Webex Devices Integrations - High level overview

Cisco Webex Devices supports a large and varied API set. All components are well documented by themselves, but it can be overwhelming for customers and developers to understand which API technology to choose, and why.

This guide gives a brief overview of the most important components ("puzzle piece"), and some tips on when to choose one over the other. For each of the component, there is a runnable example as well.

Mini dictionary

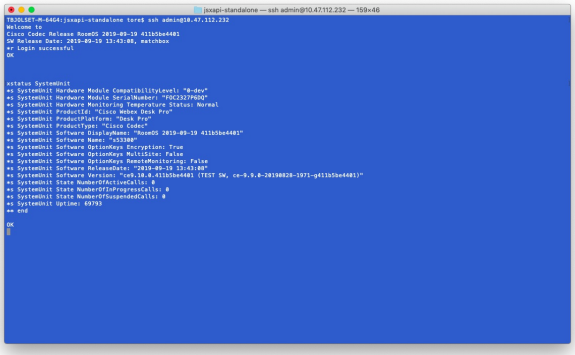
- xAPI - the core protocol used to manipulate the video system, such as making calls, adjusting volume etc
- Macros - snippets of JavaScript code that customers can write that runs on the video system itself
- jsxapi - JavaScript SDK for the xAPI, open source and available for eg Node.js
- Cloud xAPI - RESTful xAPI access for cloud registered devices
- User interface extensions - panels, buttons and widgets that can be added to the user interface of the video device to allow user to control lights, blinds, make speed dials etc
- In-room controls - the previous name for User interface extensions
- Web apps - web pages running on the newer Webex devices with the Chromium web engine
- xAPI over web socket - web browsers etc can speak directly to the xAPI over web socket, if it knows the username and password for the video system
- Bot apis - rest apis and web hooks for sending/receiving bot messages on Webex Teams

For each integration type, you can read:

- How it works
- An example, and screenshot
- When to use/not use it
- Which devices support it
- Required configs, flags, permissions, software versions
- Link to more documentation

xAPI

The xAPI is the language for speaking to the video system. All integrations, and also Cisco's internal code such as the user interfaces, uses the xAPI to communicate with the video system and retrieve status data.



Examples

The easiest way to play with the xAPI is to login with TShell from the command line. Here's how you can start a call using the xAPI and SSH from your laptop:

- Open a console terminal
- Connect to your video endpoint (replace the ip addresse with your video devices addresse):

```
ssh admin@10.47.90.231
```

- Once you are in, you can start browsing the api tree with autocomplete (use the tab key often)
- `xCommand`, `xStatus` and `xConfig` are good starting points
- To call a number that shows a fireplace:

```
xCommand Dial Number: fireplace@ivr.vc
```

- To see the number of active calls (verify that active calls is 1):

```
xStatus SystemUnit State
```

- You can register feedback to be notified any time a value changes:

```
xFeedback Register status/systemunit/state
```

- Observe that you now get a message any time a call is started or stopped.

That's a super short overview of the main features of the xAPI. You can use the xAPI to control and observe almost anything that the video system supports, such as starting calls, adding participants, doing screen share, controlling camera, adjusting volume, muting, changing video layout etc.

The best way to get the xAPI is to just play with it from the command line. Try to think of work flows that you would like to automate (start a call, add a third participant, adjust the volume to a certain setting, turn on sticky self view, choose equal layout etc) and see if you can do it all from the keyboard.

For a full reference guide, see:

https://www.cisco.com/c/en/us/support/collaboration-endpoints/spark-room-kit-series/products-command-reference-list.html

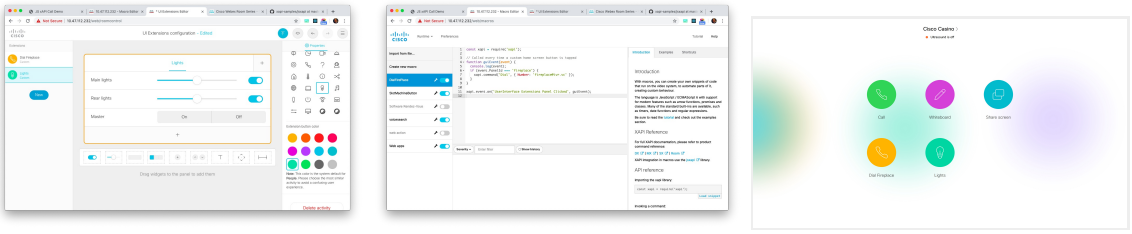
Macros and user interface extensions

Macros are snippets of code (scripts) written in JavaScript that can run on the video system itself, to customise the behaviour of the video system. A benefit of this is that you do not need any additional hardware, such as virtual machines or mini servers. Macros are typically written and tested in the macro editor, which can be accessed on the web interface of the video system itself.

A typical use case is for the macros to listen for events from custom user interface extensions. These extensions can be buttons and sliders for controlling peripherals in the room such as lights, blinds, climate controls, projectors, or to modify the behaviour of the video system to suit particular work flows, for example adding quick dials to the home screen.

Since CE 9.7 (TODO VERIFY), the macros also support communicating with with the external world using HTTP GET, POST etc, greatly increasing their usability.

Supported devices: All devices running CE 8 and higher, except SX10 and Webex Share (TODO VERIFY)
Since: CE (TODO Find out).



UI Extensions editor, macro editor and custom home screen

Example:

The following example starts a call on the video system to a fireplace. The action is triggered when a button with panel id `fireplace` is pressed.

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

// Called every time a custom home screen button is tapped
function guiEvent(event) {
  if (event.PanelId === 'fireplace') {
    xapi.command('Dial', { Number: 'fireplace@ivr.vc' });
  }
}

xapi.event.on('UserInterface Extensions Panel Clicked', guiEvent);
```

See the `macro-user-extensions` example in the Git repo for the UI Extensions file and the macro.

Documentation: TODO link

jsxapi and Node.js

The same JavaScript that was used in the macro above can also be run on an external Node server with almost no modifications, for example in a virtual machine or a Raspberry Pi. For this, we recommend the jsxapi, which are JavaScript bindings (SDK) for talking to the video system.

Requirements for this solution:

- Your integration can reach the video system on the network
- The network allows SSH
- Your integration can have user access (typically admin user or integrator user)

The main benefits over a macro integration:

- You can use system libraries (SDK for external web services, interact with non-IOT peripherals in the room)
- Your integration is centralised, so if you need to update it often, you don't need to update a macro on each endpoint
- You can use third party libraries, eg to control hardware in your room, machine learning, screen scraping, etc

Example

For this example to work, you need to have Node and npm installed. Search the web for how to install for your operating system if you don't have it already.

Make sure the jsxapi bindings are downloaded and installed by typing `npm install` in the root folder of the Git repo.

Starting a call now from an external server is easy, if you have the username and password to log on to the video system:

```
// Import the library for talking with the xAPI
const jsxapi = require('jsxapi');

// Replace with your credentials
const codec = {
  host: '10.47.112.232',
  username: 'admin',
  password: 'host',
};

// Connect to the video system
const xapi = jsxapi.connect('ssh://' + codec.host, {
  username: codec.username,
  password: codec.password,
});

// Start a call
xapi.command('Dial', { Number: 'fireplace@ivr.vc' });
```

Test the runnable example in the `jsxapi` folder, edit your codec settings in `main.js`. It should connect, call the fireplace then automatically disconnect and quit. To run it, cd to `jsxapi` folder in your shell and type:

```
node main.js
```

The jsxapi should work with any video system that has xAPI.

Required configs:

```
xConfiguration NetworkServices Mode: On
```

jsxapi with Express web server