Advantech

Open RESTful API Specification

For Ubuntu

An.Xin

2023-5-17

Content

1 Web Service Uniform Interface 3
1.1 RESTful Web Service Uniform Interface 3
1.2 GET 3
1.3 PUT 3
1.4 PATCH 3
1.5 POST 4
1.6 Safe and Idempotent 4
1.7 HTTPS Response Status-Code 4
2 Restful API Definition 6
2.1 Log in/log out 6
2.1.1 Log in 6
2.1.2 Log out 7
2.2 Image Update 9
2.2.1 Image File Information Verification 9
2.2.2 Image Upload 10
2.2.3 Image Update 12
2.2.4 Image version information 13

1 Web Service Uniform Interface

1.1 RESTful Web Service Uniform Interface

In REST, resources (a resource is any information that you want to make available to others) are identified by unique URI. User agents (a.k.a. web browsers) only interact with resources using the prescribed HTTP verbs, The main verbs (GET, PUT, POST, and DELETE) are what we call the uniform interface. Resources can be read or written to using the HTTP verbs. GET to read resource, PUT is typically used for modifying an existing resource, POST indicates the desire to perform batch operation, DELETE indicates that a client wishes to delete a resource.

1.2 GET

The GET method means retreive whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

1.3 PUT

PUT can be used when the client is sending data to the the server and the client is determining the URI for the newly created resource.

> *The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.*

- client must supply the ID
- if the resource exists, ***replace*** it with the inbound data
- if it doesn't exist, **create** a new one (assuming you can do that)

1.4 PATCH

RFC5786 defines the new HTTP/1.1 [RFC2616] method, PATCH, which is used to apply partial modifications to a resource.
PATCH can be used when the client is sending one or more changes to be applied by the the server.

> *The PATCH method requests that a set of changes described in the request entity be applied*

to the resource identified by the Request-URI. The set of changes is represented in a format called a "patch document"...

The point is that PATCH is used to doing some kind of **'partial' update**.

1.5 POST

POST can be used when the client is sending data to the server and the server will decide the URI for the newly created resource.

"The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line."

This is what most of us think of when we talk about "**creating data**" on a web server.

1.6 Safe and Idempotent

Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.

	Safe?	**Idempotent?**
GET	Y	Y
POST	N	N
PATCH	N	N
PUT	N	Y
DELETE	N	Y

1.7 HTTPS Response Status-Code

[RFC2616] The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request.
The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- 1xx: Informational - Request received, continuing process

- 2xx: Success - The action was successfully received, understood, and accepted

- 3xx: Redirection - Further action must be taken in order to complete the request

- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled

- 5xx: Server Error - The server failed to fulfill an apparently valid request

The status codes supported by ADAM web server are described below.

Status-Code	Reason-Phrase	Conditions
200	OK	The request succeeded, and that the requested information is in the response. This is the most common status code to receive.
400	Bad Request	The request could not be understood by the server due to malformed syntax. ● The value in Content-length header does not match with the real data length. ● Invalid JOSN objects or format. ● Invalid multipart contents.
403	Forbidden	The server refuses to fulfill the request due to authentication error, such as invalid cookie.
404	Not Found	The requested resource (URI) does not exist on the server.
405	Method Not Allowed	The request method (POST or GET) is not allowed on the requested resource.
411	Length Required	The required Content-length header is missing.
500	Internal Server Error	Memory access errors occurred on the server when processed the file uploaded by client. Fail to reply the JSON-format message due to out of buffer size, or others.
503	Service Unavailable	The login list is full.

2 Restful API Definition

2.1 Log in/log out

2.1.1 Log in

/sys/log_in

Description	Log in the device for configuration or image updating.
URL Structure	**https://{ip address}/sys/log_in/**
HTTP Method	PUT：
GET	None

PUT	Request：**PUT** /sys/log_in
	[Example]:
	● Request: **PUT** /sys/log_in
	Content-type: application/json
	{
	"password":"00000000"
	}
	
	Response:
	{
	"sesion_id": "c9f4baf91d3e4ed7cfb18e598c5711f5",
	}
	[Example]:
	● Request: **PUT** /sys/log_in?timeout=xxxx
	Content-type: application/json
	{
	"password":"00000000"
	}
	
	Response:
	{
	"sesion_id":"c9f4baf91d3e4ed7cfb18e598c5711f5"
	}

POST	None
PATCH	None

- Resource value definitions:

Field	Abbreviation	Data Type	Property	Description
password	password	String	RW	User password
Session index	session_id	String	R	Session index for cookies

- Query param definitions:

Field	Abbreviation	Data Type	Property	Description
Timeout	timeout	String	W	Set timeout for current login session. xxxx：timeout value. unit: second. If timeout value is zero, then the session will never timeout util the logout operation is happened.

Remarks	Session_id will be used in cookie for GET/PUT/POST/PATCH method. Cookie: SID=c9f4baf91d3e4ed7cfb18e598c5711f5 Session timeout value means if there is no web access to webservice for such time period after login, the session will be timeout. The default value is 3600 seconds.

2.1.2 Log out

`/sys/log_out`

Description	Log out the device
URL Structure	**https://{ip address}/sys/log_out/**
HTTP Method	PUT
GET	None

PUT	- Request: **PUT** /sys/log_out
	Content-type: application/json
	
	Response:
	{
	"Success":"Log out "
	}

POST	None
PATCH	None

● Resource value definitions:	
Remarks	

2.2 Image Update

2.2.1 Image File Information Verification

/sys/file_verify

Description	Get image file information from user.
URL Structure	**https://{ip address}/sys/file_verify/**
HTTP Method	POST
GET	None
PUT	None

POST	Request：**POST** /sys/file_verify
	[Example]:
	● Request: **POST** /sys/file_verify
	Content-type: application/json
	
	Response:
	{
	"size":"5375",
	"count":"2"
	"detail": [
	{
	" name ":" manifest.xml",
	" size ":"5007",
	},
	{
	"name":" checksum.md5",
	"size":”368”,
	},
	]
	}

PATCH	None
● Resource value definitions:	
Remarks	This request must be executed before /sys/upload

2.2.2 Image Upload

/sys/upload

The standard way to upload files in a web application is to use a form with a special multipart/form-data encoding.
<RFC1521> In the case of multiple part entities, in which one or more different sets of data are combined in a single body, a "multipart" Content-Type field must appear in the entity's header. The body must then contain one or more "body parts," each preceded by an encapsulation boundary, and the last one followed by a closing boundary. Each part starts with an encapsulation boundary, and then contains a body part consisting of header area, a blank line, and a body area.
Each body part is preceded by an encapsulation boundary. The encapsulation boundary MUST NOT appear inside any of the encapsulated parts. Thus, it is crucial that the composing agent be able to choose and specify the unique boundary that will separate the parts.
Encapsulation boundaries must not appear within the encapsulations, and must be no longer than 70 characters, not counting the two leading hyphens.
The encapsulation boundary following the last body part is a distinguished delimiter that indicates that no further body parts will follow. Such a delimiter is identical to the previous delimiters, with the addition of two more hyphens at the end of the line:

--gc0p4Jq0M2Yt08jU534c0p--

<RFC2388> “Multipart/form-data” can be used for forms that are presented using representations other than HTML (spreadsheets, Portable Document Format, etc), and for transport using other means than electronic mail or HTTP. This document defines the representation of form values independently of the application for which it is used.
"Multipart/form-data" contains a series of parts. Each part is expected to contain a content-disposition header [RFC 2183] where the disposition type is "form-data", and where the disposition contains an (additional) parameter of "name", where the value of that parameter is the original field name in the form. For example, a part might contain a header:

Content-Disposition: form-data; name="user"

with the value corresponding to the entry of the "user" field.
Field names originally in non-ASCII character sets may be encoded within the value of the "name" parameter using the standard method described in RFC 2047.

Description	Log in the device for configuration or image updating.
URL Structure	**https://{ip address}/sys/upload/**
HTTP Method	POST：
GET	None

PUT	None
POST	Request：**POST** /sys/upload [Example]: ● Request: **POST** /sys/upload Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="advup.txt"; filename="advup.txt" Content-Type: <Content-Type header here> (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="advupdate.txt"; filename="advupdate.txt" Content-Type: text/plain (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="apps.tar.gz"; filename="apps.tar.gz" Content-Type: application/x-tar (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="checksum.md5"; filename="checksum.md5" Content-Type: <Content-Type header here> (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="flash.bin"; filename="flash.bin" Content-Type: application/octet-stream (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="fsl-imx8mq-xxxx.dtb"; filename="fsl-imx8mq-xxxx.dtb" Content-Type: <Content-Type header here> (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="Image"; filename="Image" Content-Type: <Content-Type header here> (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="manifest.xml"; filename="manifest.xml" Content-Type: text/xml (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="ramdisk.gz"; filename="ramdisk.gz"

	Content-Type: <Content-Type header here> (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Content-Disposition: form-data; name="rootfs.tar.gz"; filename="rootfs.tar.gz" Content-Type: application/x-tar (data) ----WebKitFormBoundary7MA4YWxkTrZu0gW Response: 200 OK
PATCH	None
Remarks	Must log_in first

2.2.3 Image Update

`/sys/update`

Description	Begin to update the files of the device.
URL Structure	**https://{ip address}/sys/update/**
HTTP Method	POST
GET	None
PUT	None

POST	Request：**POST** /sys/update
	[Example]:
	● Request: **POST** /sys/update
	Content-type: application/json
	
	Response:
	{
	"Notice":"updating"
	}

PATCH	None
Remarks	Must log in first

2.2.4 Image version information

/sys/version?filename=manifest.xml

/sys/version

Description	Show the version information of the update package for the device.
URL Structure	**https://{ip address}/sys/version?filename=manifest.xml** **https://{ip address}/sys/version**
HTTP Method	GET

GET	Request：**GET /s**ys/version?filename=manifest.xml
	[Example]:
	● Request:
	GET /sys/version?filename=manifest.xml
	Content-type: application/json
	
	● Request:
	GET /sys/version
	Content-type: application/json
	
	Response:
	{
	"Success":"Local version [3.0.1]"
	}

PUT	None
POST	None
PATCH	None
Remarks	