Paint Web Remote Control API

SINCE DAY ONE.

Http

Current API version: **1**.

Login

Test login session.

`HTTP GET /api/{version}/login`

REQUEST

```
empty
```

RESPONSE

```
empty
```

RESPONSE CODE

- 200 - OK

Create a login session, sign in.

`HTTP POST /api/{version}/login`

REQUEST

```
"password" : <string>
```

RESPONSE

```
empty
```

RESPONSE CODE

- 200 - OK
- 401 - Invalid password

Delete login session, sign out.

`HTTP DELETE /api/{version}/login`

REQUEST

```
"Password" : <string>
```

RESPONSE

`text/plain`

RESPONSE CODE

- 200 - OK
- 401 - Invalid password

Security

Return public key in base64 format.

```
HTTP GET /api/{version}/security/key
```

REQUEST

```
empty
```

RESPONSE

```
text/plain
```

RESPONSE CODE

- 200 - OK

Settings

Return setting of web page.

```
HTTP GET /api/{version}/setting
```

REQUEST

```
empty
```

RESPONSE

```
{
        "socketPort" : <int>
        "videoPort" : <int>
}
```

RESPONSE CODE

- 200 - OK

Clips

Return thumbnail of given clip.

```
HTTP GET
/api/{version}/clips/thumbnail?bank=<bankId>&index=<clipIndex>
```

REQUEST

```
empty
```

RESPONSE

```
image/png stream
```

RESPONSE CODE

- 200 - OK
- 400 - Clip not found or clip has no thumbnail

Tools

Return thumbnail of given tool.

```
HTTP GET
/api/{version}/tools/thumbnail?toolbar=<toolbarId>&index=<toolIndex>
```

REQUEST

```
empty
```

RESPONSE

```
image/png stream
```

RESPONSE CODE

- 200 - OK
- 400 - Tool not found

WebSocket connector

Connector provides methods that are used for communication with modules via websocket duplex connection.

Connector

Retrieving

Instance of connector may be retrieved in javascript either by explicit specification in `define()` method when javascript module is registered or by `require()` inside of already registered javascript module.
Example:

When js module is registered

```
define(["socket/connector"], function(connector) {
      connector.call(...);
});
```

Inside of registered js module

```
define(["require"], function(require) {
      var connector = require("socket/connector");
      connector.call(...);
});
```

Errors

Error is thrown when a method is called or error is returned via promise in following cases:

- target module not found
- target method not found
- target method expects different count of parameters or it is not possible to cast passed parameters to expected types

Methods

Methods `check()` and `call()` are asynchronous and return a promise that is notified when return value is known or error occurred.
Example:

```
connector.check("moduleName", function(response) {
      //process response
});

connector.call("moduleName", "methodName", ["param", 123])
.then(function(response) {
      //process response
}).catch(function(error) {
      //process error
});
```

Method `listen()` takes a callback function that is notified when target property is changed.
Example:

```
connector.listen("moduleName", "propertyName", function(data) {
      //process data
});
```

Check and return whether given module is registered or not.

```
check(string moduleName) : promise(boolean)
```

Call method and return response.
`moduleName` and `methodName` must be present or error is thrown and passed parameters must be the same as target method expects or error is thrown.

`moduleName <string>` - name of target module
`methodName <string>` - name of target method

`parameters <object/[objects]>` - single or multiple parameters (as array) passed to target method
`binaryData <object>` - binary data send to target method
`skipResponse <boolean>` - do not wait for response when true

```
call(moduleName, methodName, parameters, binaryData, skipResponse) :
promise(response)
```

Listen for changes of target property and receive data via given callback function.
`moduleName` and `methodName` must be present or error is thrown.

```
listen(moduleName, propertyName, callbackFunction)
```

WebSocket modules

Methods syntax

```
methodName(paramType paramName) : typeOfValueReturnedViaPromise
```

Properties syntax

```
propertyName : typeOfValueReceivedViaCallback
```

Actions

Methods

Call action without a parameter.

```
action(string actionName) : void
```

Call action with a parameter.

```
action(string actionName, string parameter) : void
```

Cameras

Methods

Return lists of live and recorder cameras.

```
cameras() : {
      "liveIds" : <int[]>,
      "recorderIds" : <int[]>
}
```

Return selected camera id.

```
selected() : int
```

Select given camera.

```
select(int cameraId) : void
```

Clips

Methods

Return clips.

```
clips() : {
      "name" : <string>,
      "index" : <int>,
      "bank" : <int>,
      "thumbnail" : <string> url of clip thumbnail or undefined
}
```

Return selected clip index.

```
selected() : int[bankId, clipIndex]
```

Select given clip.

```
select(int bank, int clipIndex) : void
```

Properties

Notified when clips are changed.

```
clips : {
      "name" : <string>,
      "index" : <int>,
      "bank" : <int>,
      "thumbnail" : <string> url of clip thumbnail or undefined
}
```

Notified when clip selection is changed.

```
selected : [int bank, int clipIndex]
```

Logging

Methods

Register logging source.

```
register(boolean isSource) : void
```

Log message.

```
log(string message) : void
```

Connect logging source.

```
connect(string srcId) : void
```

Disconnect logging source.

```
disconnect(string srcId) : void
```

Disconnect all logging sources.

```
disconnect() : void
```

Properties

Notified when log is enabled / disabled.

```
enabled : boolean
```

Notified when message is logged.

```
log : string
```

Notified when sources are changed.

sources : string (Json with available and listened sources)

Preview

Methods

Dispatch mouse click event.

```
mouseclick(float x, float y, int button, int modifiers) : void
```

Dispatch mouse down event.

```
mousedown(float x, float y, int button, int modifiers) : void
```

Dispatch mouse up event.

```
mouseup(float x, float y, int button, int modifiers) : void
```

Dispatch mouse move event.

```
mousemove(float x, float y, int modifiers) : void
```

Dispatch mouse drag event.

```
mousedrag(float x, float y, int button, int modifiers) : void
```

Dispatch mouse wheel event.

```
mousewheel(float x, float y, int deltaY, int modifiers) : void
```

Dispatch mouse enter event.

```
mouseenter(int modifiers) : void
```

Dispatch mouse leave event.

```
mouseleave(int modifiers) : void
```

Dispatch key down event.

```
keydown(int keyCode, int modifiers) : void
```

Dispatch key up event.

```
keyup(int keyCode, int modifiers) : void
```

Recorder

Methods

Play.

```
play() : void
```

Play with rate.

```
playWithRate(float rate) : void
```

Pause.

```
pause() : void
```

Jog by given amount of frames.

```
jog(int value) : void
```

Shuttle with given rate.

```
shuttle(float rate) : void
```

Seek to given position in clip (0 = begin, 1 = end).

```
seek(float value) : void
```

Properties

Notified when recorder state is changed.

```
state : [int cameraId, long timecode, float rate, float clipProgress,
long clipLength]
```

Save

Methods

Return current state of save button [0: HIDDEN, 1: SAVED, 2: UNSAVED, 3: EDIT].

```
state() : int
```

Properties

Notified when the save button state is changed.

```
state : int
```

Tools

Methods

Return tools.

```
tools() : {
      "name" : <string>,
      "id" : <int>,
      "toolbar" : <index>,
      "index" : <int>,
      "thumbnail" : <string> url of tool thumbnail
}
```

Set tool setting.

```
setSetting(int toolId, int settingIndex, int settingValue) : void
```

Return selected tool id.

```
selected() : int
```

Select given tool.

```
select(int toolId) : void
```

Properties

Notified when tools are changed.

```
tools : {
      "name" : <string>,
      "id" : <int>,
      "toolbar" : <index>,
      "index" : <int>,
```

```
      "thumbnail" : <string> url of tool thumbnail
}
```

Notified when tool selection is changed.

```
selected : int toolId
```