Webex Help Center: August 21, 2024

Use of third-party USB input devices with Board, Desk and Room Series

You can use a third-party USB peripheral to control certain functions on your device, such as a USB keyboard or a Bluetooth® remote control with a USB dongle.

This feature is meant to *complement* the functionalities of the touch controller or the touch user interface. It is not meant to *replace* them.

Examples of applications:

- In classrooms and during lectures, a small remote control can be used to wake up a device from standby mode. it may also be convenient to use a remote control to select which input source to present.
- Controlling the camera view (pan, tilt, and zoom) in situations where you are not allowed to use the touch controller. For example, in operating rooms in a hospital.

Functional overview

When a button is pressed on the USB input device, it triggers an action within Cisco device's API. Macros or external control devices from third parties can be set up to detect these actions and react accordingly. This functionality is similarly to how In-Room Control buttons behave. Additionally, it's feasible to monitor these actions through webhooks or directly within an SSH session.

A pre-existing library of actions to choose from is not provided. You are required to define and establish the specific actions that should occur in response to the events. For example:

- Increase the volume of the Cisco device when the Volume Up key is pressed.
- Put the Cisco device in standby mode when the Sleep key is pressed.

Configurations, Events, and Status

The configurations and status that are referred in this article, are available both from the local web interface of the device and the APIs. Read the Device configurations article for information how to access the web interface and use the API.

When the web interface of the device is opened, click on Settings . Under Configurations , change Peripherals > InputDevice Mode to **On**. The support for third-party USB input devices is disabled by default.

Pressing and releasing a button generates a *Pressed* and a *Released* event:

*e UserInterface InputDevice Key Action Key: <name of the key> *e UserInterface InputDevice Key Action Code: <id of the key> *e UserInterface InputDevice Key Action Type: Pressed ** end *e UserInterface InputDevice Key Action Key: <name of the key> *e UserInterface InputDevice Key Action Code: <id of the key> *e UserInterface InputDevice Key Action Type: Released ** end

To listen for events, you must register feedback from the *InputDevice* events:

```
xFeedback Register /event/UserInterface/InputDevice ** end
```

When the Cisco device detects the third-party peripheral, it will be listed under Status and in Peripherals > ConnectedDevice . The third-party device may be reported as multiple devices.

Further information

Find more information about the use of a third-party input device in the Customization guide . Choose the latest version.

Cisco support (TAC) doesn't support debugging of third-party code, including macros. Please check the Cisco RoomOS for Collaboration Devices if you need help with macros and third-party code. Check this page for more examples of macros and extensions.

Example

In this example, we want to show you how to use the keys of a third-party USB input device (in this case a remote control) to control certain functions on a Cisco device.

We show you how to use the buttons on a Bluetooth remote control (connected through a USB dongle) to manage functions like standby, volume adjustment, and control of a Cisco camera device. You can develop a macro that listens to relevant events and executes corresponding actions through the Cisco device's API.

In the following example, you must enter the text that is written in normal font. The text in italics is the response received from the Cisco device.

1. Sign in to the Cisco device on SSH. You need a local *admin* user.
2. Configure the device to allow the use of a third-party USB remote control.

```
xConfiguration Peripherals InputDevice Mode: On
** end OK
```

   You can check if the configuration is On or Off by using this command:

```
xConfiguration Peripherals InputDevice Mode
*c xConfiguration Peripherals InputDevice Mode: On ** end OK
```

3. Register for feedback, so that we are notified when the remote control buttons are pressed and released.

```
xFeedback Register /event/userinterface/inputdevice
** end OK
```

You can check which feedbacks the device is registered for using this command:

```
xFeedback list
/event/userinterface/inputdevice ** end OK
```

4. Press and release a button on the remote control to check that feedback registration works.

This action generates two events: *Pressed* and *Released*. If you press and hold a button, you see the *Pressed* event until you release the button. Then the *Released* event is generated.

These events are issued when pressing and releasing the Enter key:

```
*e UserInterface InputDevice Key Action Key: KEY_ENTER *e UserInterface InputDevice Key Action Code: 28 *e UserInterface InputDevice Key Action Type: Pressed ** end *e UserInterface InputDevice Key Action Key: KEY_ENTER *e UserInterface InputDevice Key Action Code: 28 *e UserInterface InputDevice Key Action Type: Released ** end
```

5. Write a macro that listens for the relevant *InputDevice* events, and carries out the associated actions using the API of the device.

- Bring the standby, volume up and volume down buttons to life. When the macro sees an event containing KEY_VOLUMEUP, KEY_VOLUMEDOWN, or KEY_SLEEP, it executes the related commands.
- Create a camera control function for the arrow keys. We want to keep moving the camera as long as the button is pressed. When the button is released, the camera movement stops. When the macro sees an event containing KEY_LEFT, KEY_RIGHT, KEY_UP, or KEY_DOWN, it executes the related commands.

```
const xapi = require('xapi'); function com(command, args='') { xapi.command(command, args); log(command + ' ' + JSON.stringify(args)); } function log(event) { console.log(event); } function notify(message) { xapi.command('UserInterface Message TextLine Display', { Text: message, duration: 3 }); } function cameraControl(motor, direction, cameraId='1') { com('Camera Ramp', { 'CameraId': cameraId, [motor]: direction }); } function init() { let standbyState; xapi.status.get('Standby').then((state) => {standbyState = state.State === 'Off' ? false : true; }); xapi.status.on('Standby', state => { standbyState = state.State === 'Off' ? false : true; }); xapi.event.on('UserInterface InputDevice Key Action', press => { if (press.Type == "Pressed") { switch (press.Key) { case "KEY_LEFT": cameraControl('Pan', 'Left'); break; case "KEY_RIGHT": cameraControl('Pan', 'Right'); break; case "KEY_UP": cameraControl('Tilt', 'Up'); break; case "KEY_DOWN": cameraControl('Tilt', 'Down'); break; default: break; } } else if (press.Type == "Released") { switch (press.Key) { case "KEY_LEFT": cameraControl('Pan', 'Stop'); break; case "KEY_RIGHT": cameraControl('Pan', 'Stop'); break; case "KEY_UP": cameraControl('Tilt', 'Stop'); break; case "KEY_DOWN": cameraControl('Tilt', 'Stop'); break; case 'KEY_VOLUMEUP': com('Audio Volume Increase'); break; case 'KEY_VOLUMEDOWN': com('Audio Volume Decrease'); break; case 'KEY_SLEEP': com(standbyState ? 'Standby Deactivate' : 'Standby Activate'); break; default: break; } } }); } init();
```