CISCO

You make **possible**

Advanced Coding for Cisco Video Devices

Davide Grandis, dgrandis@cisco.com
Technical Solutions Architect, EMEAR

Stève Sfartz, stsfartz@cisco.com
API Architect, DevNet

BRKDEV-3244

Barcelona | January 27-31, 2020

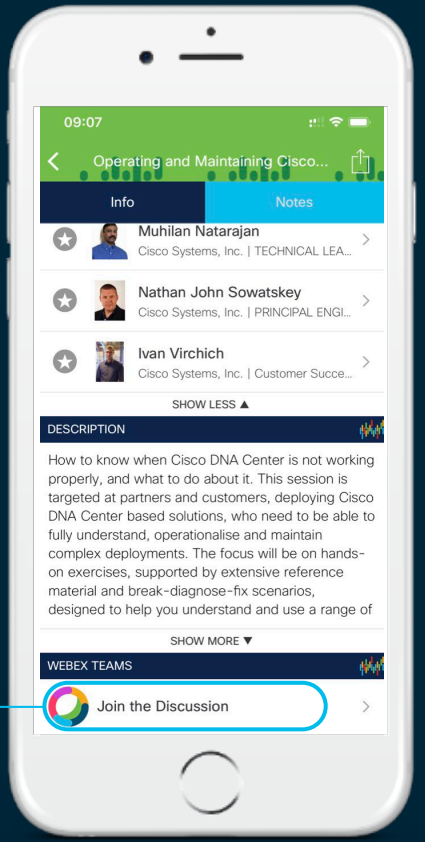
Cisco Webex Teams

Questions?

Use Cisco Webex Teams to chat with the speaker after the session

How

1. Find this session in the Cisco Events Mobile App
2. Click “Join the Discussion”
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space



/Cisco/DevNet/SteveSfartz

- API Architect at Cisco DevNet
- Working to deliver the greatest API Experience for Cisco's developer community
- Lead for Cisco's API Style Guide aiming for simplicity and consistency
- Webex Teams & Devices APIs
- Contributor to DevNet CodeExchange
  - code samples, developer tools, postman collections, awesome-webex, awesome-xapi...

webex: stsfartz@cisco.com
github: ObjectIsAdvantag
twitter: @SteveSfartz

"vision without execution is hallucination"

Davide Grandis

- Webex Architect in the Specialist Team EMEAR
- Lead for transition to the cloud journey for on-premises video customers specifically for meetings and video endpoints
- Lead for video endpoint programmability
- Cisco Webex Ambassador Professional
- EMEAR Quarterly Webinars for Partners

webex: dgrandis@cisco.com
twitter: @DavGrandis
github: DavCisco

"practice makes perfect"

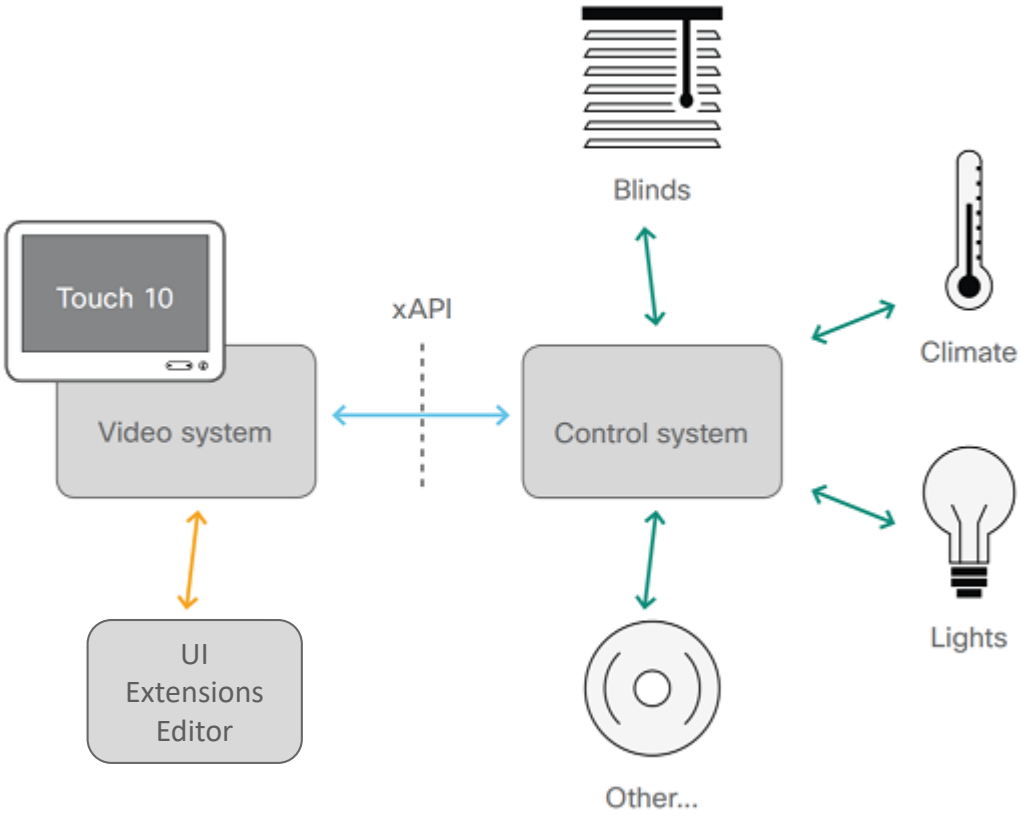
Agenda

- CE Programmability and Use Cases
- Creating UI Extensions
- Creating Macros and standalone Applications
- Deploying UI Extensions
- Deploying Macros

CE Progammability & use cases

Webex devices programmability at a glance

- Enhance the User eXperience of your meetings
  - add UI Extensions to Room/Desk/Boards devices
  - interact with IoT devices
  - customized behaviors as macros
  - Integrate with enterprise process
- Automate your devices from code
  - initiate calls, fetch history
  - collect room analytics
  - apply branding across devices
  - deploy UI Extensions & macros



Demo [OnAir]

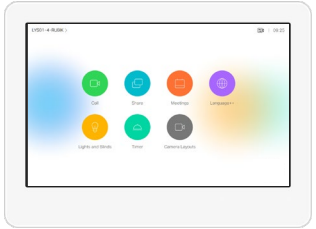
In-Room Control Architecture

HttpClient commands (GET/PUT/POST/DELETE/PATCH)

commands

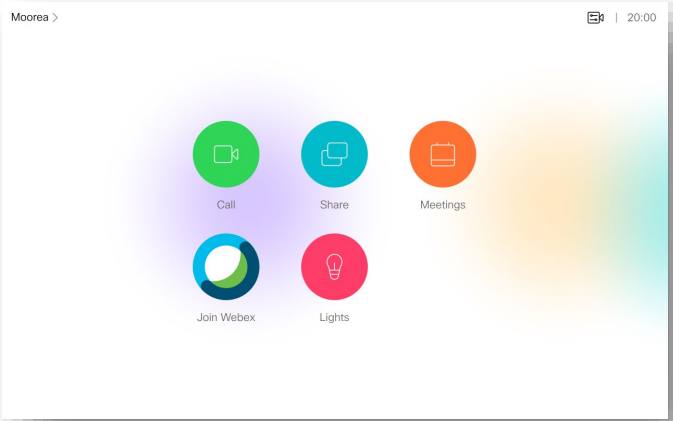
events

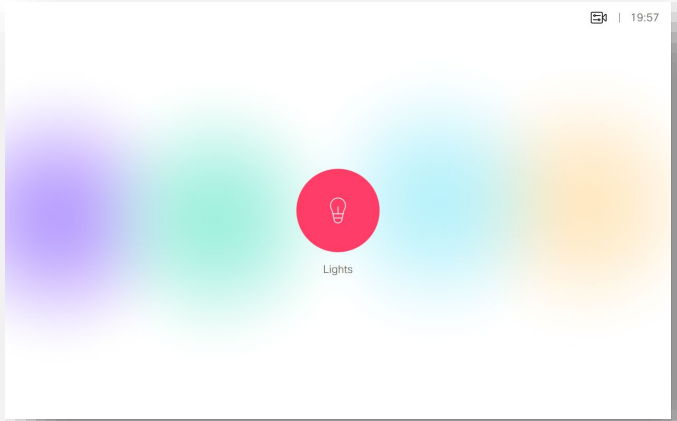
A single Touch10 interface to rule them all



Locked down interface

- ‘Kiosk-like’ User experience, no settings menu and standard buttons
- Example: Speed dial to experts via a custom button






```
xConfiguration UserInterface Features HideAll: True
xConfiguration UserInterface SettingsMenu Visibility: Hidden
```

Cisco Collaboration Devices Programmability

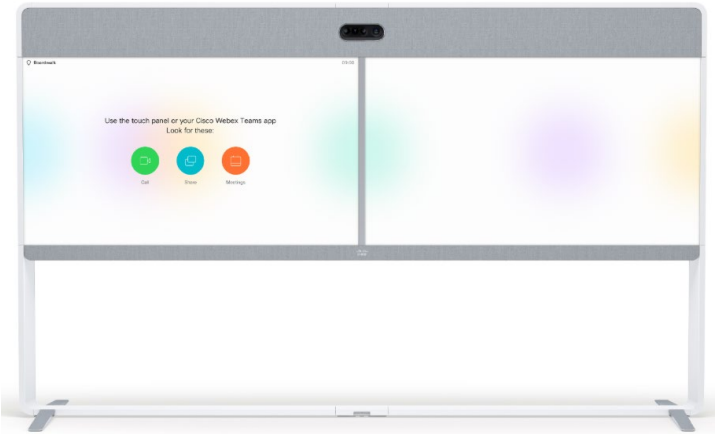
Available on all devices running CE & RoomOS

Room Kit, Room Kit Mini and Pro

*no Macros on SX10

Complete programmability

DX, Desk Pro



DevNet Sandboxes

WINDOW: INSTRUCTIONS | COMMANDS | ACTIVITY | OUTPUT | NAVIGATOR | STYLE: | BEHAVIOUR:

INSTRUCTIONS

Room Kit | DevBox | VPN

Overview:
This sandbox is provided for the purpose of discovering Cisco Collaboration Software - CE programmability for Room Devices (MX, SX, DX and Room Kit series). It comes with a Room Kit device and a developer box.

Integrators, IT engineers and application developers can discover the extensibility of Cisco Collaboration Endpoints, and learn to code for Room Devices by getting hands-on with In-Room Controls and Macros, but also checking status, changing configuration and sending commands through xAPI.

Roomkit:
Once you have created a VPN session to the sandbox, you can:

- **Discover the admin interface from your laptops Web browser**
- **Open a ssh session to the Room Kit and enter t-shell**

SANDBOX

Version CE 9.9

Room Kit
WebEx Device

Room Kit CE 9.9
Remotely connect with a Room Kit running CE9.9, and experiment with xAPI.

RESERVE

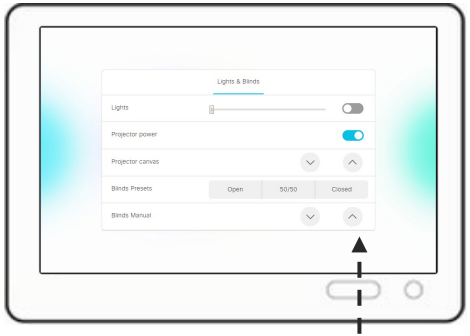
RoomKit-009 (Room...
10.10.20.160
Room Kit

DevBox_f83c983d
10.10.20.150
Model DevBox Linux

VLAN 2003
Service
VLAN GWY RANGE1

CE Programmability (xAPI)

Touch10 Interface



Standalone
(JavaScript, Python...)

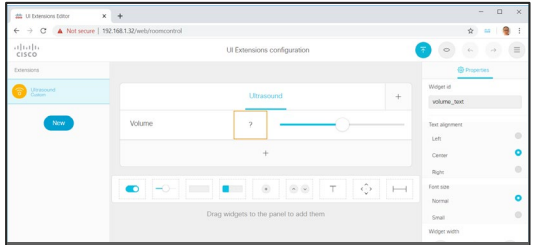
Macros
(JavaScript)

- status
- configure
- command

*over ssh, Websocket, serial, or HTTP**

events

deploy



UI Extensions Editor

* HTTP is unidirectional (cannot listen to events)

xAPI module at DevNet

https://developer.cisco.com/learning/modules/xapi-intro

Introduction to Webex Devices Programmability

Discover how to customize and extend Webex Devices through xAPI – the API exposed by Cisco Collaboration Endpoint CE software. Learn to configure your device, start video calls from code, add Branding but also how to create custom In-Room Controls and deploy Macros on your devices. Go hands-on with a CE-capable device or a provided RoomKit sandbox. This module assumes you have some basic programming experience.

1 Hour 50 Minutes

Introduction to xAPI for Cisco Collaboration Devices

Explore the programmability of Cisco Collaboration Devices and understand xAPI – the API exposed by Cisco TelePresence CE software.

Creating custom in-room controls for Cisco collaboration devices

Learn how to create custom in-room controls for Cisco collaboration devices, using the on-board control simulator tool and the in-room control editor. Then make those controls interactive via a PC-based Node.js script.

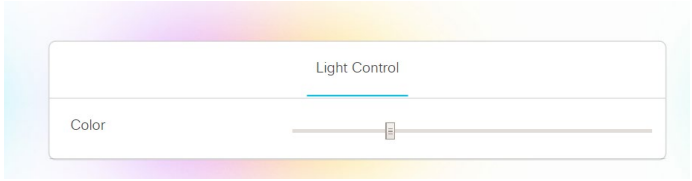
Customizing collaboration devices from code

Learn to customize display logos, signage and custom messages for Cisco Collaboration Devices via SSH, HTTP and Node.js/JavaScript.

Creating UI Extensions and Macros

From Zero to Hero

Tips & best practices to build & deploy



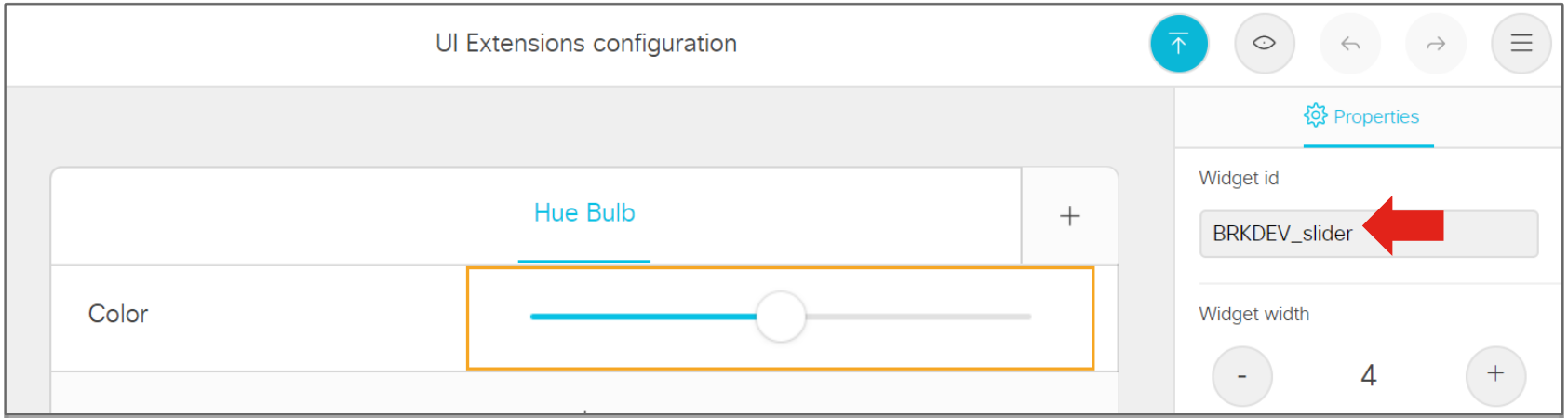
Event

Macro

HTTP Put

Command

UI Extensions Editor / Macro Editor




```
const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {
   if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
      console.log(`updated slider to: ${event.Value}`);
   }
});
```

Changing the color of a Philips Hue bulb

Hue bridge IP | Hue user | light number

PUT http://172.17.201.69/api/SECRET/lights/4/state

{ "hue": 25500 }

color

200 OK

[{ "success": { "/lights/4/state/hue": 25500 } }]

Command Reference

xCommand HttpClient Put

Applies to: *All products*

Requires user role: ADMIN

Sends an HTTP(S) Put request to the server that is specified in the Url parameter.

You can use the AllowInsecureHTTPS parameter to specify whether or not to validate the server's certificate before sending data over HTTPS. This parameter has no effect unless the xConfiguration HttpClient AllowInsecureHTTPS is set to On.

This is a multiline command, so the payload (data) follows after the parameters.

USAGE:

```
xCommand HttpClient Put [AllowInsecureHTTPS: AllowInsecureHTTPS] [Header:
"Header"] [Timeout: Timeout] Url: "Url"
```

```
xConfiguration HttpClient Mode: On
xConfiguration HttpClient AllowInsecureHTTPS: True
```

Required

Changing the color of a Philips Hue bulb

```
xapi.command('HttpClient Put', {
    Header: ["Content-Type: application/json"],
    Url: `http://${bridgeip}/api/${user}/lights/${light}/state`,
    AllowInsecureHTTPS: "True"
  },
  JSON.stringify({ "hue": 25500 }))
```

Demo
[Putting the pieces together]

10 lines of code?

```
const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {

   if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {

      // Change color for Philips Hue light
      const HUE_BRIDGE = '192.168.1.33';
      const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
      const HUE_LIGHT = 1; // number of the light in your deployment

      xapi.command('HttpClient Put', {
            Header: ["Content-Type: application/json"],
            Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
            AllowInsecureHTTPS: "True",
            ResultBody: 'plaintext'
         },
         JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
   }
```


Is the device configured for HttpClient?

```
// The device must be configured to support HttpClient
//  - xConfiguration HttpClient Mode: On
//  - xConfiguration HttpClient AllowInsecureHTTPS: True
```

Is this the correct event?

Is the panel deployed?

```
const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {
  if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
    // Change color for Philips Hue light
    const HUE_BRIDGE = '192.168.1.33';
    const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
    const HUE_LIGHT = 1; // number of the light in your deployment
    xapi.command('HttpClient Put', {
        Header: ["Content-Type: application/json"],
        Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
        AllowInsecureHTTPS: "True",
        ResultBody: 'plaintext'
      },
      JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
  }
```

How can I pass these values?

What if the request fails?

Tips & Best Practices

```
// The device must be configured to support HttpClient
//  - xConfiguration HttpClient Mode: On
//  - xConfiguration HttpClient AllowInsecureHTTPS: True

const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {

   if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {

      // Change color for Philips Hue light
      const HUE_BRIDGE = '192.168.1.33';
      const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
      const HUE_LIGHT = 1; // number of the light in your deployment

      xapi.command('HttpClient Put', {
            Header: ["Content-Type: application/json"],
            Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
            AllowInsecureHTTPS: "True",
            ResultBody: 'plaintext'
         },
         JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
   }
```

1

Is this the correct event?

Detecting events

1) ‘Preview’ mode from the UI Extensions Editor

From Touch	From Control system
BRKDEV /**Opened**	
	BRKDEV_slider /**170**
BRKDEV_slider /changed /**170**	
	BRKDEV_slider /**141**
BRKDEV_slider /changed /**141**	
	BRKDEV_slider /**127**
BRKDEV_slider /changed /**127**	
	BRKDEV_slider /**126**
BRKDEV_slider /changed /**126**	
BRKDEV_slider /released /**126**	

Detecting events

1) ‘Preview’ mode from the UI Extensions Editor
2) On the command line: ‘xFeedback register’

```
xfeedback register /Event/UserInterface/Extensions

*e UserInterface Extensions Panel Clicked PanelId: "BRKDEV"
** end
*e UserInterface Extensions Event Pressed Signal: "BRKDEV_slider:135"
** end
*e UserInterface Extensions Widget Action WidgetId: "BRKDEV_slider"
*e UserInterface Extensions Widget Action Value: "135"
*e UserInterface Extensions Widget Action Type: "pressed"
** end

xfeedback deregister /Event/UserInterface/Extensions
```

Detecting events

- To detect ALL events fired on a device

```
xfeedback register /
```

- Don’t forget to deregister

```
xfeedback deregisterall
```

Detecting events

1) ‘Preview’ mode from the UI Extensions Editor
2) On the command line: ‘xFeedback register’
3) Via a Macro

```
1  const xapi = require('xapi');
2
3  xapi.event.on('UserInterface Extensions', (event) => {
4     console.log(`NEW event: ${JSON.stringify(event)}`);
5  });
6
```

Severity ▾ | Enter filter | ☐ Show history

```
18:15:05  [system] > Using XAPI transport: TSH
18:15:05  [system] > Starting macros...
18:15:05  slider   > Loading...
18:15:07  slider   > Ready!
18:15:08  slider   > 'NEW event: {"id":"1","Panel":{"id":"1","Clicked":{"id":"1","PanelId":"BRKDEV"}}}'
18:15:09  slider   > 'NEW event: {"id":"1","Event":{"id":"1","Pressed":{"id":"1","Signal":"BRKDEV_slider:150"}}}'
18:15:09  slider   > 'NEW event: {"id":"1","Widget":{"id":"1","Action":{"id":"1","WidgetId":"BRKDEV_slider","Value":"150","Type":"pressed"}}}'
```

Detecting events

1) ‘Preview’ mode from the UI Extensions Editor
2) On the command line: ‘xFeedback register’
3) Via a Macro
4) Debugging in VSCode

```
41
42
43  //
44  // Code logic
45  //
46
47  function init() {
48      xapi.event.on('UserInterface Extensions Widget Action', (event) => {
49          if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
50              console.debug(`updated slider to: ${event.Value}`);
51              changeColorFromSliderLevel(parseInt(event.Value));
52          }
53      });
54  }
```

```
Object {id: "1", WidgetId: "BRKDEV_slider", Value…
  id: "1"
  Type: "pressed"
  Value: "134"
  WidgetId: "BRKDEV_slider"
> __proto__: Object {constructor: , __defin
```

Is the panel deployed?

2

```
// The device must be configured to support HttpClient
//  - xConfiguration HttpClient Mode: On
//  - xConfiguration HttpClient AllowInsecureHTTPS: True

const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {

  if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {

    // Change color for Philips Hue light
    const HUE_BRIDGE = '192.168.1.33';
    const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
    const HUE_LIGHT = 1; // number of the light in your deployment

    xapi.command('HttpClient Put', {
        Header: ["Content-Type: application/json"],
        Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
        AllowInsecureHTTPS: "True",
        ResultBody: 'plaintext'
      },
      JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
  }
```


Is the panel even deployed?

- Check the components are present among the list of UI Extensions

```
xstatus /UserInterface/Extensions
*s UserInterface Extensions Widget 1 Value: "63"
*s UserInterface Extensions Widget 1 WidgetId: "BRKDEV_slider"
** end
```

<u>Consider these events to optimize the user experience:</u>

- Initialize panels as they are (re-)deployed

```
*e UserInterface Extensions Widget LayoutUpdated
```

- Take action when a panel is opened

```
*e UserInterface Extensions Panel Clicked PanelId: "BRKDEV"
```

```
// The device must be configured to support HttpClient
//  - xConfiguration HttpClient Mode: On
//  - xConfiguration HttpClient AllowInsecureHTTPS: True

const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {

   if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {

      // Change color for Philips Hue light
      const HUE_BRIDGE = '192.168.1.33';
      const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
      const HUE_LIGHT = 1; // number of the light in your deployment

      xapi.command('HttpClient Put', {
            Header: ["Content-Type: application/json"],
            Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
            AllowInsecureHTTPS: "True",
            ResultBody: 'plaintext'
         },
         JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
   }
```

3

What if the request fails?

HttpClient with response body…

```
xapi.command("HttpClient Put", {
        Header: ["Content-Type: application/json"],
        Url: `http://${bridgeip}/api/${apikey}/lights/${light}/state`,
        AllowInsecureHTTPS: "True",
        ResultBody: "plaintext"
    },
    JSON.stringify({ "hue": 25000 }))
  .then((response) => {

    console.log(`request sent, status code: ${response.StatusCode}`)
    console.debug(`received response: ${response.Body}`)

    // Check response
    let result = JSON.parse(response.Body)
    if (result[0]) {
        if (result[0].success) {
            console.log("success")
```


...and correct error handling

```
.then((response) => {
    if (response.StatusCode == 200) {
        console.log("message pushed to bridge");

        // Retrieve response
        console.debug(`received response: ${response.Body}`);
        let result = JSON.parse(response.Body);
        if (result[0] && (result[0].success)) {
            console.debug("success");
            if (cb) cb(result[0].success);
            return;
        }

        if (result[0] && (result[0].error)) {
            console.debug("error");
            if (cb) cb(null, result[0].error);
            return;
        }

        return;
    }

    console.log("failed with status code: " + response.StatusCode);
    if (cb) cb("failed with status code: " + response.StatusCode, response.StatusCode);
})
.catch((err) => {
    console.log("failed with err: " + err.message);
    if (cb) cb("Could not contact the bridge");
});
```

...and correct error handling

```
function updateLight(bridgeip, username, light, payload, cb) {

   // Post message
   xapi.command(
      'HttpClient Put',
      {
         Header: ["Content-Type: application/json"],
         Url: `http://${bridgeip}/api/${username}/lights/${light}/state`,
         AllowInsecureHTTPS: "True",
         ResultBody: 'plaintext'
      },
      JSON.stringify(payload))
      .then((response) => {
         if (response.StatusCode == 200) {
            console.log("message pushed to bridge");

            // Retrieve response
            console.debug(`received response: ${response.Body}`);
            let result = JSON.parse(response.Body);
            if (result[0] && (result[0].success)) {
               console.debug("success");
               if (cb) cb(result[0].success);
               return;
            }

            if (result[0] && (result[0].error)) {
               console.debug("error");
               if (cb) cb(null, result[0].error);
               return;
            }

            return;
         }
```

```
         console.log("failed with status code: " + response.StatusCode);
         if (cb) cb("failed with status code: " + response.StatusCode, response.StatusCode);
      })
      .catch((err) => {
         console.log("failed with err: " + err.message);
         if (cb) cb("Could not contact the bridge");
      });
}
```

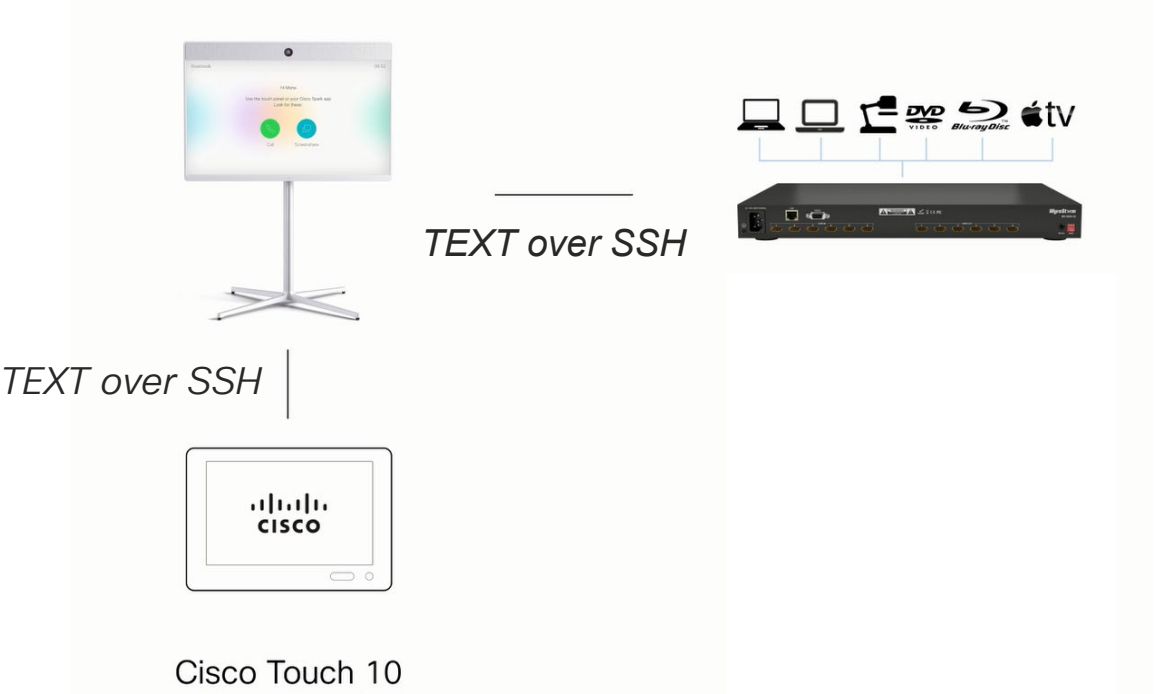
4

<u>Is it too much to ask?</u>

- to the Macro Editor
- from the Macro Runtime
- considering my coding skills

Let’s take a few steps back…

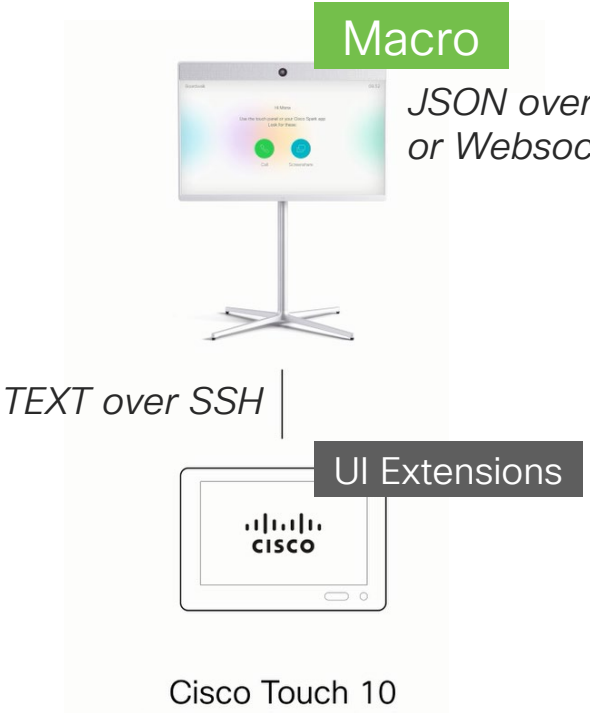
Historically… and still relevant for some use cases



Macro Runtime

- CE & RoomOS Webex devices (except SX10)
- Maximum number of macros (10 as of today)
- Macros are executed as ‘Admin’ role
- Macro tutorial accessible from the codec
  - http://<ip-address>/static/docs/macro-tutorial.pdf
- Duktape JavaScript runtime with Babel
  - babel-preset-latest’ is specified to support latest ES6+ features
  - async/await is supported

Embedded macros…



Tuning the Macro Runtime

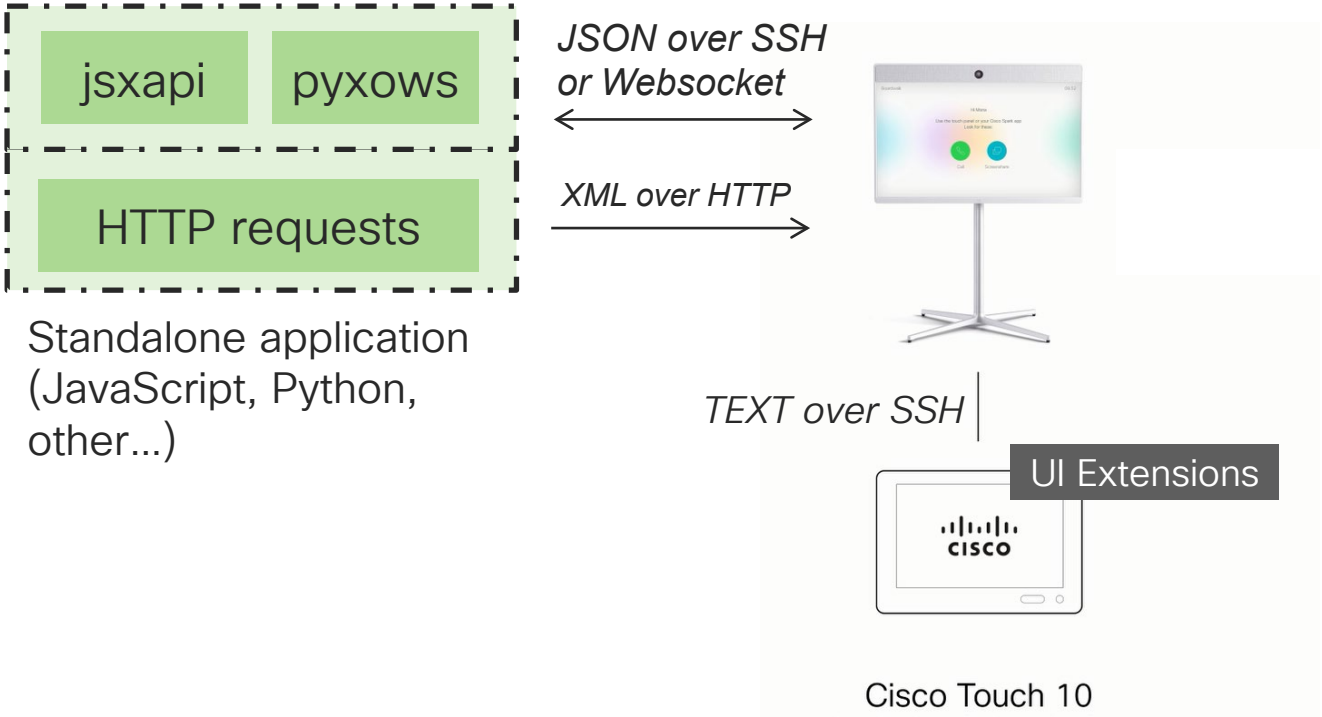
Configuration

Macros

AutoStart	Off
Mode	On
UnresponsiveTimeout	10 (0 to 65535)
XAPI Transport	TSH

TSH
WebSocket

… or standalone applications



Standalone application (JavaScript, Python, other…)

‘jsxapi’: the Node.js module for xAPI

https://github.com/cisco-ce/jsxapi

- Build CE integrations as standalone applications
- MIT License, available on npm, source code on Github
- **Same capabilities as the Macro runtime**
- Connect via ‘ssh’ or ‘websockets’ (since jsxapi v4.4 & CE 9.8)
- TypeScript and new style coming with jsxapi v5

Node.js ‘jsxapi’ module

https://github.com/cisco-ce/jsxapi

coming

- TypeScript and new style coming with jsxapi v5

```
// Set up a call
xapi.Command.Dial({ Number: 'user@example.com' });

// Fetch volume and print it
xapi.Status.Audio.Volume
  .get()
  .then((volume) => { console.log(volume); });
```


Node.js ‘jsxapi’ module

https://github.com/cisco-ce/jsxapi

- Connect via ‘ssh’ or ‘websockets’ (since jsxapi v4.4 & CE 9.8)

```
const jsxapi = require('jsxapi');

jsxapi
  .connect('ssh://host.example.com', {
    username: 'admin',
    password: 'password',
  })
  .on('error', console.error)
  .on('ready', async (xapi) => {
    const volume = await xapi.status.get('Audio Volume');
    console.log(`volume is: ${volume}`);
    xapi.close();
  });
```

```
jsxapi
  .connect('wss://host.example.com', {
    username: 'admin',
    password: 'password',
  })
```

- Performances
- Connect from Web applications without proxying

Embedded macro or standalone app?

Macros

- Status, configuration, commands, events
- Invoke HTTP-based APIs
- Limited number of macros (currently 10)
- Stateless only (no db persistance)*
- No macro to macro communication**

<u>Use cases</u>

⇒ Custom behaviors

⇒ Interact with IoE exposing HTTP APIs

Standalone Apps

- Status, configuration, commands, events
- Interact with legacy APIs (beyond REST)
- Server-side limitations only (memory, disk)
- Stateful with data persistance

<u>Use cases</u>

⇒ Manage/Automate multiple devices

⇒ Integrate with processes or 3rd party product

...with proper error handling

```
function updateLight(bridgeip, username, light, payload, cb) {

   // Post message
   xapi.command(
      'HttpClient Put',
      {
         Header: ["Content-Type: application/json"],
         Url: `http://${bridgeip}/api/${username}/lights/${light}/state`,
         AllowInsecureHTTPS: "True",
         ResultBody: 'plaintext'
      },
      JSON.stringify(payload))
      .then((response) => {
         if (response.StatusCode == 200) {
            console.log("message pushed to bridge");

            // Retrieve response
            console.debug(`received response: ${response.Body}`);
            let result = JSON.parse(response.Body);
            if (result[0] && (result[0].success)) {
               console.debug("success");
               if (cb) cb(result[0].success);
               return;
            }

            if (result[0] && (result[0].error)) {
               console.debug("error");
               if (cb) cb(null, result[0].error);
               return;
            }

            return;
         }

         console.log("failed with status code: " + response.StatusCode);
         if (cb) cb("failed with status code: " + response.StatusCode, response.StatusCode);
      })
      .catch((err) => {
         console.log("failed with err: " + err.message);
         if (cb) cb("Could not contact the bridge");
      });
}
```

Is it too much to ask?

- to the Macro Editor
- from the Macro Runtime
- considering my coding skills

5

IDE: a better coding experience!

File Edit Selection View Go Debug Terminal Help — app.js - standalone - Visual Studio Code

EXPLORER

OPEN EDITORS
- app.js U
- launch.json .vscode
- multi.js M

STANDALONE
- .vscode
  - launch.json
- node_modules
- app.js U
- multi.js M
- package-lock.json
- package.json

app.js | launch.json | multi.js

app.js > ...

```
16  const jsxapi = require('jsxapi');
17  const xapi = jsxapi.connect(process.env.JSXAPI_DEVICE_URL, {
18      username: process.env.JSXAPI_USERNAME,
19      password: process.env.JSXAPI_PASSWORD ? process.env.JSXAPI_PASSWORD : ""
20  })
21      .on('error', (err) => {
22          console.error(`connexion failed: ${err}, exiting`);
23          process.exit(1);
24      })
25      .on('close', () => {
26          console.error(`connexion closed, exiting`);
27          process.exit(1);
28      })
29      .on('ready', () => {
30          console.log('connected!');
31          init();
32      });
33
34  // Code logic
35  function init() {
36      xapi.event.on('UserInterface Extensions Widget Action', (event) => {
37          if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
38              console.debug(`updated slider to: ${event.Value}`);
39              changeColorFromSliderLevel(parseInt(event.Value));
40          }
41      });
42  }
```

Demo
[a better coding experience]

Code/debug with ‘jsxapi’, deploy as a Macro

Running as a Node.js application

```
> npm install jsxapi
```

```
// Connect to your device
const jsxapi = require(‘jsxapi’);
const xapi = jsxapi.connect(ip, {user, passwd})
  .on(‘ready’, init)
  .on(‘error’, ...)
  .on(‘close’, ...);
```

Running as a Macro

```
# no install
```

```
// Embedded, no connection need
const xapi = require(‘xapi’);
xapi.on(‘ready’, init)
```

```
function init() {
  // custom code logic
  xapi.command(…
}
```

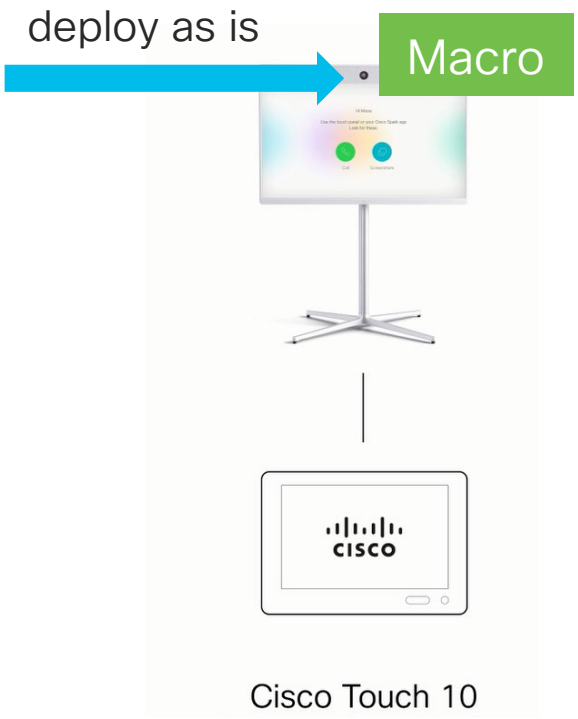
☺ code logic can be simply copy/pasted

Initializing your UI Extension Panel

```
const xapi = require('xapi');

xapi.on('ready', init);

// Code logic
function init() {
    xapi.event.on('UserInterface Extensions Widget Action', (event) => {
        if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
            console.debug(`updated slider to: ${event.Value}`);
            changeColorFromSliderLevel(parseInt(event.Value));
        }
    });
}
```


A single codebase to rule them all

```
// Detect if the code running as a JS macro or as a Node.js process
let xapi;
try {
    // Succeeds if running as a macro
    xapi = require('xapi');
}
catch (err) {
    //Running as a standalone app: Connect to the device

    console.log(`connecting to device with url: ${process.env.JSXAPI_DEVICE_URL}`);
    const jsxapi = require('jsxapi')
    xapi = jsxapi.connect(process.env.JSXAPI_DEVICE_URL, {
        username: process.env.JSXAPI_USERNAME,
        password: process.env.JSXAPI_PASSWORD ? process.env.JSXAPI_PASSWORD : ""
    })
        .on('error', (err) => {…
        })
        .on('close', () => {…
        });
}

xapi.on('ready', () => {
    console.log('connected!');
    init();
});
```



Is the device configured for HttpClient?

6

```
// The device must be configured to support HttpClient
//   - xConfiguration HttpClient Mode: On
//   - xConfiguration HttpClient AllowInsecureHTTPS: True

const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {
  if ((event.WidgetId == 'BRKDEV_slider') && (event.Type == 'released')) {
```

Is the panel deployed?

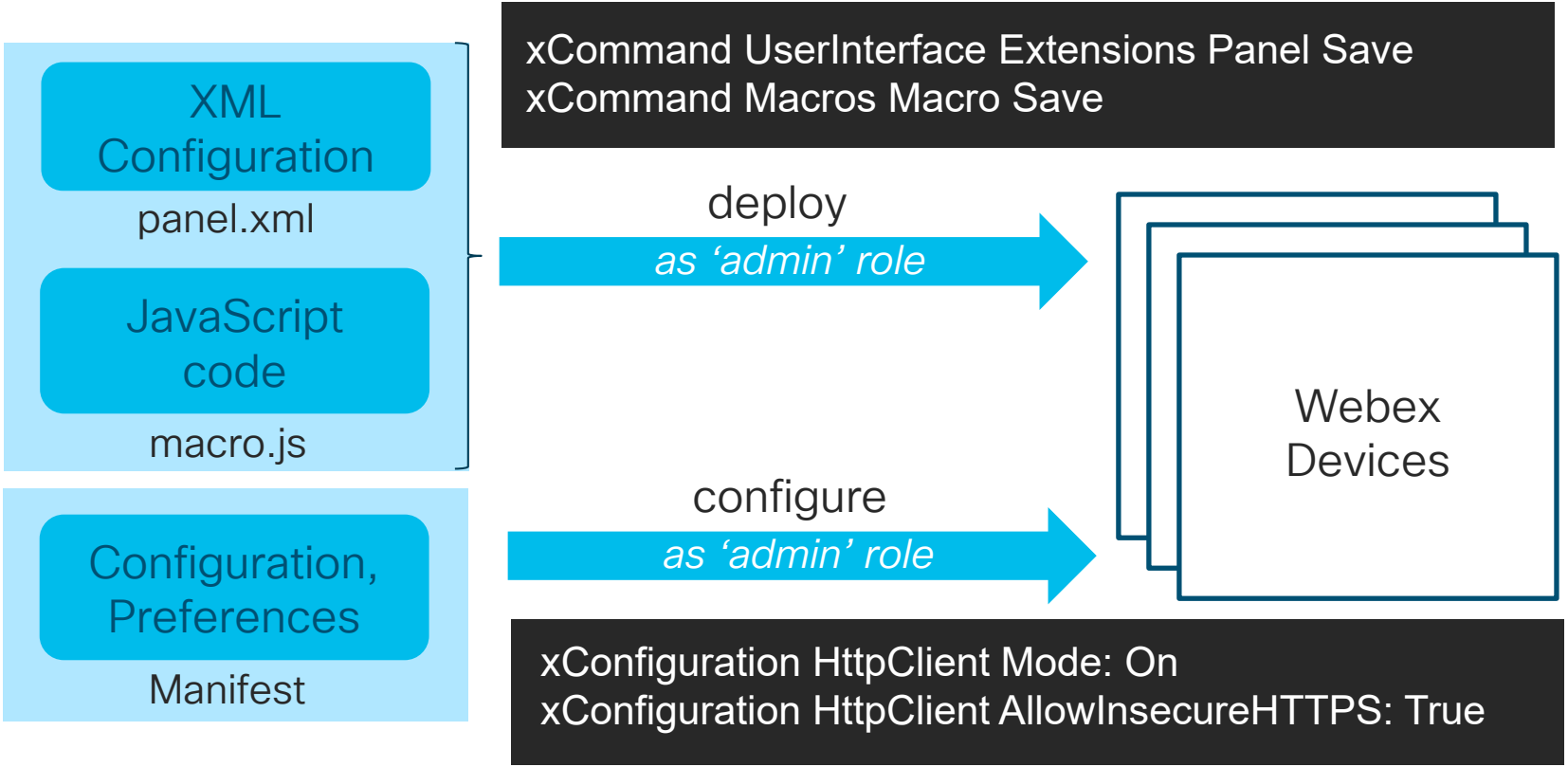
6

How can I pass these values?

```
    // Change color for Philips Hue light
    const HUE_BRIDGE = '192.168.1.33';
    const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
    const HUE_LIGHT = 1; // number of the light in your deployment

    xapi.command('HttpClient Put', {
        Header: ["Content-Type: application/json"],
        Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
        AllowInsecureHTTPS: "True",
        ResultBody: 'plaintext'
      },
      JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
  }
```

Deploying UI Extensions & Macros

Deploying to Webex Devices



Automated deployment of Macros

xCommand Macros Macro Save

Applies to: *DX70/DX80 SX20 SX80 MX200G2/MX300G2 MX700/MX800/MX800D RoomKit RoomKitMini CodecPlus CodecPro Room55 Room70/Room55D Room70G2 Boards*

Requires user role: ADMIN

Saves the details of a macro. This is a multiline command.

USAGE:

`xCommand Macros Macro Save Name: "`*`Name`*`" [Overwrite: `*`Overwrite`*`] [Transpile: `*`Transpile`*`]`

where

`Name`:

String (0..255)

The name of the macro that is saved.

`Overwrite`:

False/True

XML over HTTP: /putxml

unidirectional

POST https://192.168.1.26/putxml

Params | Authorization | Headers (2) | Body | Pre-request Script | Tests

none | form-data | x-www-form-urlencoded | raw | binary | XML (text/xml)

```
<Command>
    <Macros>
        <Macro>
            <Save>
                <Name>HelloWorld</Name>
                <OverWrite>True</OverWrite>
                <body>console.log("hello world")</body>
            </Save>
        </Macro>
    </Macros>
</Command>
```

Post Macros as ‘text’,
as ‘Base64’ encoding for images
Escape XML tags for UI Extensions.

serial port
ssh
HTTP
WebSocket

text
XML
JSON

Automated deployment of UI Extensions

xCommand UserInterface Extensions Panel Save

Applies to: *All products*

Requires user role: ADMIN, INTEGRATOR, ROOMCONTROL

Adds a custom panel to the current configuration. The panel will be added to the configuration, but if a panel with the same panel ID already exists, it will be overwritten. This is a multiline command.

USAGE:

`xCommand UserInterface Extensions Panel Save PanelId: "PanelId"`

where

PanelId:

String (0..255)

The unique identifier of the custom panel.

Deploying UI Extensions

xAPI over HTTP

```
<Extensions>
  <Version>1.6</Version>
  <Panel>
    <PanelId>BRKDEV</PanelId>
    <Type>Home</Type>
    <Icon>Lightbulb</Icon>
    <Order>3</Order>
    <Color>#FF3D67</Color>
    <Name>Lights</Name>
    <ActivityType>Custom</ActivityType>
    <Page>
      <Name>Lights</Name>
      <Row>
        <Name>Color</Name>
        <Widget>
          <WidgetId>BRKDEV_slider</WidgetId>
          <Type>Slider</Type>
          <Options>size=4</Options>
        </Widget>
      </Row>
      <Options/>
    </Page>
  </Panel>
</Extensions>
```

Not considered: the panel is passed as a parameter in the /putxml payload

Customize for your device and/or user preferences

Ensure unique widget identifiers across local panels

Deploying UI Extensions

xAPI over HTTP

POST {{endpoint}}/putxml

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL BETA | XML

```
1  <Command>
2      <UserInterface>
3          <Extensions>
4              <Panel>
5                  <Save>
6                      <PanelId>BRKDEV</PanelId>
7                          <body>
8                              &lt;Extensions&gt;
9                                &lt;Version&gt;1.6&lt;/Version&gt;
10                               &lt;Panel&gt;
11                                &lt;Type&gt;Home&lt;/Type&gt;
12                                &lt;Icon&gt;Lightbulb&lt;/Icon&gt;
13                                &lt;Order&gt;1&lt;/Order&gt;
14                                &lt;Color&gt;#AACCEE&lt;/Color&gt;
15                                &lt;Name&gt;Lights&lt;/Name&gt;
16                                &lt;ActivityType&gt;Custom&lt;/ActivityType&gt;
17                                &lt;Page&gt;
```

XML escaped

Deploying UI Extensions

xAPI over HTTP

```
curl --request  POST 'http://192.168.1.32/putxml' \
     --header   'Content-Type: text/xml' \
     --header   'Authorization: Basic bG9jYWxhZG1pbjpjaXNjb3BzZHQ=' \
     --data-raw
"<Command><UserInterface><Extensions><Panel><Save><PanelId>BRKDEV</PanelI
d><body>&lt;Extensions&gt;&lt;Version&gt;1.6&lt;/Version&gt;&lt;Panel&gt;
&lt;Type&gt;Home&lt;/Type&gt;&lt;Icon&gt;Lightbulb&lt;/Icon&gt;&lt;Order&
gt;1&lt;/Order&gt;&lt;Color&gt;#AACCEE&lt;/Color&gt;&lt;Name&gt;Lights&lt
;/Name&gt;&lt;ActivityType&gt;Custom&lt;/ActivityType&gt;&lt;Page&gt;&lt;
Name&gt;Light&lt;/Name&gt;&lt;Row&gt;&lt;Name&gt;Color&lt;/Name&gt;&lt;Wi
dget&gt;&lt;WidgetId&gt;BRKDEV_slider&lt;/WidgetId&gt;&lt;Type&gt;Slider&
lt;/Type&gt;&lt;Options&gt;size=4&lt;/Options&gt;&lt;/Widget&gt;&lt;/Row&
gt;&lt;Options/&gt;&lt;/Page&gt;&lt;/Panel&gt;&lt;/Extensions&gt;</body><
/Save></Panel></Extensions></UserInterface></Command>"
```


Deploying UI Extensions

xAPI over SSH or WebSockets

```
async with xows.XoWSClient(url_or_host=device_url, username=user, password=passwd) as client:

    panel_id = 'BRKDEV'
    panel = '<Extensions> \
        <Version>1.6</Version> \
        <Panel> \
            <Type>Home</Type> \
            <Icon>Lightbulb</Icon> \
            <Order>1</Order> \
            <Color>#AACCEE</Color> \
            <Name>Lights</Name> \
            <ActivityType>Custom</ActivityType> \
            <Page>... </Page> \
        </Panel> \
        </Extensions>'

    response = await client.xCommand(['UserInterface', 'Extensions', 'Panel', 'Save'], \
        PanelId=panel_id, body=panel)
    if (response['status'] == 'OK'):
        print(f'successfully deployed extension: {panel_id}')
```

/!\ no need to XML escape when deploying with ‘jsxapi’

Deploying UI Extensions

xAPI over SSH or WebSockets

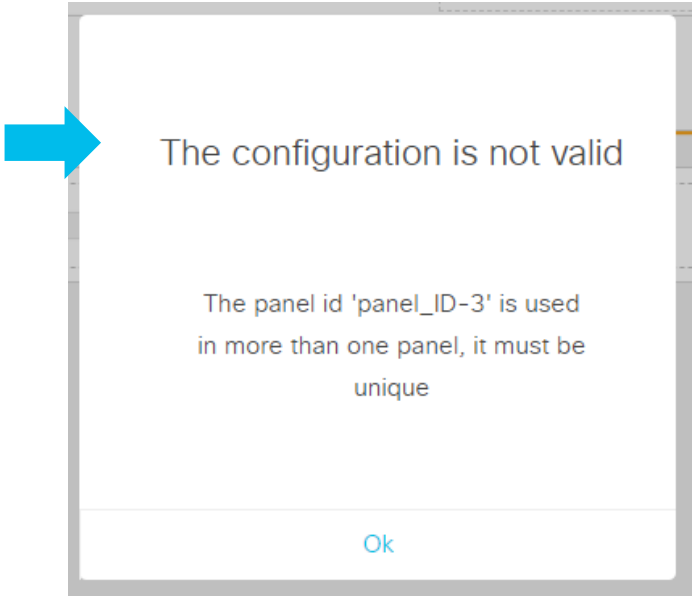
/!\ no need to XML escape when deploying with ‘pyxows’

```
xapi.command('UserInterface Extensions Panel Save',
  { PanelId: 'BRKDEV' },
  `<Extensions>
  <Version>1.6</Version>
   <Panel>
     <Type>Home</Type>
     <Icon>Lightbulb</Icon>
     <Order>1</Order>
     <Color>#AACCEE</Color>
     <Name>Lights</Name>
     <ActivityType>Custom</ActivityType>
     <Page>
       <Name>Light</Name>
       <Row>
         <Name>Color</Name>
         <Widget>
           <WidgetId>BRKDEV_slider</WidgetId>
```


UI Extensions: naming conventions (1/2)

- Panel identifiers must be unique
  - The configuration is checked by the UI Extensions Editor before deployment
  - If a panel with the same identifier is saved via xAPI, it will over-ride the existing one

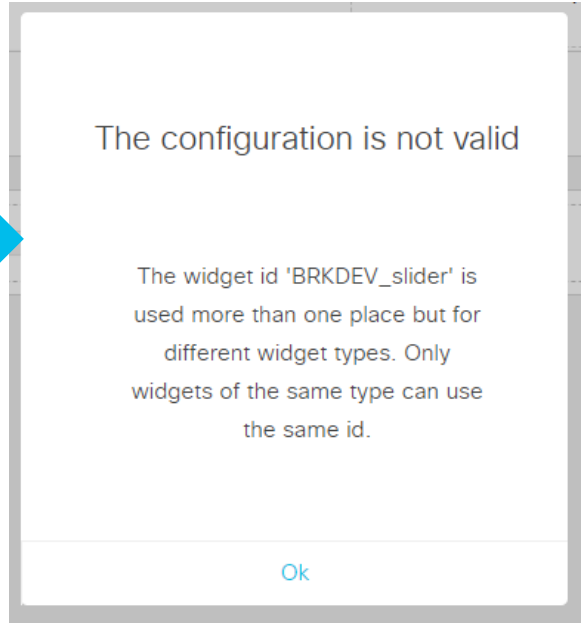
- Tip: Change the default ‘panel_1’
  - Example: ‘LIGHT’ or ‘BRKDEV’



UI Extensions: naming conventions (2/2)

- Each widget is assigned a unique identifier automatically
  - A widget with a similar name and type will be shared across several pages or panels

- Tip: Use the same prefix for the various widgets of a panel
  - Example: ‘BRKDEV_slider’
  - Example: ‘BRKDEV_volume_slider’



```
// The device must be configured to support HttpClient
//  - xConfiguration HttpClient Mode: On
//  - xConfiguration HttpClient AllowInsecureHTTPS: True

const xapi = require('xapi');

xapi.event.on('UserInterface Extensions Widget Action', (event) => {

    if ((event.Widgetid == 'BRKDEV_slider') && (event.Type == 'released')) {

        // Change color for Philips Hue light
        const HUE_BRIDGE = '192.168.1.33';
        const HUE_USERNAME = 'EM2Vg2GtNUqAASukv47wm1pWY0FayFe48D03f6Cb';
        const HUE_LIGHT = 1; // number of the light in your deployment

        xapi.command('HttpClient Put', {
            Header: ["Content-Type: application/json"],
            Url: `http://${HUE_BRIDGE}/api/${HUE_USERNAME}/lights/${HUE_LIGHT}/state`,
            AllowInsecureHTTPS: "True",
            ResultBody: 'plaintext'
        },
        JSON.stringify({ "hue": Math.round(event.Value / 255 * 65535), "sat": 255 }));
    }
```

7

How can I pass these values?

Deployment challenges for Macros

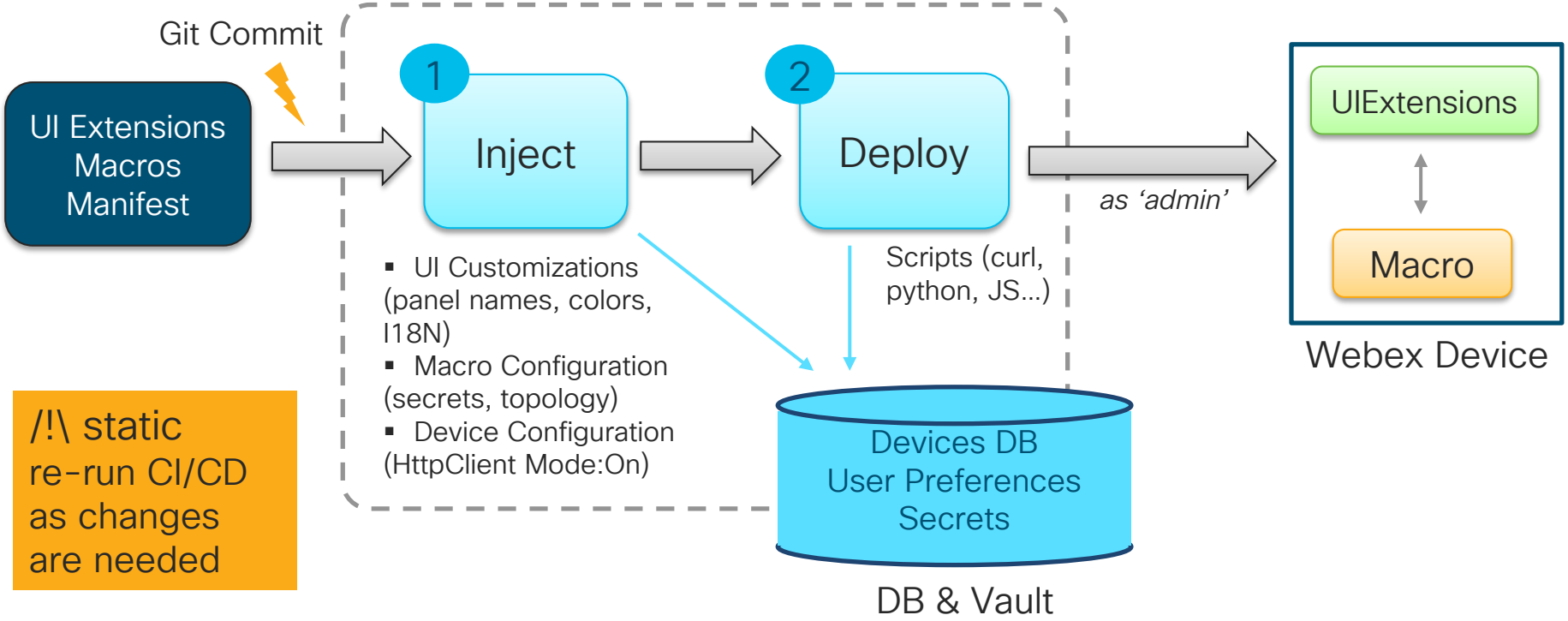
- Technical pre-requisites (Manifest)
  - Example: xConfigure HttpClient Mode: On
- Topology Information
  - Example: HUE_BRIDGE = ‘192.168.1.11’, HUE_BULB_ID = 3
- Secrets (API Keys)
  - Example: HUE_USERNAME = ‘RYXWAZJ63VZ3RFSDVZ345EFVZ43V’

⇒ common approaches

- Inject topology & secrets along the CI/CD pipeline
- Load dynamically from a vault
- Pass values as environment variables

Deployment strategies

Inject topology and secrets along the CI/CD pipeline

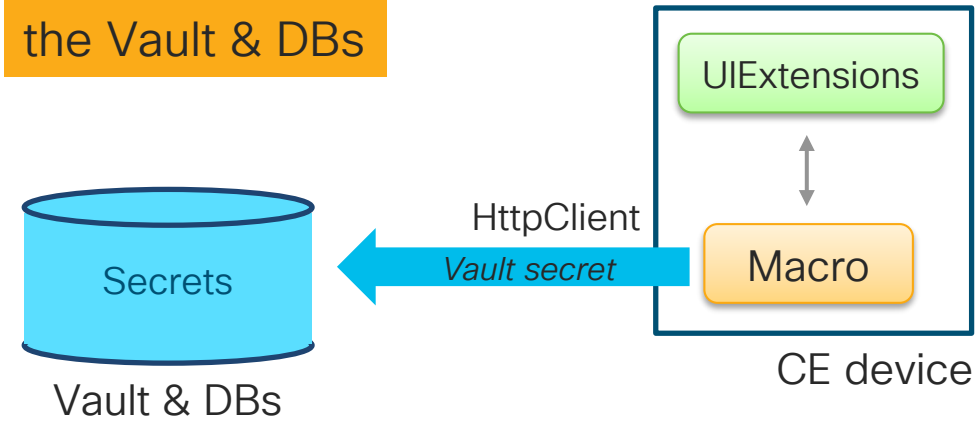


/!\ static re-run CI/CD as changes are needed

Deployment strategies

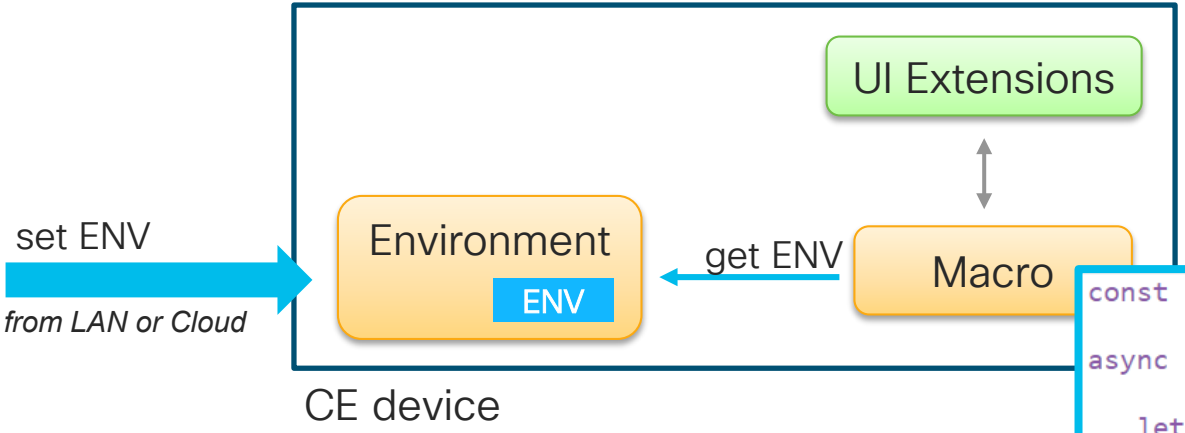
Load topology and secrets dynamically

/!\ reliability
LAN access to
the Vault & DBs



Deployment strategies

Pass values as environment variables




```
const xapi = require('xapi');

async function init(ENV) {

   let value = await ENV('DEVICE_SECRET');
}
```

/!\ prototype based on 'xCommand Message Send'

https://github.com/ObjectIsAdvantag/macros-env

Beyond Macros

Extend CE with your own protocols

```
xapi.command('Message Send',
{ Text: 'PROTOCOL_FORMAT' })
```

Macro

notifies

listens

Messages Bus (local to the device)

- Use cases
  - Communications inter-macros
  - Receiving notifications from external services

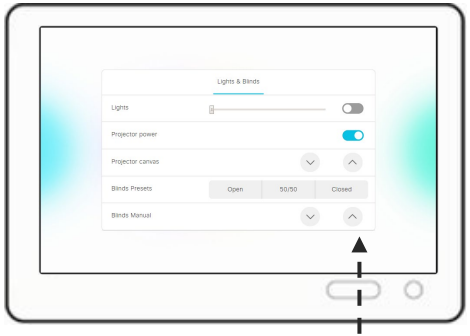
Creating and Mapping Accounts to User Roles

Roles	Role Description	Device Accounts
Admin	Have unrestricted access to the device's local web interface. Access the API over SSH, serial connection, or HTTP(S). Create and manage users with the *Integrator* and *RoomControl* roles.	- CI/CD pipeline - Ops team - Standalone App1
Integrator	Access the device's local web interface. *Integrator* has the same access as an *Admin* user, except creation of new users. Access the API over SSH, serial connection, or HTTP(S).	- Standalone App2
RoomControl	Access the In-Room Control editor and corresponding development tools on the web interface, to create touch interface extensions (in-room controls).	- Support team

Wrapup

CE Programmability (xAPI)

Touch10 Interface



Standalone
(JavaScript, Python...)

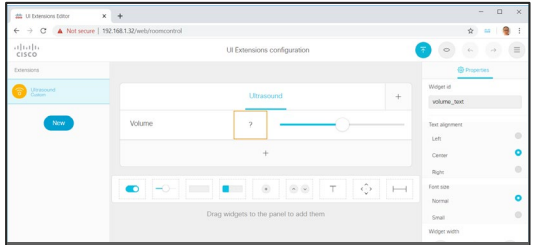
Macros
(JavaScript)

- status
- configure
- command

*over ssh, Websocket, serial, or HTTP**

events

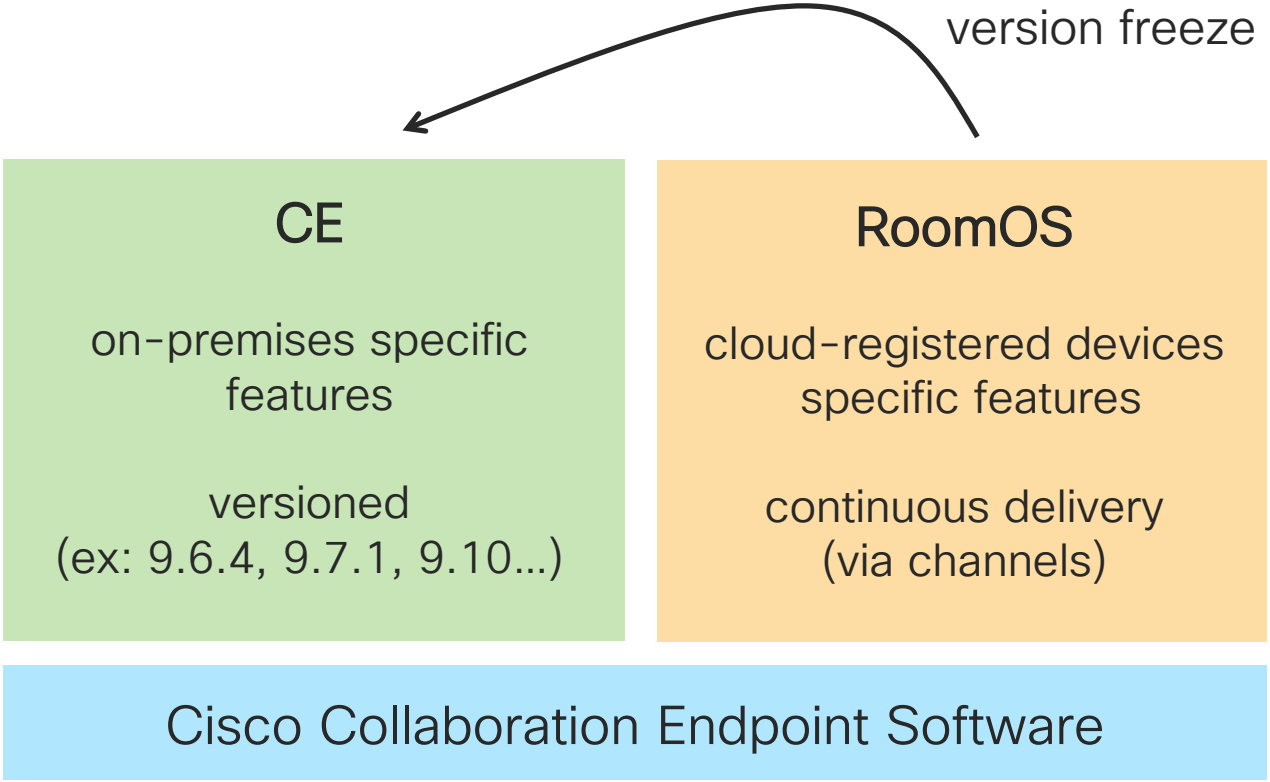
deploy



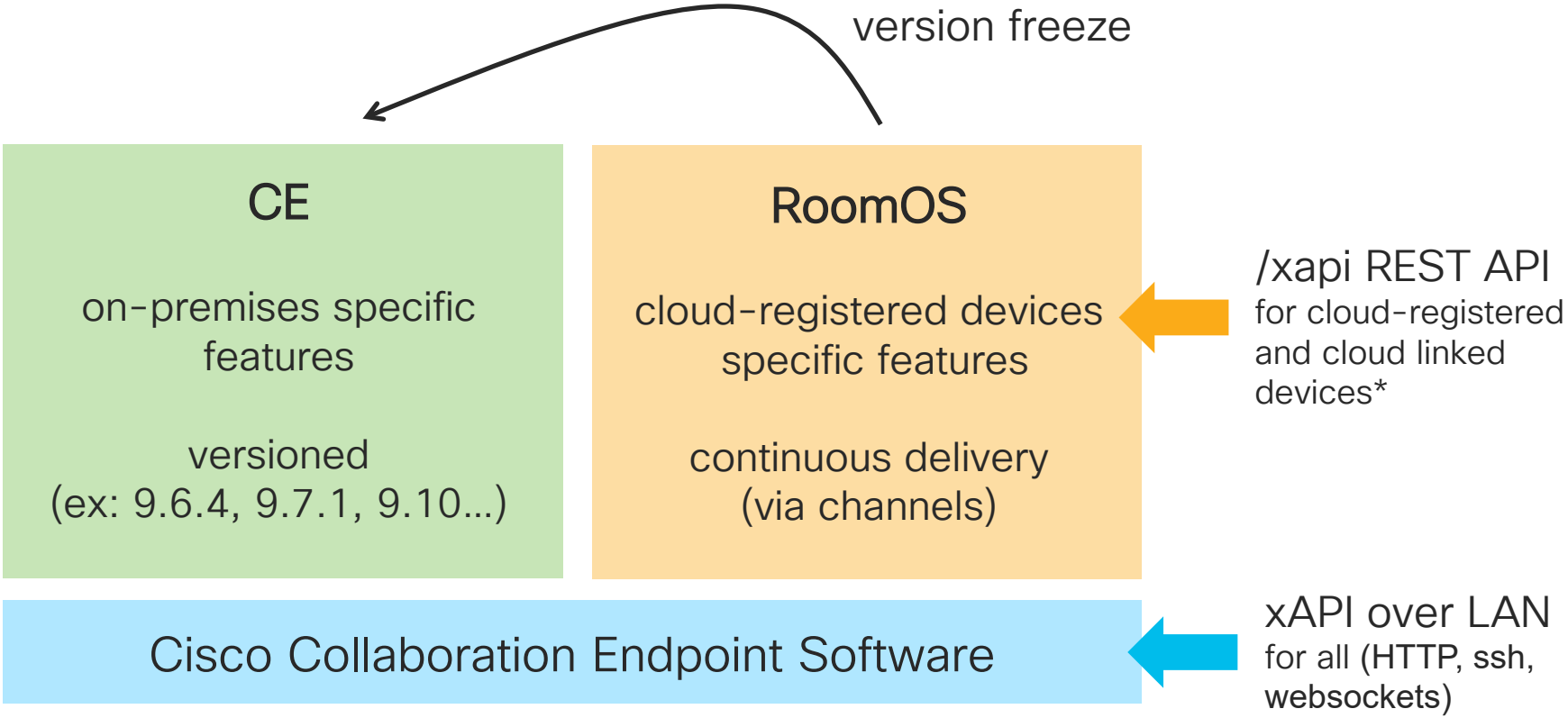
UI Extensions Editor

* HTTP is unidirectional (cannot listen to events)

CE and RoomOS



CE and RoomOS



** See also BRKCOL-2241*

Webex Device Programmability & APIs

- Tuesday, 10:00 – ‘Meet DevNet’ podium - DEVNET-3010.f
  - Learn how to make Network Automation Simple with the Community
- Wednesday, 12:15 – Hall 8, C128 - BRKDEV-3244
  - Advanced coding for Cisco Video devices
- Wednesday, 15:00 – ClassRoom 3 - DEVNET-2071
  - Customizing Cisco Collaboration Devices
- Thursday, 10:00 – DevNet Theater - DEVNET-1462
  - Webex Room Device APIs (latest features
- Friday, 11:30 - BRKCOL-3008
  - Customization and Integrations of Cisco Video Room Devices

Awesome xAPI

awesome | DEVNET | published

https://github.com/CiscoDevNet/awesome-xapi

A curated list of developer resources for **Webex Room and Desk Devices** inspired by awesome-go and awesome-python.

> Looking for developer resources for **Webex Teams**? check awesome-webex.

Contents

DISCLAIMER: Cisco does not make any commitments about the resources listed in this document, nor the accuracy of the third party resources and any content accessible via the links below.

- !Get Started!
- Articles and Blogs
- Building Blocks
- Code samples
- Developer Tools
  - DevNet Sandbox
- Reference
  - PDF Guides
- 3rd Party Hardware

Join the 'xAPI Devs' Teams Space

http://bit.ly/join-xapi-devs

Learn more about the new DevNet Certifications and how you can prepare now!

	Associate Level	Specialist Level	Professional Level	Expert Level
Engineering	CISCO CERTIFIED CCNA	CISCO CERTIFIED SPECIALIST	CISCO CERTIFIED CCNP	CISCO CERTIFIED CCIE
Software	CISCO CERTIFIED DEVNET Associate	CISCO CERTIFIED DEVNET SPECIALIST	CISCO CERTIFIED DEVNET Professional	CISCO CERTIFIED DEVNET Expert

Future Offering

Complete your online session survey

- Please complete your session survey after each session. Your feedback is very important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (starting on Thursday) to receive your Cisco Live t-shirt.
- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Content Catalog on ciscolive.com/emea.

Cisco Live sessions will be available for viewing on demand after the event at ciscolive.com.

Continue your education

Demos in the Cisco Showcase

Walk-In Labs

Meet the Engineer 1:1 meetings

Related sessions

Thank you

CISCO

You make **possible**