property is an array of event names (multiple events can be delivered in a single notification).

The valueChanges property is an array of ValueChange objects that have the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| parameter | Yes | string | Parameter name |
| value | Yes | string | New parameter value. The type will vary by parameter but should agree with what is returned for that parameter from GetParameterInfo. |
| thresholdRule | No | string | Threshold rule as defined in GetEventConfiguration (onChange, onChangeBy, onEquals, onLessThan, onGreaterThan) |
| thresholdValue | No | <type> | The type will vary by parameter and threshold, but should agree with what is returned for that threshold from GetEventConfiguration |

The valueChanges property should be set to the empty array if no onValueChange event was delivered with this notification.

The application/json Content -Type should be used for the initial event POST and for each response to any subsequent commands sent by the platform. All POSTs must set the Content-Length header to the correct value by inspecting the composed POST body prior to sending.

While the example below shows pretty-printed JSON, devices should consider sending a more concise representation where unnecessary whitespace is removed.

POST /event/BlackBox/Multisensor2/00049B3C7A05 HTTP/1.1  
Content-Type: application/json  
Content-Length: 338  
{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"events" : ["onValueChange"],

"valueChanges" : [

{ "parameter" : "ms2.temperature",

"value" : 22.0,

"thresholdRule" : "onGreaterThan",

"thresholdValue" : 20

}

]

}

In response to the POST of a report, the device should expect the platform to send zero or more queued commands (described in Section 4.0 below), followed by an empty POST. The on-demand device should close the connection on receipt of the empty POST from the platform.

### 3.2.2 Event from Persistently Connected Devices

Persistently connected devices should prefer to send events over reports. When a persistently connected device needs to send an event, it does so as a UTF-8 encoded JSON object in a single unmasked text message. The JSON object must have the device, events, and valueChanges properties.

The device child object identifies the device and protocol version. It has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| vendor | Yes | string | Vendor parameter. |
| model | Yes | string | Model parameter. |
| sn | Yes | string | Serial number parameter. |
| ipcdver | Yes | string | IPCD Protocol version parameter. |

The events property is an array of event names (multiple events can be delivered in a single notification).

The valueChanges property is an array of ValueChange objects that have the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| parameter | Yes | string | Parameter name |
| value | Yes | string | New parameter value. The type will vary by parameter but should agree with what is returned for that parameter from GetParameterInfo. |
| thresholdRule | No | string | Threshold rule as defined in GetEventConfiguration (onChange, onChangeBy, onEquals, onLessThan, onGreaterThan) |
| thresholdValue | No | <type> | The type will vary by parameter and threshold, but should agree with what is returned for that threshold from GetEventConfiguration |

The valueChanges property should be set to the empty array if no onValueChange event was delivered with this notification.

While the example below shows pretty-printed JSON, devices should consider sending a more concise representation where unnecessary whitespace is removed.

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"events" : ["onValueChange"],

"valueChanges" : [

{ "parameter" : "ms2.temperature",

"value" : 22.0,

"thresholdRule" : "onGreaterThan",

"thresholdValue" : 20

}

]

}

# 4.0 Commands

Device agents (clients) send unsolicited Report and Event messages the IPCD server. The IPCD server can send unsolicited Commands to device agents who must in turn send the appropriate response.

All command requests and their responses are represented as JSON-encoded objects. The IPCD Protocol defines a set of 11 core commands. The Upload command is currently reserved for future use. The Download command is optional. Devices must implement all other commands.

### 4.0.1 Command Request Objects

Requests must always contain an attribute called command that specifies the command to call. Any additional attributes on the request object represent additional input parameters for the command. If a device receives a command with an attribute that it does not understand, it should return an error.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | Vendor parameter. |
| txnid | No | String | An arbitrary "transaction id". If the server includes this attribute in a command, then the agent must return this value in the request encapsulated inside the response |
| \* | Command-specific | string | Specific commands may require specific additional attributes. See the command reference below |

Below is an example request object for the GetDeviceInfo command, which takes no additional parameters:

{

"command" : "GetDeviceInfo"

}

With an optional transaction id:

{

"command" : "GetDeviceInfo",

"txnid" : "ab72230e2455"

}

Below is an example request object for the GetParameterValues commands, which takes a list of parameters whose values should be returned as an additional input:

{  
 "command" : "GetParameterValues",  
 "parameters" : ["ms2.temperature","ms2.humidity","ms2.luminosity"]  
}

The JSON specification describes JSON objects as consisting of an unordered set of attributes. Some devices agent implementations, especially those that execute in memory-constrained environments, may choose not to implement a full JSON parser. To make it easier for devices to consume commands, IPCD requires that server implementations serialize command message attributes in the order in which they appear in this document (i.e. the command attribute first, trxnid second, etc.)

### 4.0.2 Command Response Objects

The top-level command response object always contains four child objects: device, request, status, and response. The device child object contains basic device identification information. The request child object recapitulates the request that was made. The status child object indicates whether the command was successful and returns any warning or error messages. Finally, the response child object contains the body of the response to the command.

The example below illustrates a GetDeviceInfo response:

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0",

},

"request" : {

"command" : "GetDeviceInfo"

},

"status" : {

"result": "success"

},

"response" : {

"fwver" : "1.19-b118",

"connection" : "on-demand",

"connectUrl" : "https://ipthings.inetothings.net/ipcd",

"uptime" : 981452,

"actions" : ["Report","Event"],

"commands" : ["GetDeviceInfo", "GetParameterValues", "SetParameterValues",  
 "GetParameterInfo", "GetReportConfiguration",  
 "SetReportConfiguration", "GetEventConfiguration",  
 "SetEventConfiguration"]

}  
}

### 4.0.3 Command Response Device Object

The device child object has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| vendor | Yes | string | Vendor parameter. |
| model | Yes | string | Model parameter. |
| sn | Yes | string | Serial number parameter. |
| ipcdver | Yes | string | IPCD Protocol version parameter. |

### 4.0.4 Command Response Request Object

The request child object in the response recapitulates the original request with the command attribute and any other attributes required by the specific command called.

### 4.0.5 Command Response Status Object

The status child object has the following structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| result | Yes | string | "success" when request was successfully handled.  "warn" when the request was successfully handled but the device has result messages to return  "fail" when unexpected input was provided or a precondition of the command was not satisfied  "error" when an exception occurred in the device during execution |
| messages | No | array<string> | A list of messages explaining the warning, failure, or error |

The below example illustrates the status child object included in the response to a GetParameterValues command where the platform requested the value of an unknown parameter:

"status" : {

"result": "warn"

"messages": ["Parameter 'ns.state' is not defined"]

}

### 4.0.6 Command Response Object

Specific data returned as a result of a command is contained within the response child object. Different commands define specific response structures. See the command reference below for more information.

## 4.1 GetDeviceInfo

The GetDeviceInfo command instructs the device to return information about the identity and capabilities of the device.

### 4.1.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "GetDeviceInfo". |

{

"command" : "GetDeviceInfo"

}

### 4.1.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| fwver | Yes | string | Firmware version parameter. |
| connection | Yes | string | Connection type, one of ["on-demand", "persistent"]. |
| connectUrl | Yes | string | Service endpoint URL that device connects to. |
| uptime | No | number | Number of seconds since last device restart/reboot. |
| actions | Yes | array<string> | Array of IPCD Action names supported by this device. |
| commands | Yes | array<string> | Array of IPCD Command names supported by this device. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "GetDeviceInfo"

},

"status" : {

"result": "success"

},

"response" : {

"fwver" : "1.19-b118",

"connection" : "on\_demand",

"connectUrl" : "https://ipthings.inetothings.net/ipcd",

"uptime" : 981452,

"actions" : ["Report","Event"],

"commands" : ["GetDeviceInfo", "GetParameterValues", "SetParameterValues",

"GetParameterInfo", "GetReportConfiguration",

"SetReportConfiguration", "GetEventConfiguration",

"SetEventConfiguration"]

}

}

## 4.2 SetDeviceInfo

The SetDeviceInfo command instructs the device to write values to specific device info fields.  Multiple fields may be set with a single instance of the command, though transactionality at the device level is not guaranteed.  
If a device receives a request to set the value of a known field to an illegal value, an error should be returned.  An error should also be returned in the case that a device receives a request to set the value of an unknown field or a field that is read-only.   
If a device supports the SetDeviceInfo command, then any changes to the value of a device info field must be persisted in the device such that they survive a reboot.

At the current time, SetDeviceInfo supports only a single device field, connectUrl, which holds the URL of the IPCD server to connect to.

### 4.2.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "SetDeviceInfo". |
| values | Yes | object <string:value> | An array of field name and value pairs. |

{  
  "command" : "SetDeviceInfo",  
  "values" : {  
    "connectUrl" : "https://ipthings.inetothings.net/ipcd"  
  }  
}

### 4.2.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| <field names> | Yes | <command specific> | One attribute for each of the fields referenced in the values attribute of the request, showing the current value after the set. |

{  
  "device" : {  
    "vendor" : "BlackBox",  
    "model" : "Multisensor2",  
    "sn" : "00049B3C7A05",  
    "ipcdver" :"1.0"  
  },  
  "request" : {  
      "command" : "SetDeviceInfo",  
      "values" : {  
        "connectUrl" : "https://ipthings.inetothings.net/ipcd"  
      }  
  },  
  "status" : {  
    "result": "success"  
  },  
  "response" : {  
    "connectUrl" : "https://ipthings.inetothings.net/ipcd"  
  }  
}

## 4.3 GetParameterValues

The GetParameterValues command instructs the device to return name-value pairs for the requested parameters.

When no specific parameters are requested, the device should return the full list of parameters.

If a device receives a request to return the value of an unknown parameter, the device should return null for the value of that parameter rather than return an error in place of the GetParameterValues response.

Parameters which have the writeonly attribute will return null for the value rather than the actual value of the parameter.

### 4.3.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "GetParameterValues". |
| parameters | No | array<string> | An array of requested parameter names. If this attribute is omitted from the request, the device will return all known parameters. |

{

"command" : "GetParameterValues",

"parameters" : ["ms2.temperature","ms2.humidity","ms2.luminosity"]

}

### 4.3.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| <parameter names> | Yes | string | One attribute for each of the parameters referenced in the parameters attribute of the request. If the device does not understand one of the requested attributes, that attribute should be returned in the response with value null. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "GetParameterValues",

"parameters" : ["ms2.temperature","ms2.humidity","ms2.luminosity"]

},

"status" : {

"result": "success"

},

"response" : {

"ms2.temperature" : 22.8,

"ms2.humidity" : 34.2,

"ms2.luminosity" : 10270

}

}

## 4.4 SetParameterValues

The SetParameterValues command instructs the device to write values to specific parameters. Multiple parameters may be set with a single instance of the command, though transactionality at the device level is not guaranteed.

In general, the IPCD Protocol is forgiving for parameter gets but strict for parameter sets. If a device receives a request to set the value of a known parameter to an illegal value, an error should be returned. An error should also be returned in the case that a device receives a request to set the value of an unknown parameter.

If a device supports the SetParameterValues command, then any changes to the value of a parameter must be persisted in the device such that they survive a reboot.

### 4.4.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "SetParameterValues". |
| values | Yes | object <string:value> | An array of requested parameter names. If this attribute is omitted from the request, the device will return all known parameters. |

{

"command" : "SetParameterValues",

"values" : {

"ms2.lightsensorenabled" : true,

"ms2.temperaturescale" : "celsius"

}

}

### 4.4.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| <parameter names> | Yes | <command specific> | One attribute for each of the parameters referenced in the values attribute of the request, showing the current value after the set. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "SetParameterValues",

"values" : {

"ms2.lightsensorenabled" : true,

"ms2.temperaturescale" : "celcius"

}

},

"status" : {

"result": "success"

},

"response" : {

"ms2.lightsensorenabled" : true,

"ms2.temperaturescale" : "celcius"

}

}

## 4.5 GetParameterInfo

The GetParameterInfo command instructs the device return metadata about the parameters it supports, including each parameter's name, its type and its read/write attribute values.

In general, the IPCD Protocol is forgiving for parameter gets but strict for parameter sets. If a device receives a request to set the value of a known parameter to an illegal value, a failure should be returned. A failure should also be returned in the case that a device receives a request to set the value of an unknown parameter. An error should be returned if there is an exception while setting a known good parameter to a known good value.

### 4.5.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "GetParameterInfo". |

{  
 "command" : "GetParameterInfo"

}

### 4.5.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| <parameter names> | Yes | ParameterInfo | One attribute for each of the parameters supported by the device with a value of ParameterInfo type. |

#### ParameterInfo Type:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| type | Yes | string | One of ["string", "number", "boolean", "enum"] |
| enumvalues | No | array<string> | When type is "enum", the enumvalues attribute should be used to list the valid values for the attribute. |
| attrib | Yes | string | One of ["readonly", "readwrite", "writeonly"] |
| unit | No | string | A human-readable hint about the unit space of the parameter. Where possible, parameters should be defined in terms of SI units. |
| floor | No | number | A human-readable hint about the lower boundary of a numeric attribute |
| ceiling | No | number | A human-readable hunt about the upper boundary of a numeric attribute |
| description | No | string | Human readable description of the attribute |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "GetParameterInfo"

},

"status" : {

"result": "success"

},

"response" : {

"ms2.temperature" : {

"type": "number",

"attrib": "readonly",

"description" : "Temperature in degrees. Scale is either Celcius or Fahrenheit based on the value of the ms2.temperaturescale parameter"

},

"ms2.humidity" : {

"type": "number",

"attrib": "readonly",

"unit" : "percent",

"floor" : 0,

"ceiling" : 100,

"description" : "Relative humidity as a percentage from 0 to 100"

},

"ms2.luminosity" : {

"type": "number",

"attrib": "readonly",

"unit" : "lux",

"floor" : 1.0,

"ceiling" : 100000.0,

"description" : "Illuminance in lux"

},

"ms2.batterylevel" : {

"type": "number",

"attrib": "readonly",

"unit" : "percent",

"floor" : 1,

"ceiling" : 100,

"description" : "Approximate percentage of battery power remaining in units of 10"

},

"ms2.lightsensorenabled" : {

"type": "enum",

"enumvalues" : ["on", "off"],

"attrib": "readwrite",

"description" : "When on, the light sensor will be enabled and illuminance measurements will be reported. When off, ms2.luminosity will always return 0"

},

"ms2.temperaturescale" : {

"type": "enum",

"enumvalues" : ["celcius", "fahrenheit"],

"attrib": "readwrite",

"description" : "The scale of the ms2.temperature attribute"

}

}

}

## 4.6 GetReportConfiguration

The Report action is used by devices to periodically report their status. The GetReportConfiguration command allows the platform to retrieve the configuration of the device's reporting ability.

### 4.6.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "GetReportConfiguration". |

{  
 "command" : "GetReportConfiguration"  
}

### 4.6.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| interval | Yes | number | The interval, in seconds, between successful reports. |
| parameters | Yes | array<string> | An array of the names of all parameters that are included in reports |
| lastReportTime | No | number | Time of last report in Unix epoch time. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "GetReportConfiguration"

},

"status" : {

"result": "success"

},

"response" : {

"interval" : 900,

"parameters" : ["ms2.temperature", "ms2.humidity", "ms2.luminosity"],

"lastReportTime" : 1391552424924

}

}

## 4.7 SetReportConfiguration

The Report action is used by devices to periodically report their status. The SetReportConfiguration command allows the platform to change the configuration of the device's reporting ability.

The platform must set either the interval, or the set of parameters, or both in the request. If neither is provided, a failure should be returned from the device.

If a device supports the SetReportConfiguration command, then any changes to the configuration of the reporting capabilities of the device must be persisted in the device such that they survive a reboot.

The initial report following a SetReportConfiguration should be randomized on an interval of [0,interval).

### 4.7.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "SetReportConfiguration". |
| interval | No | number | The interval, in seconds, between successful reports. |
| parameters | No | array<string> | The list of parameters whose values should be included in each report. |

{

"command" : "SetReportConfiguration",

"interval" : 1800,

"parameters" : ["ms2.temperature", "ms2.humidity"]

}

### 4.7.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| interval | Yes | number | The interval, in seconds, between successful reports, after set. |
| parameters | Yes | array<string> | An array of the names of all parameters that are included in reports, after set. |
| lastreporttime | No | number | Time of last report in Unix epoch time. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0",

},

"request" : {

"command" : "SetReportConfiguration",

"interval" : 1800,

"parameters" : ["ms2.temperature", "ms2.humidity"]

},

"status" : {

"result": "success"

},

"response" : {

"interval" : 1800,

"parameters" : ["ms2.temperature", "ms2.humidity"],

"lastReportTime" : 1391552843924

}

}

## 4.8 GetEventConfiguration

The Event action is used by devices to report important device lifecycle events or important changes in device state. The GetEventConfiguration command allows the platform to retrieve the configuration of the device's event generation ability.

### 4.8.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "GetEventConfiguration". |

{  
 "command" : "GetEventConfiguration"  
}

### 4.8.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| supportedEvents | Yes | array<string> | An array of the names of events that are supported by this device.  Should include a subset of ["onBoot", "onDownloadComplete", "onDownloadFailed", "onUpdate", "onFactoryReset", "onValueChange"] |
| enabledEvents | Yes | array<string> | An array of the names of all events that are currently enabled on the device. |
| supportedValueChanges | No | map<parameter, array<string>> | A map of parameter names to an array of the value change sub-events supported by the parameter |
| enabledValueChanges | No | map<parameter, ValueChangeThreshold> | A map of parameter names to a ValueChangeThreshold object |

#### ValueChangeThreshold Type:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| onChange | No | boolean | Supported by any parameter type (except null). If set to true, any change in the parameter since the last value change event for this parameter will trigger a new value change event. |
| onChangeBy | No | number | Supported by number typed parameters. If the parameter value changes by the specified amount (positive or negative) since the last value change event for this parameter, a new value change event will be triggered. |
| onEquals | No | array<type> | Supported for any parameter type. If the parameter's new value is equal to any of the values in the array, then a new value change event will be generated. Values in the array must be of the same type as the parameter. |
| onLessThan | No | number | Supported for number type parameters. If the parameter's new value is less than the provided value, then a new value change event will be generated. |
| onGreaterThan | No | number | Supported for number type parameters. If the parameter's new value is greater than the provided value, then a new value change event will be generated. |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "GetEventConfiguration"

},

"status" : {

"result": "success"

},

"response" : {

"supportedEvents" : ["onBoot", "onDownloadComplete", "onDownloadFailed",

"onUpdate", "onFactoryReset", "onValueChange"],

"enabledEvents" : ["onBoot", "onUpdate", "onFactoryReset", "onValueChange"],

"supportedValueChanges" : {

"ms2.temperature" : ["onChange", "onChangeBy", "onEquals", "onLessThan",  
 "onGreaterThan"],

"ms2.humidity" : ["onChange", "onChangeBy", "onEquals", "onLessThan",  
 "onGreaterThan"],

"ms2.luminosity" : ["onChange", "onChangeBy", "onEquals", "onLessThan",   
 "onGreaterThan"],

"ms2.batterylevel" : ["onChange", "onChangeBy", "onEquals", "onLessThan",  
 "onGreaterThan"],

"ms2.lightsensorenabled" : ["onChange", "onEquals"],

"ms2.temperaturescale" : ["onChange", "onEquals"]

},

"enabledValueChanges" : {

"ms2.temperature" : {

"onLessThan" : 2,

"onGreaterThan" : 27

},

"ms2.batterylevel" : {

"onChangeBy" : 10,

"onLessThan" : 15

},

"ms2.lightsensorenabled" : {

"onChange" : true

}

}

}

}

## 4.9 SetEventConfiguration

The Event action is used by devices to report important device lifecycle events or important changes in device state. The SetEventConfiguration command allows the platform to configure how the device sends events.

If a device supports the SetEventConfiguration command, then any changes to the configuration of the eventing capabilities of the device must be persisted in the device such that they survive a reboot.

Value change events that are triggered by some change since a previous event may behave differently immediately after the device boots. When this happens, the device may take the initial state of the parameter after a boot as the previous comparison values, rather than the values at the time of the last value change event.

The platform must send either the enabledEvents, or the enabledValueChanges parameter, or both in the request. If the event configuration for a parameter is sent, it overrides the current configuration for that parameter, leaving the other configuration options the same as before the set. If neither enabledEvents nor enabledValueChanges are sent in the response, the device should return a failure.

If the request attempts to enable any event that is not supported by the device as reported in GetEventConfiguration, the device should return a failure. A failure should also be returned by the device if the request attempts to enable an unsupported value change threshold or a value change threshold for an unsupported parameter.

To remove a previously set threshold from a parameter that has other thresholds defined, provide a ValueChangeThreshold object that does not have the target threshold defined but restates the thresholds to preserve. To remove all thresholds from a parameter, provide an empty ValueChangeThreshold object as the value of the parameter in the enabledValueChanges map.

### 4.9.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "SetEventConfiguration". |

{

"command" : "SetEventConfiguration",

"enabledEvents" : ["onBoot", "onDownloadComplete", "onDownloadFailed", "onUpdate",

"onValueChange"],

"enabledValueChanges" : {

"ms2.temperature" : {

"onLessThan" : 5,

"onGreaterThan" : 23

},

"ms2.batterylevel" : {

"onChangeBy" : 10,

"onLessThan" : 15

}

}

}

### 4.9.2 Response

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| enabledEvents | Yes | array<string> | An array of the names of all events that are currently enabled on the device. |
| enabledValueChages | No | map<parameter, ValueChangeThreshold> | A map of parameter names to a ValueChangeThreshold object (see 4.7.2 for definition) |

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "SetEventConfiguration",

"enabledEvents" : ["onBoot", "onDownloadComplete", "onDownloadFailed",

"onUpdate","onValueChange"],

"enabledValueChanges" : {

"ms2.temperature" : {

"onLessThan" : 5,

"onGreaterThan" : 23

},

"ms2.batterylevel" : {

"onChangeBy" : 10,

"onLessThan" : 15

}

}

},

"status" : {

"result": "success"

},

"response" : {

"enabledEvents" : ["onBoot", "onDownloadComplete", "onDownloadFailed",

"onUpdate", "onValueChange"],

"enabledValueChanges" : {

"ms2.temperature" : {

"onLessThan" : 5,

"onGreaterThan" : 23

},

"ms2.batterylevel" : {

"onChangeBy" : 10,

"onLessThan" : 15

}

}

}

}

## 4.10 Download

The Download command is used by the platform to instruct the device to download a binary file. Version 1.0 of the IPCD Protocol specifies the use of the Download command as a way to trigger a software update on the device. Other uses of the Download command are not covered in this version of the protocol, but may be specified in a future version.

The Download command instructs the device to GET a binary image from the URL provided. The device must support the HTTPS protocol for downloads. In addition to supplying a URL, the platform can also pass an optional username and password, which should be used by the device in response to an HTTP Basic or Digest Auth challenge from the server hosting the URL. On receiving the Download command, the device should either start the download in another thread or immediately send the response before initiating the download.

Upon successful completion of the download, the device should send the onDownloadComplete event. If the download fails, the device should re-attempt the download after a suitable period of time (30 seconds to 1 minute). If three download attempts fail, then the device should send the onDownloadFailed event.

If the download is successful and determined to be a valid software image, the device should apply it and send the onUpdate event. If the device must reboot to complete the update, the onUpdate and onBoot events should be sent in the same Event action after the reboot.

The mechanism by which the platform obtains and distributes download URLs (and the optional username and password used to GET the URL) is not specified in the IPCD Protocol.

### 4.10.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "Download". |
| url | Yes | string | URL to the firmware image to download |
| username | No | string | Optional username to use if HTTP-Basic or HTTP Digest auth is required for the download URL |
| password | No | string | Optional password to use if HTTP-Basic or HTTP Digest auth is required for the download URL |

{  
 "command" : "Download"

"url" : "https://thingyverse.com/images/devices/BlackBox/ms2/1.20-b236.bin"  
}

### 4.10.2 Response

The response child object on a Download response should be empty.

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "Download",

"url" : "https://thingyverse.com/images/devices/BlackBox/ms2/1.20-b236.bin"

},

"status" : {

"result": "success"

},

"response" : {

}

}

## 4.11 Upload

The Upload command is not specified in version 1.0 of the IPCD Protocol. Use of this command is reserved for a future version.

## 4.12 FactoryReset

The FactoryReset command can be issued to a device in order to reset it to its default factory state. During a factory reset, the device should reset all report configuration and event configuration to default values. All parameters that have previously been set should also be reset to their factory default values.

Upon successful reset, the device should send the onReset event. If the device must reboot in order to complete the factory reset, it should send the onBoot and onReset events in the same Event action.

This requires that the network credentials (such as Wi-Fi SSID and passphrase) **must not be** erased from the device.

### 4.12.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "FactoryReset". |

{  
 "command" : "FactoryReset"

}

### 4.12.2 Response

The response child object on a FactoryReset response should be empty.

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "FactoryReset"

},

"status" : {

"result": "success"

},

"response" : {

}

}

## 4.13 Leave

The Leave command can be issued to a device in order to remove it from Iris and put it in a state where it can be commissioned again. During a leave, the device should reset all report configuration and event configuration to default values. All parameters that have previously been may be retained to allow the device to continue performing it’s normal functions as expected by the consumer.

The response must be sent before executing the command.

This requires that the network credentials **must be** erased from the device, the device should return to it’s pairing mode (Typically AP Mode or Bluetooth).

### 4.12.1 Request

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute** | **Required** | **Type** | **Description** |
| command | Yes | string | The command to send. Must be "Leave". |

{  
 "command" : "Leave"

}

### 4.12.2 Response

The response child object on a “Leave” response should be empty.

{

"device" : {

"vendor" : "BlackBox",

"model" : "Multisensor2",

"sn" : "00049B3C7A05",

"ipcdver" :"1.0"

},

"request" : {

"command" : "Leave"

},

"status" : {

"result": "success"

},

"response" : {

}

}