Customization Guide

for Cisco Webex devices running RoomOS 10.3

Thank you for choosing Cisco!

This user guide is designed for administrators working with the setup and configuration of the video conferencing device.

The top menu bar and the entries in the table of contents are all hyperlinks. You can click on them to go to the topic.

User guides, compliance, and safety information for Cisco TelePresence systems are available at:

► https://www.cisco.com/go/telepresence/docs

For information about devices that are registered to the Cisco Webex cloud service, visit:

► https://help.webex.com

Visit the Cisco web site regularly for updated versions of the documentation.

Table of contents

Chapter 1

Introduction

Customization Opportunities

There are many ways to customize your video device.

To begin, log-in to the web interface as an Administrator and, in the left pane, select *Customization > Personalization*. From there, you can select a new background image for your device.

Video devices which support customization are:

- Cisco Webex Board series
- Cisco Webex Desk series
- Cisco Webex Room series

Note: Video devices must be running RoomOS or Collaboration Endpoint Software, version CE9.8 or newer.

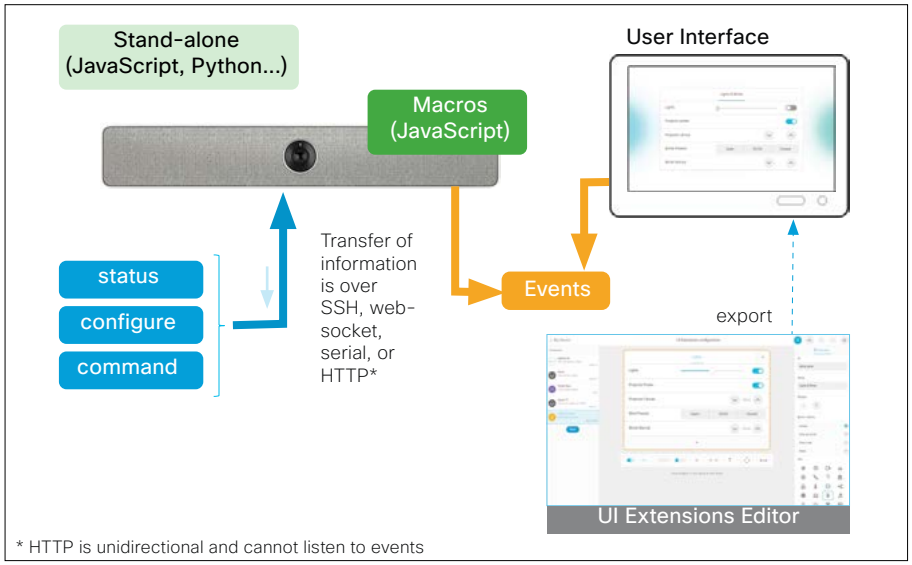
You can change your video device configurations through the website or through a variety of other ways, including:

- Terminal mode (SSH)
- XML
- HTTP/HTTPS
- xAPI
- JSXAPI
- Macros
- Websockets

The open platform enables you to create customizations that attach to your existing workflows; thus, allowing you to create additional value beyond the core Webex offering.

Some examples include:

- problem reporting to a ticketing system
- getting data from a webserver to display on the panel
- collecting acoustics data from the room and sending to a webserver where it can be plotted
- controlling third-party peripherals, such as lights, blinds, and video switches



Programmability through the xAPI

Definition of Terms

Action Button. An action button is a simple button that you can add to the user interface. When pressed, it will execute a command, such as to dial a predefined number.

Control system. A control system is third-party system with hardware drivers for peripherals (e.g., Crestron, AMX, Raspberry Pi).

Touch controller. Touch controller refers to the Cisco touch-based control device used with all products except the Desk Series. The touch controller can be either a Cisco Touch 10 or a Cisco Webex Room Navigator.

Macro. Macros are short scripts written with JavaScript and xAPI commands.See the Macros section to learn more.

Macro Editor. The **Macro Editor** is a code editing tool that allows you to create and debug custom macros. See the A Tour of the Macro Editor section for more information.

Panel. A panel is a custom-created grouping of controls (e.g., buttons, sliders, switches) that you can add to the user interface. The panel opens when you touch the corresponding control icon. See the Panels section for more information.

User Interface. The user interface refers to the touch screen for a video device. The touch screen may be embedded, such as for the Webex Board and Desk Pro, or it may be external, such as for the Room Navigator.

User Interface Extensions Editor. The **User Interface Extensions Editor** is an easy-to-use editor that allows you to create custom buttons and panels on the user interface. See the A Tour of the UI Extensions Editor section for more information.

Video device. A video device refers to a Cisco video conferencing system (e.g. Webex Board, Desk Pro).

WebApp. A WebApp is a an application that shows an internet page on the device. This can be programmed by adding a button to the user interface. When pressed, a predefined website will be launched.

Widget. A widget is a control located on a panel. Widgets may include switches, buttons, sliders, text, and others. See the Widgets section for more information.

xAPI. The xAPI is the Application Programming Interface (API) of the video device. The xAPI facilitates communication between the video device and third-party applications. See the Application Programming Interface (API) chapter for details.

Chapter 2

User Interface Extensions

What are User Interface Extensions?

The Webex series video devices include a **User Interface (UI) Extensions Editor** that allows you to expand the tools on the touch screen of your video conferencing device.

Video devices which provide the **UI Extensions Editor** are:

- Cisco Webex Board series
- Cisco Webex Desk series
- Cisco Webex Room series

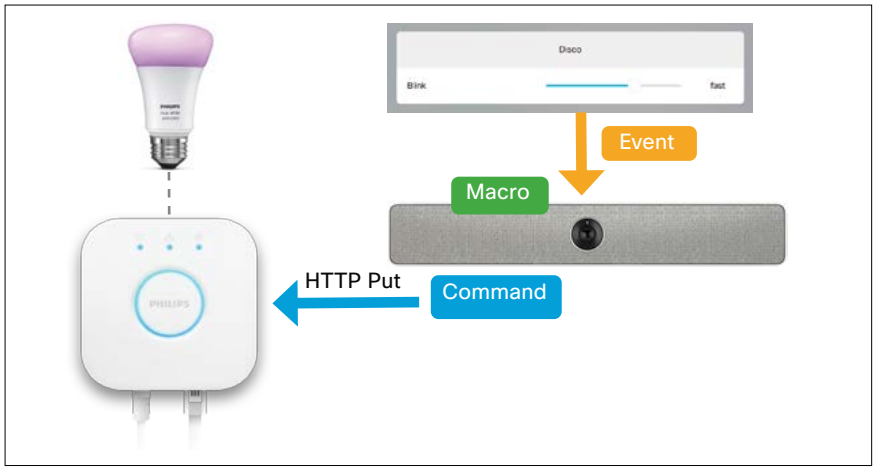
The simple drag-and-drop editor offers a library of user interface elements, referred to as *widgets*. You can use these widgets to create your own control panels.

UI Extensions, such as buttons and panels, can be programmed in an infinite number of ways.

For example, you could create a custom panel for controlling music, as shown here.

Custom panel on the User Interface

You could program the system to control the flashing of a light, as shown here.



Custom Action through xAPI and HTTP

You could create a custom panel with text fields through which users can submit information. The information can be transmitted to a server and stored, processed, and/or displayed.

This chapter provides simple examples of programming and monitoring each widget. More examples and links to Online Resources can be found in the Examples chapter.

Creating UI Extensions

To access the **UI Extensions Editor**, sign-in to the video device's web interface with administrator credentials and navigate to *Customization > UI Extensions Editor*.

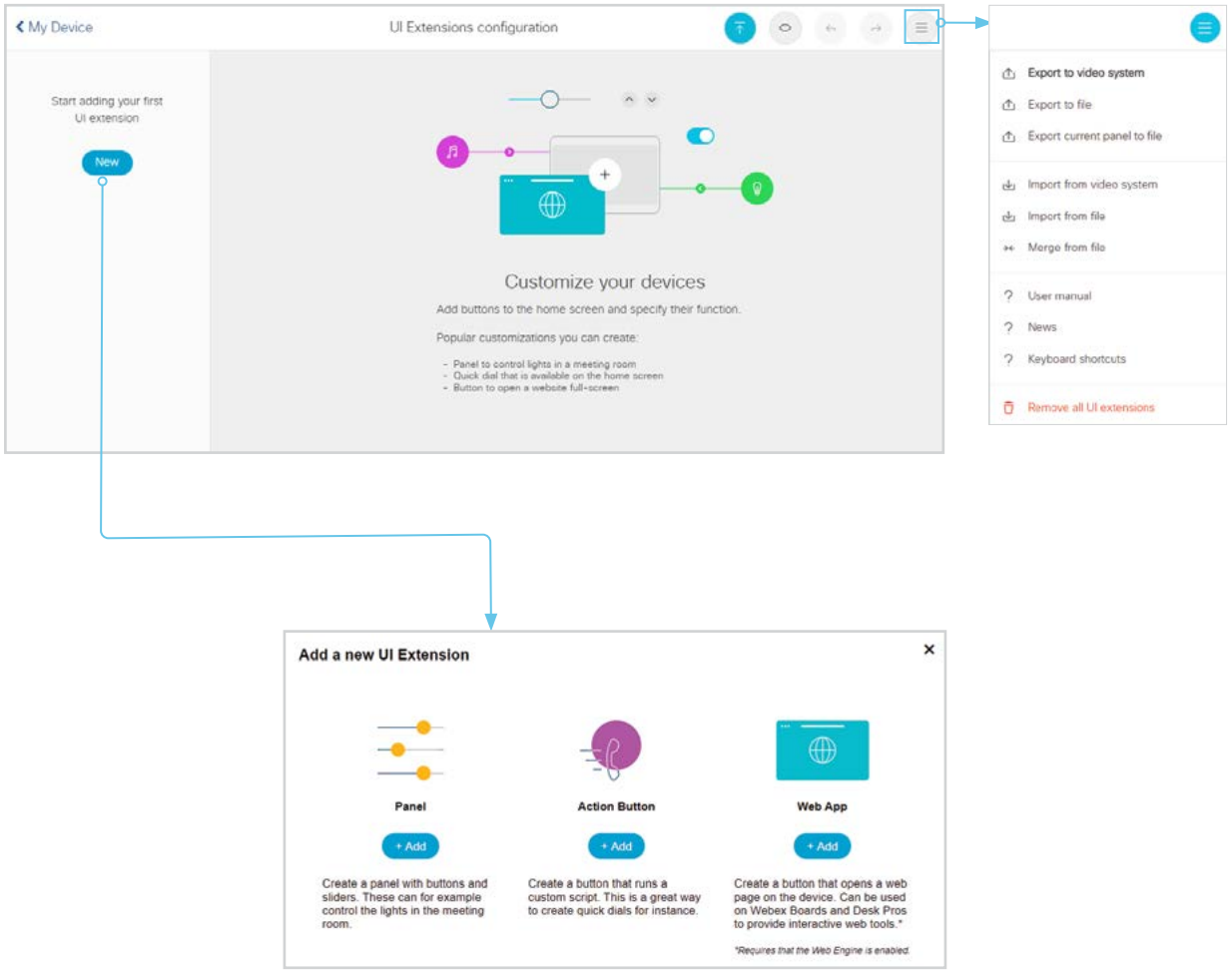
An administrator can create a user account for non-administrators, providing access limited to the **UI Extensions Editor**.

Click *New* and the **Add a new extensio**n dialog box will appear.

You will be presented with the following options.

- *Panels* – Control panels that contain a number of widgets (e.g., sliders, switches, buttons, etc). See the Panels section for more information.
- *Action buttons* – Simple buttons that execute a command when pressed (e.g., dial a number). See the Action Buttons section for more details.
- *WebApps* – (Only for video devices with a WebEngine) A button launches a web view in full screen on the user interface. See the WebApps section for more details.

Each of these will add a new button to the user interface. Only a few buttons will be added to the main page before it runs out of space. To access the overflow buttons, swipe from right to left in the screen area where the icons are displayed.

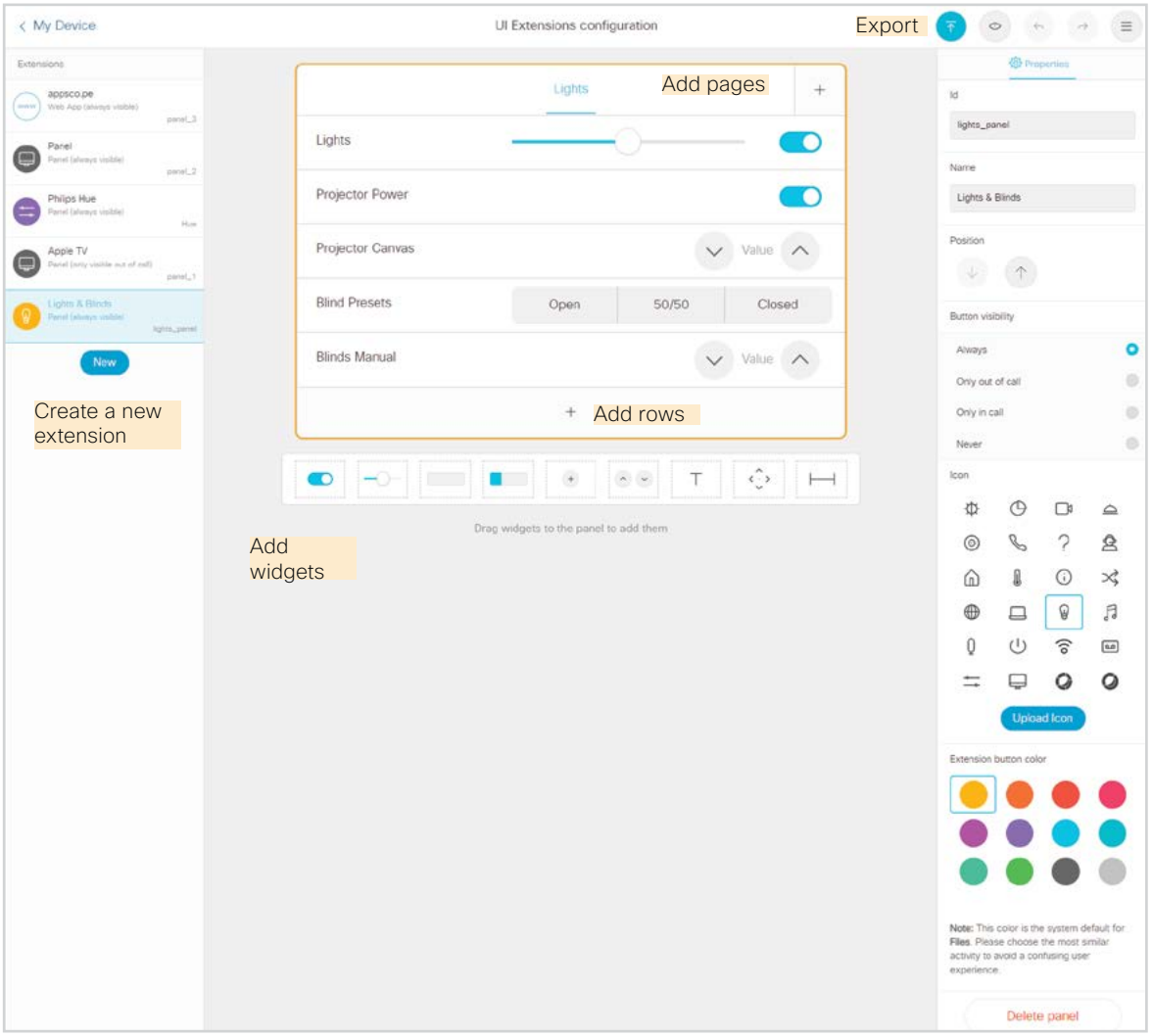


A Tour of the UI Extensions Editor (page 1 of 5)

To get started, create a new panel and drag some widgets into the panel.

You can change the default names by double-clicking on them and entering new text. Click *Enter* to apply the change.

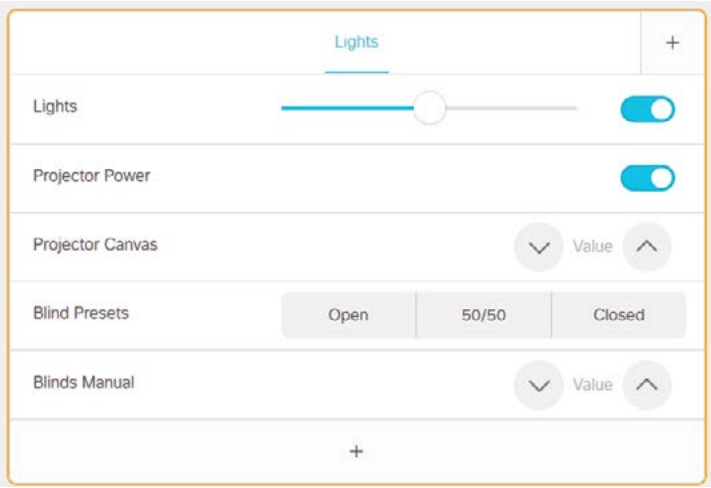
When you are ready to see your changes on the device, click the .



A Tour of the UI Extensions Editor (page 2 of 5)

You can access the properties of a panel or widget by clicking on its title.

A yellow frame wil be displayed around the item to indicate that it is selected and the *Properties* pane will display the settings.



The *Properties* pane contains the following settings:

Id - The identifier of the item. This is be used by the API to run commands or monitor events for the item.

Name - The text you put in the *name* field is displayed near the widget or in the panel heading.

Position - The *position* arrows are used to change the order of the buttons on the user interface. The up arrow moves the button earlier in the order.

Extension is available - This field specifies when the item will be available.

- *Always* - Available both in-call and out-of-call
- *Only out of call* - Available only outside calls
- *Only in call* - Available only during calls
- *Never* - Hidden

To see the buttons during a call:

- On Boards: To see the *Only in call* buttons during a call, tap the screen. To see the *Always* buttons during a call, tap the *Home* button.
- On Desk Series devices: Tap the screen during a call to see the button.

Icon - The *icon* is the image displayed on a button. Select one of the default icons or upload your own.

Color - The *color* of the button. A limited color palette used for standard buttons is available in the editor. When you select a color, a small description of the context in which this color is used by Cisco, will be provided.

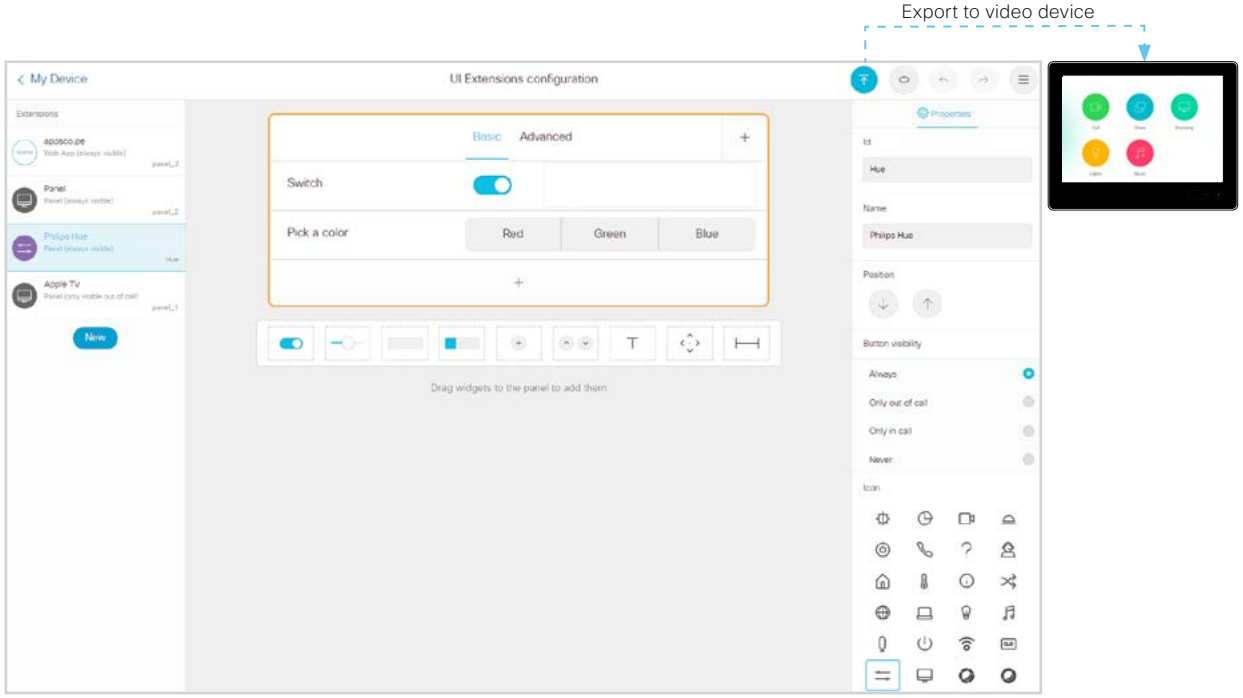
Note, the color cannot be set for buttons on the In-call user interface. These buttons are always black.

A Tour of the UI Extensions Editor (page 3 of 5)

There are several buttons in the header area of the Editor:

- Export your changes to the user interface.
- Show a preview of your changes.
- and Undo and redo changes.
- Show additional options.

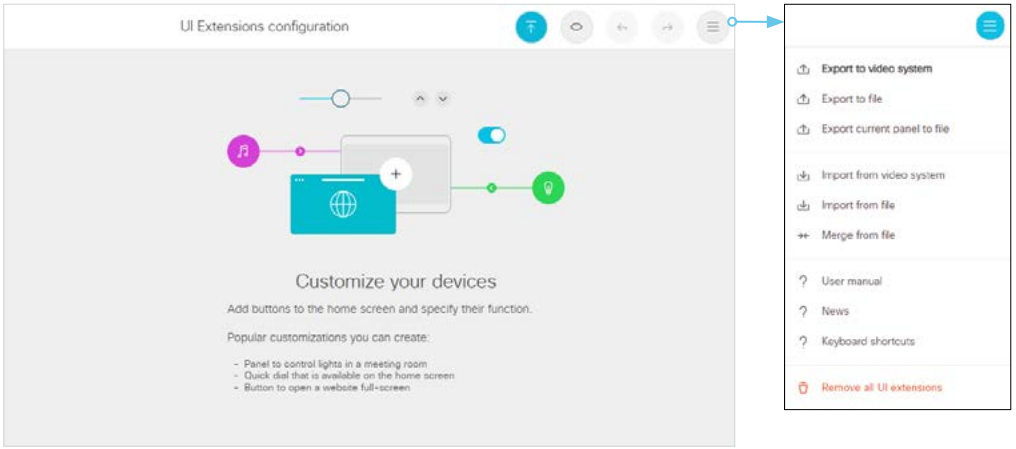
To exit, click the Cisco logo in the top left corner.



A Tour of the UI Extensions Editor (page 4 of 5)

The icon in the top-right corner of the **UI Extensions Editor** shows several important options.

- *Export to video system* – Export the UI extensions from the editor to the user interface. This will overwrite the existing custom extensions on the video device. This is the same behavior as when you click the button.
- *Export to file* – Export the UI Extensions from the editor to an XML file.
- *Export current panel to file* – Export only the configuration for the currently selected panel to an XML file.
- *Import from video system* – Get the configuration for the user interface of the video device and apply it to the editor. If you have unsaved changes in your editor, these will be erased.
- *Import from file* – Import an offline configuration as an XML file. If you have unsaved changes in your editor, these will be erased.
- *Merge from file* – Import an offline configuration as an XML file and append it to the current configuration in the editor. Note that any panels with the same name will then be overwritten.
- *User manual* – Open a web page with links to several versions of the user documentation.
- *News* – See information about changes from recent releases.
- *Keyboard shortcuts* – See a list of the commonly used keyboard shortcuts for use with the editor.



Ctrl-Enter Export configuration to video system

Ctrl-Space Preview current configuration

Ctrl-S Save configuration to file

Ctrl-O	Open configuration from file
Ctrl-Z	Undo last action
Ctrl-Shift-Z	Redo last action
Ctrl-Shift-C	Copy selected component
Ctrl-Shift-X	Cut selected component
Ctrl-Shift-V	Paste selected component
Ctrl-D	Duplicate selected component
Backspace / Delete	Delete selected component

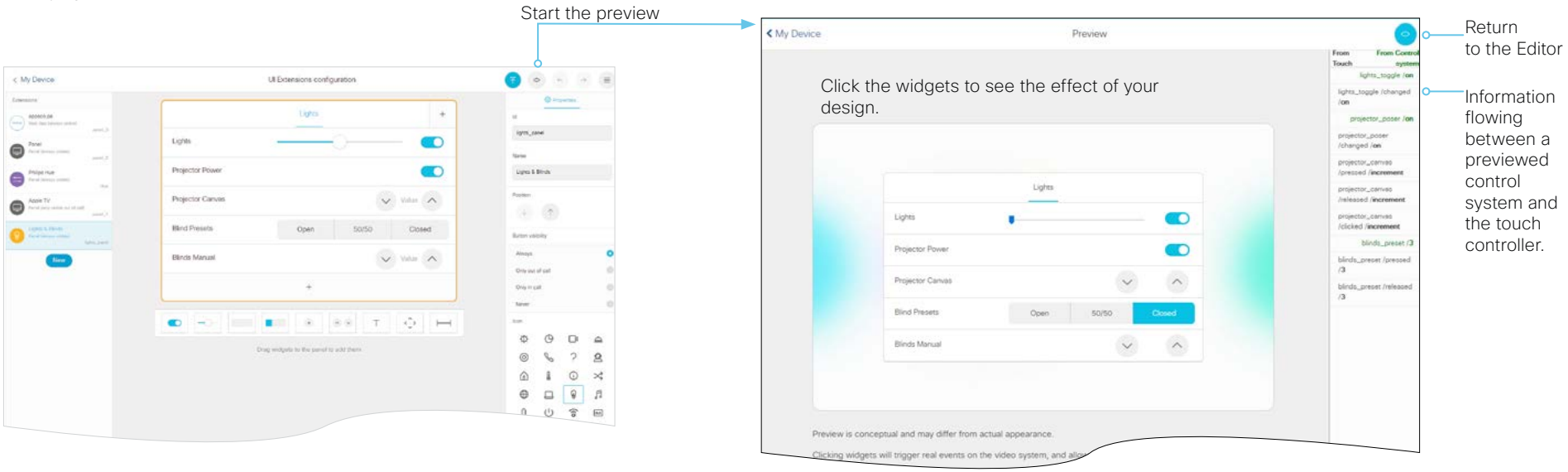
For Mac users, replace `Ctrl` with `Cmd`.

- *Remove all UI extensions* – This will clear the editor, but not the video device. To undo this change, select *Import from video system*. To push this change to the video system, select *Export to video system*.

To close the menu, click the menu icon.

A Tour of the UI Extensions Editor (page 5 of 5)

You may preview your configurations to verify them before deploying them.



Note! The preview works for all the devices, but it will look it as if all of it has been created for a touch controller.

The above provides a preview of your configuration with a simulated third-party control system connected.

When implementing your configurations (a real situation scenario), make sure your control system has been set to send `SetValue` commands wherever applicable.

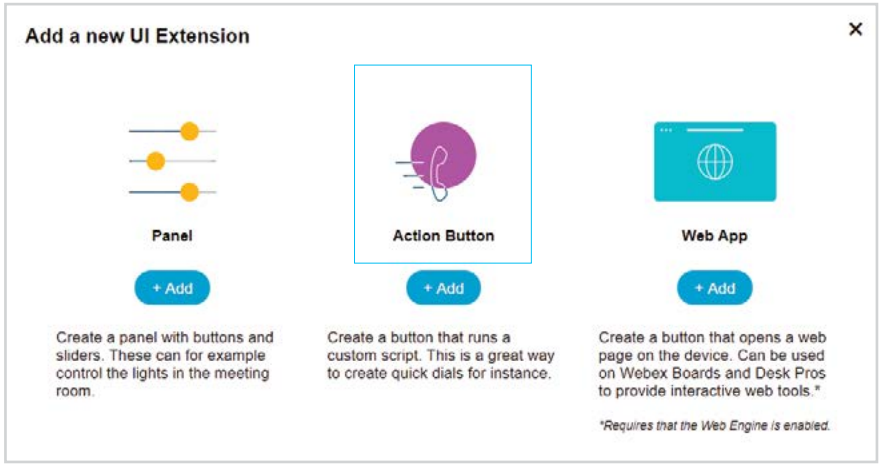
Example: If you set Lights to `On` in a real situation scenario, the touch controller needs to receive feedback confirming that the lights actually are switched on. For this to take place, the controller must switch on the lights and then send a `SetValue`, confirming the change of the lights settings. The right pane of the above example shows a simulation of what the touch controller sends to the control system and what the control system then sends back to the touch controller.

In a real situation scenario, you should also make sure that the control system sends a `SetValue` to the touch controller whenever someone operates the light switch on the wall in the meeting room.

Action Buttons

If you choose to create an action button, you will be adding a button to the user interface. When the user presses the button, a single action will be performed (e.g., dialing a number).

You must program this action by using a API command. It is simple to create this program by using the **Macro Editor**. See the Macros chapter for more information.



You can select the color and image for the icon; as well as, the position on the screen.

Click the button to export your button and view it on the user interface.

Example of an Action Button

The following steps will create an action button that displays a message:

1. Create an action button with id of "hello1_button" (without the quotation marks).
2. Click the button to export your action button to the video device.

   Look at the user interface. You should now see the action button. Nothing will happen if you press it, because we have not programmed the button yet.
3. You will need to set up a command to run when the button is pressed.

   Open the **Macro Editor** and click *Create new macro*. The text editor will open. Add the following script after the inital text:

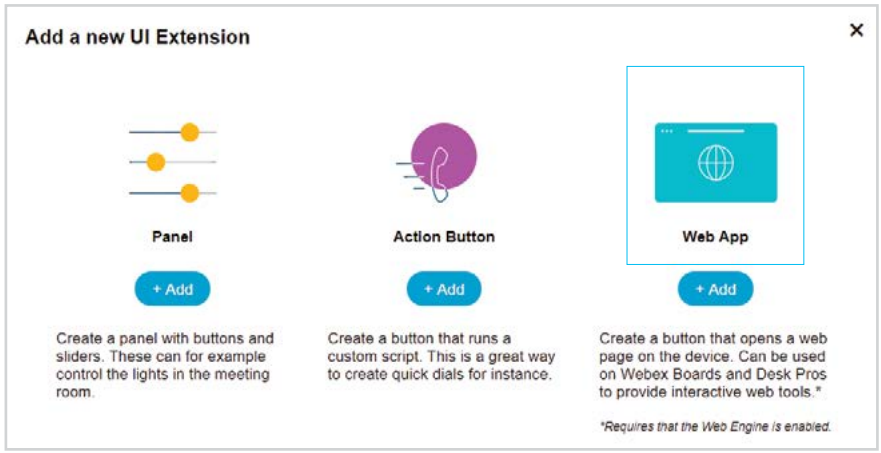
```
xapi.event.on('UserInterface Extensions Panel Clicked', (event) => {
  if (event.PanelId === 'hello1_button') {
    xapi.command('UserInterface Message Prompt Display', { Title: 'Hello!',
text: 'Have a great day!'} );
  }
});
```

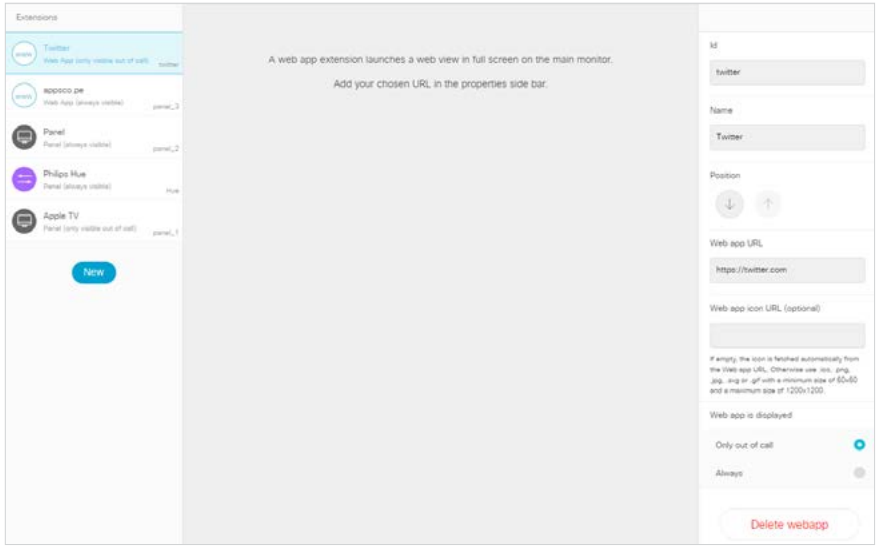
4. Type `Ctrl+s` to save the macro.
5. *Enable* the macro. This is done by setting the corresponding switch to *on*.
6. Now, on the video device, test the button. It should show you a friendly message.

WebApps

WebApps can only be used on devices with a Web Engine (e.g., Webex Board and Desk Pro). The Web Engine must be enabled to create the WebApps.

A WebApp is simply a button on the home page. When users press this button, a web view will launch in full screen.



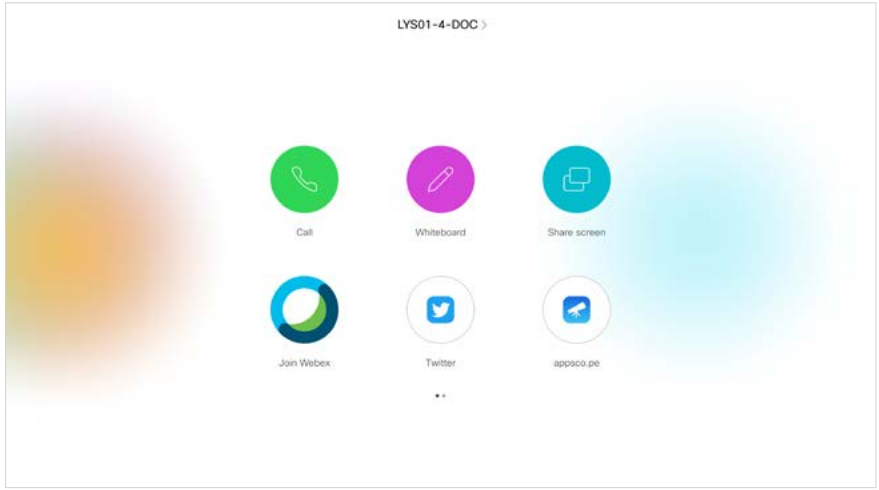


Note! On Webex Board, WebApps are only available outside of calls.

To create a WebApp:

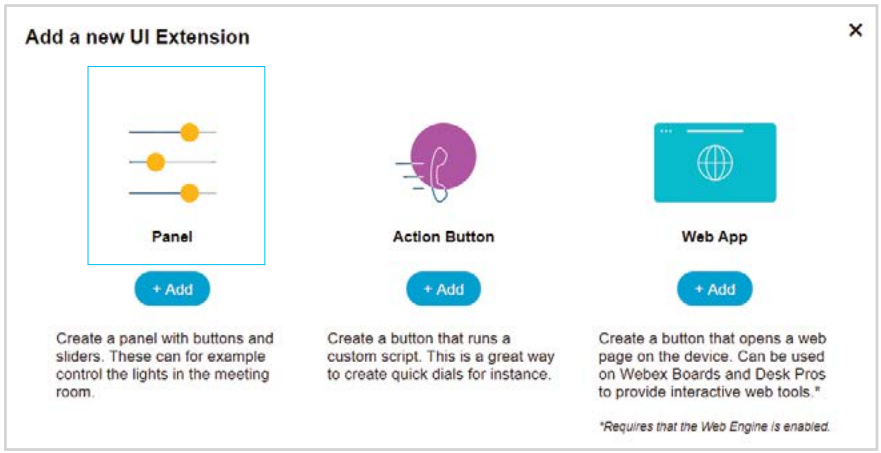
1. Provide the name you want to appear under the button (e.g., Twitter).
2. Provide the URL of the website (e.g., twitter.com).
3. Specify the color and image for the icon.
4. Alternatively, provide a URL for the web app icon to get the Favicon.

   Note that the Favicon will not be displayed in the editor, nor in the preview. It will be visible on the video device as the lower image at left indicates (using Twitter as example).
5. Click the button to export your WebApp and view it on the user interface.



Panels

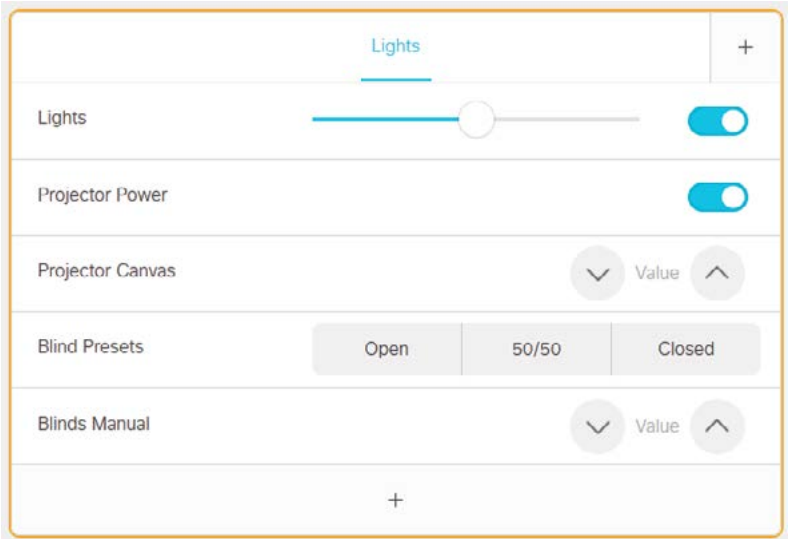
If you choose to create a panel, you will be adding a button to the user interface. When the user presses the button, the panel will open.



Panels typically have several widgets, including buttons, switches, sliders, and many more. These are described in detail in the Widgets section.

Panel widgets are arranged in a four-column grid according to the following guidelines:

- The rows are right-aligned.
- A widget fills one to four columns, depending on its size.
- If you add more widgets than fit in one line, the widgets will wrap to a new line within the same row.



It is a good idea to group related controls together on the same page. To add additional pages to a panel, click the + icon at the top-right of the panel.

A page consists of one or more rows that can be populated with widgets.

The pages you create will appear as separate tabs on the panel. There can be up to 50 pages in a panel.

The number of panels is unlimited, but only a few buttons will be displayed on each page. To access the overflow buttons, swipe from right to left in the screen area where the icons are displayed.

What happens when users interact with the panel is up to you. You must setup programs to monitor these widgets for interaction and then execute the desired commands. Macros are very convenient for this. For information about how to create a macro, see the Macros chapter.

Widgets (page 1 of 17)

About Widgets

The panel is composed of user interface elements called "widgets".

Types of widgets include:

- Switches
- Sliders
- Buttons
- Group Buttons
- Icon Buttons
- Spinners
- Text
- Directional Pad
- Spacer

Widget behavior is programmed through the API or through XML. Details for this are described on the following pages, with emphasis on:

- *Commands* that change the value of the widget
- *Events* that are sent (pressed, changed, released, clicked) and which actions trigger these events
- *Examples* of commands and events in macros, terminal output mode, or XML output mode.

Syntax and semantics for all events, commands, and statuses related to user interface extensions are included in the Application Programming Interface (API) chapter.

You can use the API through the terminal, XML, JSON, or the **Macro Editor**. For information about how to create a macro, see the Macros chapter.

The Widget Identifier

All widgets on a panel need a unique identifier, a Widget ID. The Widget ID may either be defined by you, or assigned automatically. The Widget ID can be any name or number. We recommend using a descriptive name without special characters. The maximum number of characters is 255.

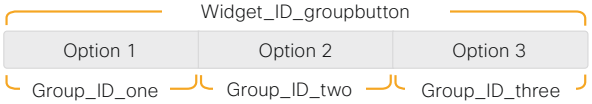
The Widget ID is the programming link between the user interface, the video device, and the control system. The Widget ID will be included in all events that are associated with a widget. You must use the same identifier when you send commands to that widget via the code.

Group Identifiers

One of the widgets, the *Group button*, has two types of identifiers:

- Widget ID – refers to the complete group of buttons
- Group IDs – unique identifiers for the individual buttons within the group.

  A Group ID is assigned automatically, but can be modified. A Group ID can be any name or number; we recommend using a descriptive name without special characters. The maximum number of characters is 255.



Events and Commands

Events are notifications about things that have happened (e.g., "the light switch has been turned off").

Commands are requests for things to happen (e.g., "turn off the lights").

If the status of any UI extension is changed (i.e., On/Off), an event will be sent. You can register to receive these events and then send commands based on your desires.

For example, if you create a switch to control the room lighting, then you need to detect if someone changes the switch from *On* to *Off*, or vice versa.

When you get the event that the light switch is changed, you then send a command to turn the physical lights on or off.

Similarly, if a light is turned on/off through some other control mechanism, you should detect this status change and then change the way the switch looks on the panel. This is done through the `SetValue` command, which sets the value of a widget.

Widgets (page 2 of 17)

Switches (page 1 of 2)

A *switch* is a two-state control which indicates either on or off.

Example of use: Anything that can be turned on or off (e.g., lights, fan, projector).

You can also use it as a toggle button together with a slider for lights to be dimmed.

Commands

Send commands to the switch to change the visual position on the panel using `SetValue`.

Example: Turn on a switch with WidgetId of "myswitch".

Terminal mode

To turn on a switch via in the terminal, for example:

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "myswitch" Value:
"on"
```

Macros

To turn on a switch with WidgetId of "myswitch" via a macro, for example:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'myswitch', value: 'on'});
```

Widgets (page 3 of 17)

Switches (page 2 of 2)

Events

Monitor the UI for changes. The following events will be issued anytime the switch is touched and the status is changed:

- *On* - Triggered when the switch is toggled from "off" to "on". Value: "on".
- *Off* - Triggered when the switch is toggled from "on" to "off". Value: "off".

Example: Press and release the button with WidgetId of "myswitch". Detect this action through the terminal, XML, or macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

When you change the switch, an event will be broadcast. For example:

```
*e UserInterface Extensions Event Changed Signal: "myswitch:on"
** end
```

XML mode

The XML will be, for example:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">myswitch</WidgetId>
          <Value item="1">on</Value>
          <Type item="1">changed</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To monitor the events through a macro, for example:

```
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'myswitch') &&
        (event.Type === 'changed')) {
      console.log(event.WidgetId, 'changed to',
                  event.Value);
    }
  });
```

Output:

```
'myswitch' 'changed to' 'on'
```

Widgets (page 4 of 17)

Sliders (page 1 of 2)

A *slider* selects values within a set range.

The minimum value is represented by 0, and the maximum value is represented by 255. When the slider is being pressed and moved, events are sent maximum 5 times a second.

When you tap the bar, the slider is immediately moved to that new position.

Example of use: Dimmable lights, volume control.

Commands

Send commands to update the slider position on the panel.

Example: Set a slider with WidgetId of "myslider" to position "98".

Terminal mode

To send a command in the terminal, for example:

```
xCommand UserInterface Extensions Widget SetValue WidgetId "myslider" Value:
"98"
```

Macros

To send a command in a macro, for example:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'myslider', value: '98'});
```

Widgets (page 5 of 17)

Sliders (page 2 of 2)

Events

Monitor the UI for changes. The following events are issued when the slider is moved or clicked:

- *Pressed* - Triggered when the slider is pressed. Value: N/A
- *Changed* - Triggered when the slider is moved while holding down, and when the slider is released.
  Value: 0-255
- *Released* - Triggered when the slider is released
  Value: 0-255

Example: Press the slider with WidgetId of "myslider", and move it into a new position ("194"), and release. Detect this action through the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

As you move the slider, several events will be broadcast. When the slider is released, the value can be set. For example:

```
*e UserInterface Extensions Widget Action WidgetId: "myslider"
*e UserInterface Extensions Widget Action Value: "194"
*e UserInterface Extensions Widget Action Type: "pressed"
** end
*e UserInterface Extensions Widget Action WidgetId: "myslider"
*e UserInterface Extensions Widget Action Value: "194"
*e UserInterface Extensions Widget Action Type: "changed"
** end
*e UserInterface Extensions Widget Action WidgetId: "myslider"
*e UserInterface Extensions Widget Action Value: "194"
*e UserInterface Extensions Widget Action Type: "released"
```

XML mode

```
The XML will be, for example:
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">myslider</WidgetId>
          <Value item="1">194</Value>
          <Type item="1">released</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To monitor the events through a macro, for example:

```
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'myslider') && (event.Type === 'released')) {
         console.log(event.WidgetId, 'changed to', event.Value);
    }
  });
```

Output:

```
'myslider' 'changed to' '194'
```

Widgets (page 6 of 17)

Buttons (page 1 of 2)

A *button*, can be used to perform an action, such as call someone, or to indicate that something is on or off.

For the second use, you will need to send a command to change the button value to "active" or "inactive" and this will change its color as well. Then, you must remember to turn it back to "inactive" if the user presses it again.

Inside the editor, double-click the button to change the text. Buttons with custom text come in different sizes. The size determines the maximum number of characters you can add. Text does not wrap to a new line.

You cannot use the `SetValue` command to change the text dynamically.

If you want to have the buttons linked so that only one can be selected at a time (e.g., radio buttons), consider to use group buttons.

Note: These are different from *action buttons*, described in the Action Buttons section. Different commands are used to monitor and change these.

Example of use: Switching things on and off.

Commands

Use the `SetValue` command to highlight the button in the user interface. A value of "active" will highlight the button, and a value of "inactive" will release it.

Example: Set the button with WidgetId of "mybutton" to active state.

Terminal mode

To activate a button with WidgetId of "mybutton" via the terminal:

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "mybutton" Value:
"active"
```

Then, to set it back to inactive:

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "mybutton" Value:
"inactive"
```

Macros

To activate a button with WidgetId of "mybutton" via macros:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'mybutton', value: 'active'});
```

Then, to set it back to inactive mode:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'mybutton', value: 'inactive'});
```

Widgets (page 7 of 17)

Buttons (page 2 of 2)

Events

The following events are issued when the button is pressed:

- *Pressed* – Triggered when the button is pressed. Value: N/A
- *Released* – Triggered when the button is released. Value: N/A
- *Clicked* – Triggered when the button is released. Value: N/A

Example: Press and release the button with WidgetId of "mybutton" and detect this action through the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

As the button is pressed and released, several events will be broadcast. For example:

```
*e UserInterface Extensions Widget Action Type: "pressed"
*e UserInterface Extensions Widget Action Type: "released"
*e UserInterface Extensions Widget Action Type: "clicked"
```

XML mode

The XML will be, for example:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">mybutton</WidgetId>
          <Value item="1"></Value>
          <Type item="1">clicked</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To detect a button press and do some action, for example:

```
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'mybutton') &&
        (event.Type === 'clicked')) {
          xapi.command('UserInterface Message Prompt Display',
                       { Title: 'Hello!', text: 'Have a great day!'});
    }
  });
```

A message will appear on the screen for you.

Widgets (page 8 of 17)

Group Buttons (page 1 of 2)

Group buttons are linked so that only one can be selected at a time.

Group buttons are ideal when you want buttons to be linked, so that only one can be selected at a time (e.g., room presets). When individual buttons in a group are too small to contain the descriptive text, you can use a *text* widget for the description.

Group buttons may be made as a matrix or as a line. You are not confined to a maximum of 4 buttons. A matrix consists of up to 4 columns and as many rows as you need.

You start by defining how many columns your matrix should contain (e.g., 1, 2, 3, or 4). This is a global setting applying to the entire matrix (i.e., all the rows) and it defines the maximum number of buttons per row.

However, a row may contain fewer buttons than this maximum number. Button auto-sizing will then take place– the buttons will always fill the space available.

Example: Assume that you have defined the matrix to consist of 3 columns and you need 7 buttons (i.e., 3 rows). The system will then put 3 buttons in the first row and 3 buttons in the second row, and the last button in the third row. The single button in the third row will be auto-sized to fill the space (spanning all 3 columns).

The size of a button determines the maximum number of characters you can add. Text does not wrap to a new line, but will be truncated, whenever needed.

You cannot use the `SetValue` command to change the text dynamically.

Example of use: Room presets that are mutually excluding, such as room presets where you can choose between Dark, Cool, and Bright. Remember to deselect (release) the preset if it is no longer valid (e.g., when changing the lights with a wall control).

Another example of use: Changing to a different UI language.

Commands

The visual appearance of the button changes immediately when you tap it on the UI. However, for changes done elsewhere, the control system always send a `SetValue` command to the video device when one of the buttons are tapped. This ensures that the status is updated accordingly.

The value to be set corresponds to the ID of the button that you wish to activate.

Use the `UnSetValue` command to release all buttons in the group so that no button is highlighted.

Example: Assume you have group buttons with WidgetId of "mygroup". In the group, there are three buttons with IDs as follows: {"high", "medium", "low"}. Select individual buttons via terminal and macro.

Terminal mode

To select the button with id "high":

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "mygroup" Value:
"high"
```

Then, to release all the buttons:

```
xCommand UserInterface Extensions Widget UnSetValue WidgetId: "mygroup"
```

Macros

To select the button with id="medium":

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'mygroup',
value: 'medium'});
```

Then, to release all the buttons:

```
xapi.command('UserInterface Extensions Widget UnSetValue', {WidgetId:
'groupbutton'});
```

Group Buttons (page 2 of 2)

Events

Monitor the UI for changes. The following types of events are issued for group buttons:

- *Pressed* – Triggered when one of the buttons is pressed. Value: The Group ID of the button (within the group) that was pressed.
- *Released* – Triggered when one of the buttons is released. Value: The Group ID of the button (within the group) that was released.

Example: Assume you have group buttons with WidgetId of "mygroup". In the group, are three buttons with IDs as follows: {"high", "medium", "low"}. Select individual buttons via the user interface and detect these in the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

As the button is pressed and released, several events will be broadcast. For example:

```
*e UserInterface Extensions Widget Action WidgetId: "mygroup"
*e UserInterface Extensions Widget Action Value: "high"
*e UserInterface Extensions Widget Action Type: "pressed"
** end
*e UserInterface Extensions Widget Action WidgetId: "mygroup"
*e UserInterface Extensions Widget Action Value: "high"
*e UserInterface Extensions Widget Action Type: "released"
** end
```

XML mode

The XML will be, for example:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">mygroup</WidgetId>
          <Value item="1">two</Value>
          <Type item="1">released</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To detect a group button press through macros, for example:

```
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'mygroup') &&
        (event.Type === 'released')) {
          console.log(event.WidgetId, 'changed to',
                  event.Value);
    }
  });
```

Output example:

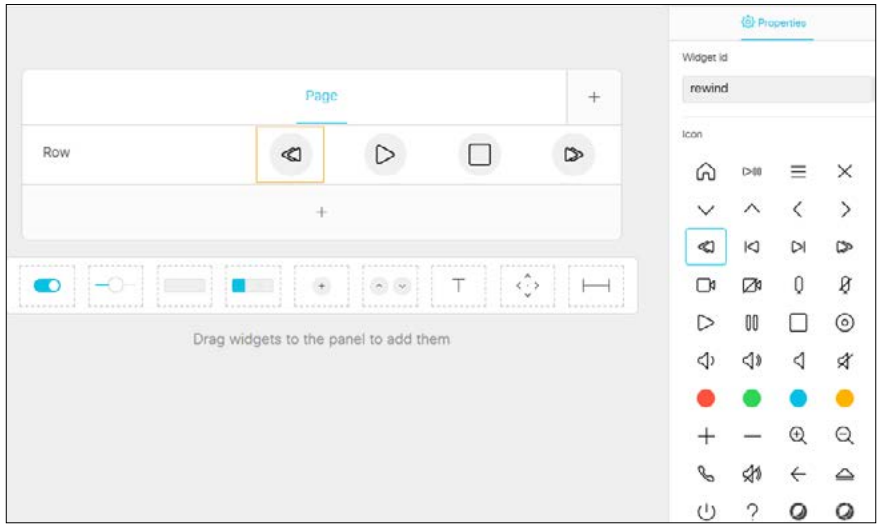
```
'mygroup' 'changed to' 'low'
```

Icon Buttons (page 1 of 2)

Icon buttons share behavior with other buttons, but they have an icon with a user-selected color or image.

An icon button has two states: active and inactive. You do not have to set the button in active state when someone taps it; the button can be used to just send a signal without changing its visual state.

Example of use: Controls for a media player, or other devices that can start, stop, pause.



Commands

Use the `SetValue` command to highlight the button in the user interface. A value of "active" will highlight the button and a value of "inactive" will release it.

Example: Assume your panel has four icon buttons with these ids: {"rewind", "play", "stop", and "fast_forward"}. Select the button with WidgetId of "play" through macros and the terminal.

Terminal mode

To select the button with WidgetId of "high":

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "play" Value: "active"
```

Then, to release the button:

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "play" Value: "inactive"
```

Macros

To select the button with WidgetId of "high":

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'play', value: 'active'});
```

Then, to release the button:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'play', value: 'inactive'});
```

Widgets (page 11 of 17)

Icon Buttons (page 2 of 2)

Events

Monitor the UI for changes. The following types of events are issued for icon buttons:

- *Pressed* - Triggered when the button is pressed. Value: N/A
- *Released* - Triggered when the button is released. Value: N/A
- *Clicked* - Triggered when the button is released. Value: N/A

Example: Assume your panel has four icon buttons with these ids: {"rewind", "play", "stop", and "fast_forward"}. Press the button with WidgetId of "play" via the user interface and detect these in the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

As the button is pressed and released, several events will be broadcast. For example:

```
*e UserInterface Extensions Widget Action WidgetId: "play"
*e UserInterface Extensions Widget Action Value: ""
*e UserInterface Extensions Widget Action Type: "pressed"
** end
*e UserInterface Extensions Widget Action WidgetId: "play"
*e UserInterface Extensions Widget Action Value: ""
*e UserInterface Extensions Widget Action Type: "released"
** end
*e UserInterface Extensions Widget Action WidgetId: "play"
*e UserInterface Extensions Widget Action Value: ""
*e UserInterface Extensions Widget Action Type: "clicked"
** end
```

XML mode

The XML will be, for example:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">play</WidgetId>
          <Value item="1"></Value>
          <Type item="1">clicked</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To detect a group button press through macros, for example:

```
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'play') &&
        (event.Type === 'clicked')) {
          console.log(event.WidgetId, 'was clicked.');
    }
  });
```

Output example:

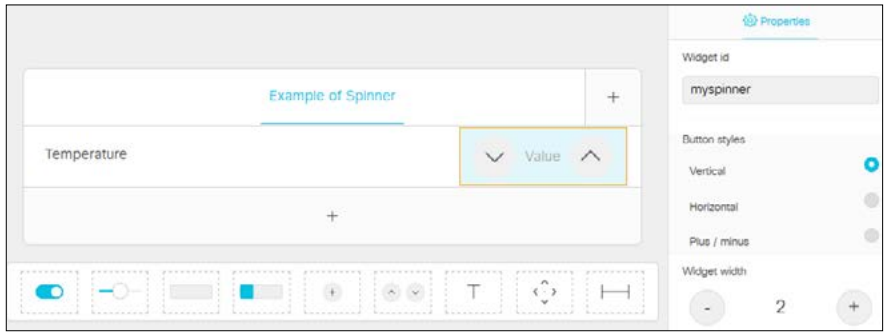
```
'play' was clicked
```

Widgets (page 12 of 17)

Spinners (page 1 of 2)

A *spinner* is used to step through a list of values. You may use the two buttons to increment or decrement a number, or to step through a list of options.

The text that is displayed on the panel initially is "Value", so you might send a command to initialize the text when you open the panel.



Example of use: Set the desired temperature in the room.

Commands

Use the `SetValue` command to add or update the text that is displayed between the buttons.

Example: For the spinner with WidgetId of "myspinner", add the text "22" between the spinner's increment and decrement buttons.

Terminal mode

To set the text between the spinner buttons to "22":

```
xCommand UserInterface Extensions Widget SetValue WidgetId: "myspinner"
Value: "22"
```

Macros

To set the text between the spinner buttons to "22":

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'myspinner', value: '22'});
```

Widgets (page 13 of 17)

Spinners (page 2 of 2)

Events

Monitor the UI for changes. The following types of events are issued for spinner buttons:

- *Pressed* – Triggered when one of the spinner buttons is pressed.
  Value: <increment/decrement>
- *Released* – Triggered when one of the spinner buttons is released.
  Value: <increment/decrement>
- *Clicked* – Triggered when one of the spinner buttons is released.
  Value: <increment/decrement>

Example: Press and release the decrement button on the spinner with WidgetId of "myspinner". Detect these in the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

As the button is pressed and released, several events will be broadcast. For example:

```
*e UserInterface Extensions Widget Action WidgetId: "myspinner"
*e UserInterface Extensions Widget Action Value: "decrement"
*e UserInterface Extensions Widget Action Type: "pressed"
** end
*e UserInterface Extensions Widget Action WidgetId: "myspinner"
*e UserInterface Extensions Widget Action Value: "decrement"
*e UserInterface Extensions Widget Action Type: "released"
** end
*e UserInterface Extensions Widget Action WidgetId: "myspinner"
*e UserInterface Extensions Widget Action Value: "decrement"
*e UserInterface Extensions Widget Action Type: "clicked"
** end
```

XML mode

The XML will be, for example:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">myspinner</WidgetId>
          <Value item="1">decrement</Value>
          <Type item="1">clicked</Type>
        </Action>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Macros

To change the displayed value when the user presses the up and down arrows:

```
let spinner_value = 22;
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'myspinner',
value: spinner_value});
xapi.event.on('UserInterface Extensions Widget Action', (event) => {
   console.log(event);
    if ((event.WidgetId === 'myspinner') && (event.Type === 'clicked')) {
        if (event.Value === 'increment') spinner_value++
        else spinner_value--
        xapi.command('UserInterface Extensions Widget SetValue', {WidgetId:
'myspinner', value: spinner_value});
    }
  });
```

Text

A *text* widget is used to place text on the display. The user does not interact with it.

Text widgets come in different sizes. They have up to two lines of text and the text automatically wraps to the new line.

A small text widget with larger font size and no line wrap is also available.

You can define the initial text for the text widget in the editor, and later on use the `SetValue` command to enter text dynamically.

Example of use: Help text, instructions, explanation of what different presets mean, or informative text from the control system, such as "The projector is warming up".

The text box with larger font size is primarily meant for status values, such as the current temperature in the room.

When individual buttons in a group are too small to contain the descriptive text, you can usea text widget for the description.

Commands

Use the `SetValue` command to modify the text in the text widget.

Example: Set the following text in the text widget with WidgetId of "mytext": "The projector is warming up.".

Terminal mode

To set the text using the terminal:

```
xCommand UserInterface  Extensions Widget SetValue WidgetId: "mytext" Value:
"The projector is warming up."
```

Then to remove it later, use a space:

```
xCommand UserInterface  Extensions Widget SetValue WidgetId: "mytext" Value: " "
```

Macros

To set the text using a macro:

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'mytext',
value: 'The projector is warming up.'});
```

Then to remove it later, set it to space, otherwise it will show "Text":

```
xapi.command('UserInterface Extensions Widget SetValue', {WidgetId: 'mytext',
value: ' '});
```

Events

There are no events associated with the text widget and it is not interactive.

Widgets (page 15 of 17)

Directional Pad (page 1 of 2)

The *directional pad* can be regarded as a set of five buttons: the four *directional* buttons and the *center* button.

The directional pad is used for controlling the movement of a device in two directions.

Example of use: Controlling a camera or an AppleTV

Commands

There are no commands available for the directional pad.

Widgets (page 16 of 17)

Directional Pad (page 2 of 2)

Events

Monitor the UI for changes. The following types of events are issued for the directional pad. One of the following values will be provided to indicate which button was pressed:
Value can be one of the following: "up", "down", "right", "left", "center".

- *Pressed* - Triggered when one of the buttons is pressed.
  Value: <"up"/"down"/"right"/"left"/"center">
- *Released* - Triggered when one of the buttons is released.
  Value: <"up"/"down"/"right"/"left"/"center">
- *Clicked* - Triggered when one of the buttons is released.
  Value: <"up"/"down"/"right"/"left"/"center">

Example: Press and release a button on the directional pad that has WidgetId of "mydirectionalpad". Detect this action in the terminal and through macros.

Terminal mode

Subscribe to widget events through the terminal. For example:

```
xFeedback Register Event/UserInterface/Extensions/Widget/Action
```

When one of the buttons is pressed and released, several events will be broadcast. For example, when pressing the "up" button:

```
*e UserInterface Extensions Widget Action WidgetId: "mydirectionalpad"
*e UserInterface Extensions Widget Action Value: "up"
*e UserInterface Extensions Widget Action Type: "pressed"
** end
*e UserInterface Extensions Widget Action WidgetId: "mydirectionalpad"
*e UserInterface Extensions Widget Action Value: "up"
*e UserInterface Extensions Widget Action Type: "released"
** end
*e UserInterface Extensions Widget Action WidgetId: "mydirectionalpad"
*e UserInterface Extensions Widget Action Value: "up"
*e UserInterface Extensions Widget Action Type: "clicked"
** end
```

XML mode

The XML will be, for example:

```
<Event><UserInterface item="1"><Extensions item="1">
<Widget item="1">
        <Action item="1">
          <WidgetId item="1">mydirectionalpad</WidgetId>
          <Value item="up"></Value>
          <Type item="1">clicked</Type>
        </Action>
      </Widget></Extensions></UserInterface></Event>
```

Macros

To change the displayed value when the user presses the arrows on the directional pad:

```
const xapi = require('xapi');
var x, x0, y, y0;
x = x0 = y = y0 = 50;

xapi.event.on('UserInterface Extensions Widget Action', (event) => {
    if ((event.WidgetId === 'mydirectionalpad') && (event.Type === 'clicked')) {
        switch(event.Value) {
          case 'right': x++; break;
          case 'left': x--; break;
          case 'up': y++; break;
          case 'down': y--; break;
          case 'center': // reset to original values
            x = x0;
            y = y0;
            break;
        }
        console.log('x:', x, 'y:', y);
    }
});
```

Widgets (page 17 of 17)

Spacer

The *spacer* lets you add space between or after widgets.

The width of the spacer is adjustable (1–4). If you set it to maximum, it will occupy its own line, making it usable as a vertical spacer, as well.

The *spacer* is no more than a layout tool. Consequently, there are no events or commands associated with it.

Commands

There are no commands available for the spacer.

Events

There are no events for the spacer.

Tips for UI Extensions (Page 1 of 2)

Register after Restart

When either the video device or the control system restarts, the control system must re-register to the events that the video device sends when someone uses the custom controls or exports a new control panel to the video device.

For terminal output mode:

```
xfeedback register event/UserInterface/Extensions/Widget
```

For XML output mode:

```
xfeedback register event/UserInterface/Extensions/Event
xfeedback register event/UserInterface/Extensions/Widget/LayoutUpdated
```

Consult the Application Programming Interface (API) chapter for more details.

Initialize All Widgets

Make sure the control system initializes all the widgets on the panel in the following situations:

- When the control system connects to the video device for the first time
- When the video device restarts
- When the control system restarts
- When a new control panel is exported to the video device (as response to a `LayoutUpdated` event).

If this is not done, then the system screen may show incorrect values and not truly reflect the status of the room.

Use the `SetValue` command to set the initial values.

Always send values back to the video device when something changes.

To avoid unexpected behavior and ambiguities, the control system must always send `SetValue` commands to the video device when something changes. This applies also when the change is triggered by someone using the controls on the system.

For example, it makes no difference if you use a slider on the panel to dim the light, or a physical dimmer in the room, or another touch controller. The control system must always send the dimmer value back to the video device using the `SetValue` command.

Restore Previous Values

Remember previous values when turning lights off (e.g., a light with a slider for dimming and toggle for on/off), remember the current light state when turning off, and use that value when turning back on.

Example: If the light is at 40%, and the user presses off, they will expect it to go back to 40% (not 100%) when pressing the *on* again. Remember also to set the power switch to off if sliding to 0%.

Using Widget IDs

When you drag a widget (e.g., a text field) onto a page, give it a customized id. Widget ids do not have to be unique. Widgets can share ids, but they must be of the same type. This means that you can have two sliders in different panels called "main-light", but you cannot have one slider and one toggle button both called "main-light".

To create a duplicate of an existing widget on another page or panel, just use copy-and-paste.

Tips for UI Extensions (page 2 of 2)

Updating a Panel

When you export a new panel to the video device, the old panel is overwritten and replaced by the new one.

To update:

1. Launch the **UI Extensions Editor** from the video device's web interface.
2. Create the panel you want, or import a previously saved panel from file (*Import > From file*).
3. Click *Export > To codec*.

Removing a Panel

If there is a custom panel on the video device, then there is also a corresponding button in the user interface. Even if a panel is empty and contains no widgets, both the icon and the panel will be visible.

To remove a Control panel and icon:

1. Launch the **UI Extensions Editor** from the video device's web interface.
2. Select the panel to be removed.
3. Click *Delete panel*.

Transitioning from Third-party Control Systems to CE

If you already have been using a third-party control system and want to start using CE as described in this document, we recommend the following:

1. Let any programming made to control third party stuff remain untouched.
2. Remove all code that controls the Cisco video device as that is now controlled via the UI Extension.
3. Reprogram the signaling from the button-presses coming from the third-party control system panel so that it listens to button presses from the Cisco video device instead.

This programming can be very simple to do, as the largest control system manufacturers provide modules/drivers for controls.

Troubleshooting UI Extensions

Sign In

Sign in to the video device's web interface with administrator credentials, navigate to *Integration > UI Extensions Editor*. Click the arrow to show the *Development Tools*.

Overview of all Widgets and Their Status

The **Widget State Overview** window lists all widgets and their status. The status is shown in the *Current Value* column.

If the *Current Value* column is empty, the widget has not been initialized and has no value. We recommend that the control system initializes all widgets when it initially connects to the video device.

Send Value Updates to the Video Device

A control system sends `SetValue` commands to the video device, telling it to update a widget. For test purposes, you can use the *Update Value* column in the **Widget State Overview** window to simulate a control system.

Enter a value in one of the input fields to immediately send the corresponding `SetValue` command to the video device. The *CurrentValue* column (status) will be updated, and the touch controller panel changes accordingly.

Click *Unset* to clear the value of the widget (send an `UnsetValue` command).

If a control system is connected to the video device, the *Current Value* and *Update Value* columns may come out-of-sync. The *Current Value* column always shows the current status, regardless of whether the SetValue command is sent from a real control system, or from the *Update Value* column.

Check for Events and Status Updates

All events and status updates related to widgets appear immediately in the Log window. Events are prefixed with `*e`, and statuses are prefixed with `*s`.

Events appear when you use the controls on the user interface. The status is updated when a command that changes the video device's status is sent to the video device.

If a Panel Fails to Load

If an existing panel fails to load automatically on launching the **UI Extensions Editor**, you may need to manually import the panel from video device or load a backup XML file.

Be aware that these alternatives erase any unsaved data in the editor, but the existing panel on the video device is neither overwritten nor deleted until a new panel is exported to the video device.

Check the Macros

If you experience unintended behavioral changes and you run macros on your system, make sure you deactivate the macros before proceeding with your troubleshooting.

Use `xConfiguration Macros Mode: On/Off` to do this.

The macro framework has its own log file called *macros.log*.

The *macros.log* file contains much of what is printed in the **Console Log**. The macros can be configured to print output to the console and this will be stored in the log, so keep in mind that you can see custom log messages (which may have been created by the developer) in this file.

Chapter 3

Macros

The Macro Framework

Macros are scripts written in the **Macro Editor** using JavaScript and xAPI commands.

The Macro Framework has many benefits, as it allows integrators to:

- Tailor deployments
- Create custom "features" or "workarounds"
- Automate scenarios/re-configurations
- Create custom tests or monitoring

It can be useful to combine macros and external systems (e.g., adjusting the light depending on presentation status or call status).

Your macros may contain customized text to be displayed on the user interface. This text may alert users to observe the activation or deactivation of certain features; as well as, alert them to act in accordance with the message given, etc.

Such text can be purely informative, but it may also prompt the user to respond to it by keying in information. This information may, in turn, cause the video device to directly act upon it.

With the macros, UI extensions no longer need an external control system to activate local functionality.

However, performing local actions via the xAPI, such as to control other things like lights and blinds will still require a suitable third-party control system.

Examples of local functionality can be a panel for speed dialing or to trigger a "Room Reset" that puts all the configurations back to default.

Note that extensive use of macros may slow down video device performance due to heavy load.

Macros can alter the expected behavior of the system. If your macro is likely to surprise or confuse normal users, let them know, for example, by providing a notification on the video device screen.

We recommend that you never let macros depend on other macros. You can, of course, create several macros that monitor the same xapi values (e.g., the call state) as long as their actions are independent of each other.

Disclaimer: Cisco will support the Macro Framework itself only. Cisco will not support code that fails to compile or fails to work as the developer "intended". It is entirely up to the person writing the code to ensure that the syntax is correct and that possessed coding skills are sufficient to write macros in JavaScript. Public developer forums can provide assistance with creating macros.

A Tour of the Macro Editor (page 1 of 3)

Sign-in to the video device's web interface with administrator credentials and navigate to *Customization* > *Macro Editor*.

The first time this is done, you will be asked whether to enable the use of macros on this video device.

To start a new macro programming session, you can either import existing code from a file (*.js), by clicking *Import from file...* or create a blank macro by clicking *Create new macro*.

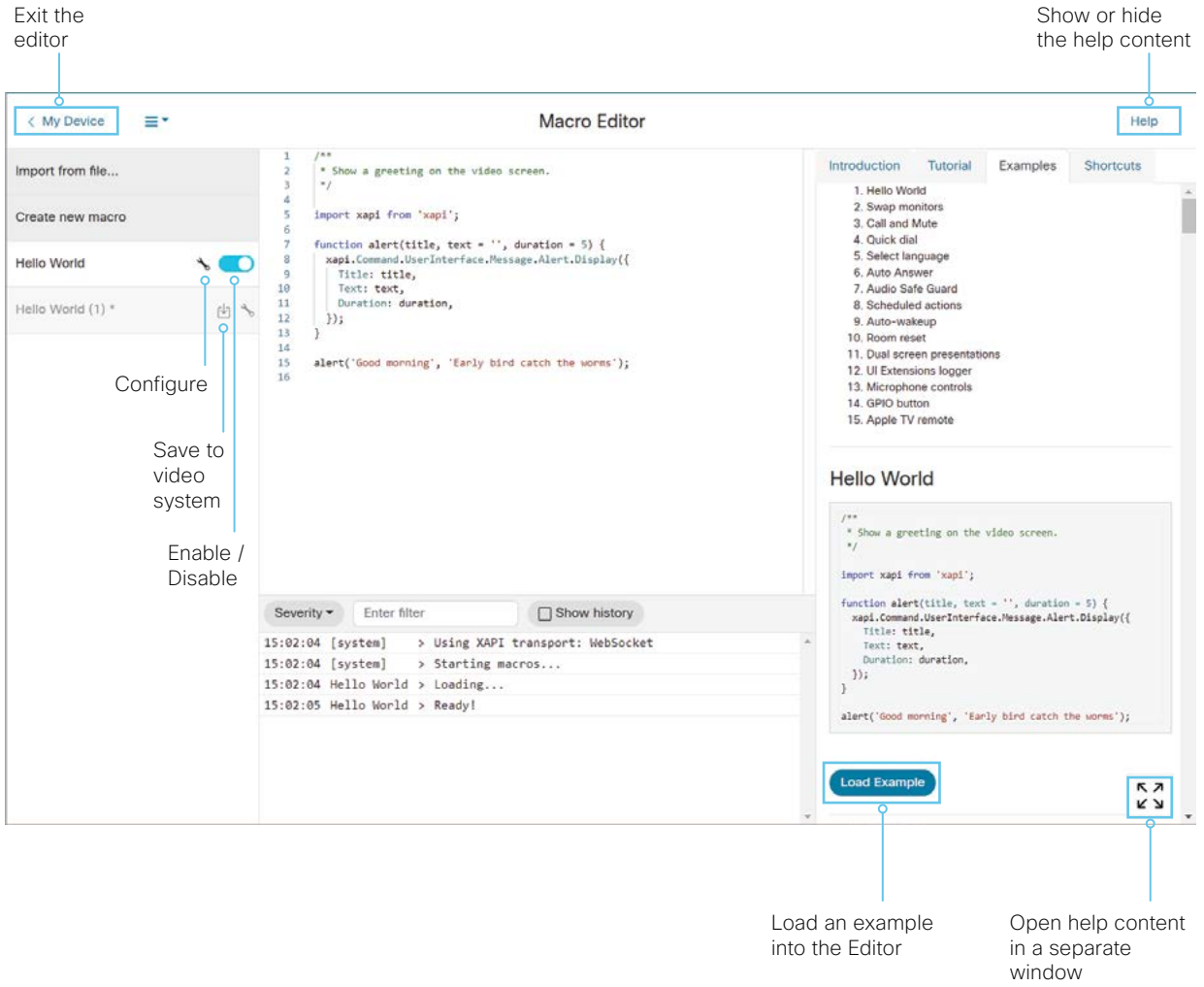
For an example, locate the "Hello World" example in the *Help* section > *Examples* tab. Click *Load Example*.

You will see a new macro example appear in the editor. Click the *Save* icon on the new macro and then the *Disable/Enable* toggle will appear. Toggle to enable the macro.

When this happens, the macro will automatically run and you should see a greeting on your touch controller or screen.

The **Macro Editor** contains many examples and tutorials. Examples for monitoring and commanding specific widgets are found under Widgets.

See Online Resources for links to tutorials and examples available on the web.



A Tour of the Macro Editor (page 2 of 3)

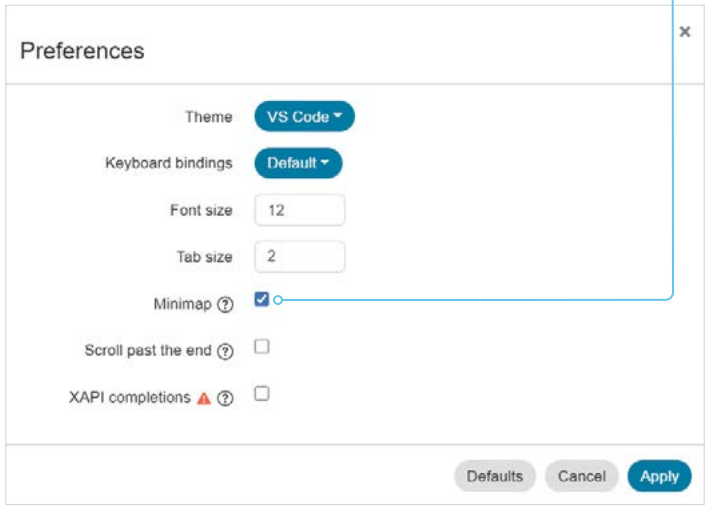
Anytime a macro is saved, deleted, activated, or deactivated, the whole *Runtime* is restarted and all the macros are re-run. Disable all the unused macros while you are troubleshooting.

The **Macro Editor** automatically detects syntax errors and will prevent you from saving the code if there are errors detected in the script. Hover over the error to see details.

Use the options in the dropdown to start, stop, or restart the runtime.

- *Stop* - Stop the *Runtime*. This will stop all macros.
- *Start* - Start the *Runtime*. This will automatically execute all enabled macros.
- *Restart* - Stop and restart the *Runtime*. This will automatically execute all enabled macros.

  Note! After a restart, macros will be re-enabled automatically.
- *Toggle log* - Hide or Show the **Log Console** pane.
- *Preferences* - Open the *Preferences* dialog box.
- *Delete all macros* - Delete all macros from the editor.



The *Preferences* box contains several options for the **Macro Editor**.

- *Theme* - Choose a color themes for the editor.
- *Keyboard bindings* - Define which set of keyboard shortcuts to use. Options are: *Default, Vim,* and *Emacs*.
- *Font size* - Define the character font size.
- *Tab size* - Define the number of spaces the editor will use for each tab indention.
- *Minimap* - Toggle the display of the minimap overview scroller. This shows a small map of your script to the right of the editor and lets you know your position in the full script. It is most useful for long scripts.
- *Scroll past the end* - Keep the bottom lines of your macro in the middle of your screen.
- *XAPI completions* - Enable automatic word completion editing to allow the editor to predict the rest of the word you are typing.

  You can also use *Ctrl-Space* for suggestions when auto-completion is disabled.

A Tour of the Macro Editor (page 3 of 3)

The purpose of the **Log Console** is to reveal what happens when you run the macro.

If you do not see the **Log Console** in the lower part of the window, you can display it with the *Runtime* > *Toggle Log* menu, or by pressing `Ctrl+Shift+G`.

The **Log Console** will report on the actions of the *Runtime* and your macros. You can choose to print out information to the console while your macro is running.

For example, create a new macro and add the following line:

```
console.log('Hello World!');
```

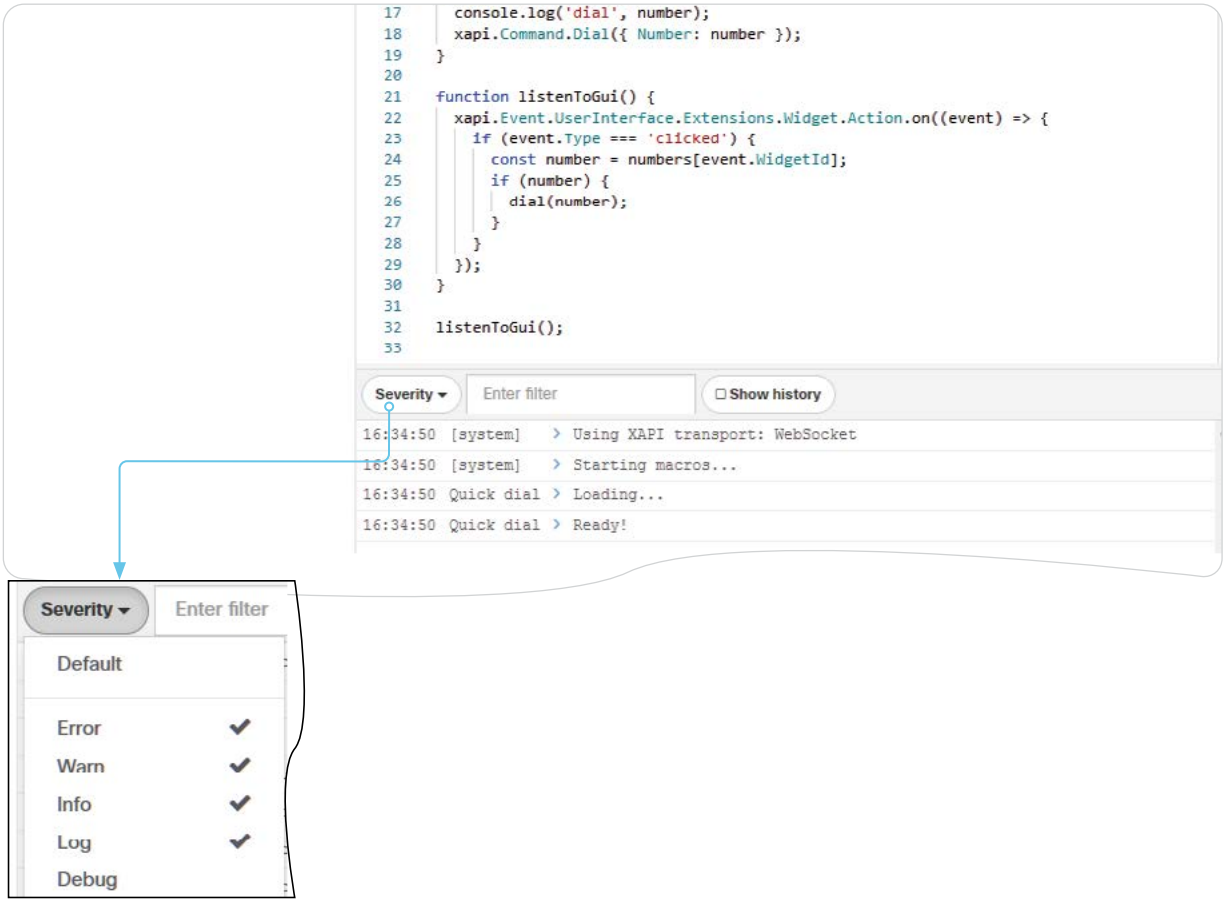
Now, press `Ctrl+S` to save and run the macro. Watch the **Log Console** and you should see your message appear in the log.

Console messages and other error messages will give you insight into problems with your macros.

There are several options available:

- *Severity* - Select what level of information you want to see in the log. Options include: *Default, Error, Warn, Info, Log, Debug.*
  Make sure the type of information you want to be logged is checked here; otherwise, it will not appear in the console log window.
- *Enter filter* - Enter text that should be used to filter the logs. Only messages with the text you entered will be displayed.
- *Show history* - If enabled, show the entire history of log messages. Otherwise, show only what has taken place since the last restart.

Most of what is displayed in the **Log Console** pane is written to the *macros.log* file on the video device.



The Macro Runtime

About Macro Runtime

All the activated macros run in a single process on the video device, called the *Runtime*. It should be running by default, but you can choose to stop and start it manually from the editor. If you restart the video device, the *Runtime* will automatically start again, if xconfiguration macros autostart is `On`.

If any macro becomes unresponsive or fails to respond within a few seconds (for example, due to an infinite loop), a safety mechanism will stop the *Runtime*; thereby, stopping all macros.

The *Runtime* will be automatically restarted after a few seconds. This will continue, but the time between restarts will increase every time the *Runtime* is shut down. If this happens more than a certain number of times, this will cause a system diagnostic to be displayed to notify that the macros are having problems.

Enabling/Disabling Macros

You may disable the use of macros from within the **Macro Editor**. This will not permanently disable macros from running. Every time the video device is restarted, the macros will be re-enabled automatically.

To disable automatic restart, you must use the `xConfiguration Macros Mode: Off`.

You may want to use this command in the event of unintended behavior by the system. In such cases you should always disable the macros before proceeding with your troubleshooting.

Troubleshooting

If you experience unintended behavioral changes and you run macros on your system, make sure you deactivate the macros before proceeding with your troubleshooting.

Use `xConfiguration Macros Mode: On/Off` to do this.

The macro framework has its own log file called *macros.log*.

The *macros.log* file contains much of what is printed in the *Log Console*. The macros can be configured to print output to the console and this will be stored in the log, so keep in mind that you can see custom log messages (which must have been created by the developer) in this file.

Resources

Learning material for the creation of macros can be found in the online help section of the **Macro Editor**.

See Online Resources for links to tutorials and examples available on the web.

Chapter 4

Application Programming Interface (API)

API for Programming UI Extensions (page 1 of 2)

Connect to the Video Device

The video device's API (also known as the xAPI) allows bidirectional communication with third-party control systems, such as those from AMX or Crestron. There are multiple ways to access the xAPI:

- Telnet
- SSH
- HTTP/HTTPS
- RS-232 / serial connection

Regardless of the method you choose, the structure of the xAPI is the same. Choose the access method that suits your application and video device the best.

Consult the API guide for your video device to get a full description of available access methods and how to use the xAPI.

Go to:

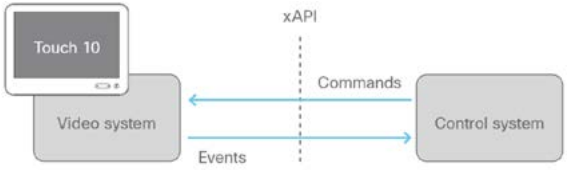
- ▶ Cisco Webex Board series
- ▶ Cisco Webex Desk series
- ▶ Cisco Webex Room series

Then, click *Reference Guides > Command References* to find the API guides.

Communicate via the API

The video device and the control system exchange messages through the xAPI to make sure that the user interface always reflects the actual status of the room.

The video device sends one or more events when someone uses one of the controls on the custom user interface panel. The control system should send a command to the video device when there is a change in the room settings.



The video system and the control system exchange messages through the xAPI.

Examples:

When someone taps a **Lights On** button on the user interface, the video device sends the associated events. The control system should respond to these events by switching on the lights in the room and send the corresponding command back to the video device.

When someone switches on the lights in the room, the control system should send a command to the video device, so that the video device can update the custom panel on the user interface to reflect that the light is on.

This can be done with the terminal or with macros. For information about how to create a macro, see the Macros chapter.

Pairing Video Device and Control System

You can register the control system as a peripheral connected to the video device:

```
xCommand Peripherals Connect ID: "ID" Type: ControlSystem
```

Here, `ID` is the unique ID for the control system. This is typically the MAC address.

Refer to the API guide for your video device more details about this command and its options.

Heartbeats. The control system must send heartbeats to inform the video device that it is still connected:

```
xCommand Peripherals HeartBeat ID: "ID" [Timeout: Timeout]
```

Here, `ID` is the unique ID for the control system, typically the MAC address. `Timeout` is the number of seconds between each heartbeat. If `Timeout` is unspecified, it is assumed to be 60 seconds.

The control system stays on the connected devices list as long as the video device receives its heartbeats. Refer to `xStatus Peripherals ConnectedDevice` in the API guide for your device.

If a connected unit ceases to send heartbeats, some time will elapse until the video device detects the absence of heartbeats. This could take as long as 2 minutes.

Similarly, a couple of minutes may elapse before the video device detects new heartbeats from a control system.

API for Programming UI Extensions (page 2 of 2)

Using SetValue

The `SetValue` command, which sets the value of a widget, is essential when working with UI Extensions:

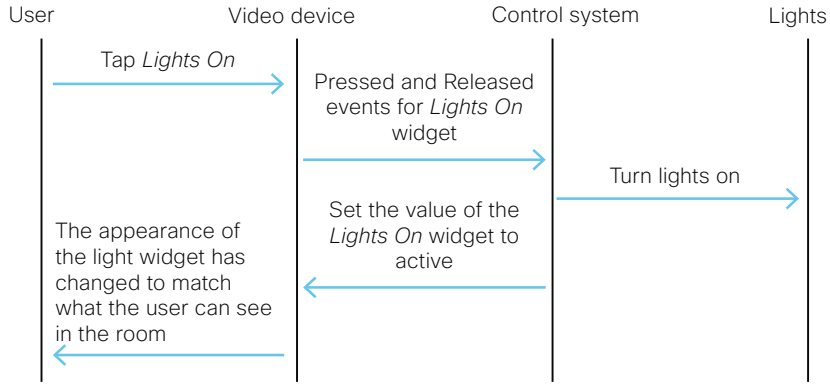
```
xCommand UserInterface Extensions Widget SetValue Value: Value WidgetId: WidgetId
```

When the video device receives a `SetValue` command, the video device's status and the custom panels on the user interface are updated accordingly.

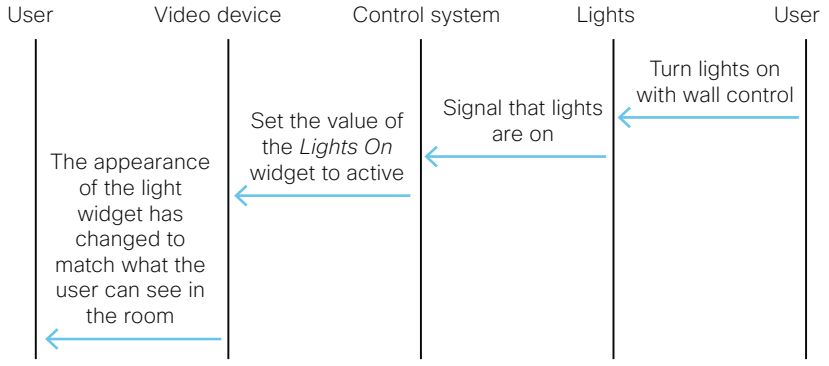
It is important that the control system sends `SetValue` commands in the following situations, so that the video device truly reflects the status of the room:

- When the control system initially connects to the video device.
- When the video device restarts.
- When the control system restarts.
- When a new custom panel is exported to the video device from the **User Interface Extensions Editor** (as response to the LayoutUpdated event). See User Interface Extensionsfor details.
- When someone physically changes something in the room (e.g., turns on the lights using a wall control).
- As a response to an event (e.g., when someone has tapped the *Lights On* button on the custom panel).
- The control system must also do all that is necessary in the room to reflect the action on the custom UI panel (e.g., physically switch on the light).

Examples



Message flow: Turn on the lights using the controls on custom panel



Message flow: Turn on the lights using the wall control

Command Reference

UserInterface Extensions Widget SetValue

This command sets the value of the given widget, and the UserInterface Extensions statuses are updated accordingly. If the value is out of range, the command returns an error.

USAGE:

```
xCommand UserInterface Extensions Widget SetValue Value: Value WidgetId:
WidgetId
```

Arguments:

Value: String (0, 255) - The value of the widget. The range of values depends on the widget type.

WidgetId: String (0, 40) - The unique identifier for the widget.

UserInterface Extensions Widget UnsetValue

This command empties the value of the given widget, and the UserInterface Extensions statuses are updated accordingly. The user interface is notified that the widget is no longer selected.

USAGE:

```
xCommand UserInterface Extensions Widget UnsetValue WidgetId: WidgetId
```

Arguments:

WidgetId: String (0, 40) - The unique identifier for the widget.

UserInterface Extensions Clear

This command deletes all user interface extensions (widgets) from the video device.

USAGE:

```
xCommand UserInterface Extensions Clear
```

UserInterface Extensions List

Use this command to list all user interface extensions (widgets) that exist on the video device.

USAGE:

```
xCommand UserInterface Extensions List
```

Status Reference

UserInterface Extensions Widget [n] WidgetId

UserInterface Extensions Widget [n] Value

This status returns the identifier (WidgetId) and the current value of the widgets.

The value is an empty string until a value is set by using the `UserInterface Extensions Widget SetValue` command.

USAGE:

```
xStatus UserInterface Extensions
```

Value space of the result returned:

Value: The value of the widget. Depends on widget type. String (0, 255).

WidgetId: The unique widget identifier. String (0, 40).

Example:

```
xstatus UserInterface Extensions
*s UserInterface Extensions Widget 1 Value: "on"
*s UserInterface Extensions Widget 1 WidgetId: "togglebutton"
*s UserInterface Extensions Widget 2 Value: "255"
*s UserInterface Extensions Widget 2 WidgetId: "slider"
*s UserInterface Extensions Widget 3 Value: "Blinds"
*s UserInterface Extensions Widget 3 WidgetId: "spinner"
*s UserInterface Extensions Widget 4 Value: "inactive"
*s UserInterface Extensions Widget 4 WidgetId: "button"
*s UserInterface Extensions Widget 5 Value: "2"
*s UserInterface Extensions Widget 5 WidgetId: "groupbutton"
*s UserInterface Extensions Widget 6 Value: "Projector is ready"
*s UserInterface Extensions Widget 6 WidgetId: "textfield"
** end
```

Events Reference (page 1 of 4)

The video device sends one or more of the following events when someone uses the controls on the custom panel of the user interface:

- **Pressed**—sent when a widget is first pressed
- **Changed**—sent when changing a widget's value (applies to toggle buttons and sliders only)
- **Released**—sent when a widget is released (also when moving away from the widget before releasing)
- **Clicked**—sent when a widget is clicked (pressed and released without moving away from the widget).

These events are sent in two versions:

- **UserInterface Extensions Event**—suited for terminal output mode
- **UserInterface Extensions Widget**—suited for XML output mode.

See the table at right to find out the version best suited for your control system to register to.

When, and by which widgets (user interface elements), these events are triggered, are described in the Widgets section.

UserInterface Extensions Event (terminal output mode)	**UserInterface Extensions Widget** (XML output mode)
A single string contains information about the type of action, which widget triggered the event (identified by the Widget ID), and the widget value.	The type of action, which widget triggered the event (identified by the Widget ID), and the widget value are included as separate elements in the XML tree.

How to register:
`xfeedback register event/UserInterface/Extensions/Event`
 	
Example:
`*e UserInterface Extensions Event Pressed Signal: "WidgetId:Value"`
`** end`
`*e UserInterface Extensions Event Changed Signal: "WidgetId:Value"`
`** end`
`*e UserInterface Extensions Event Released Signal: "WidgetId:Value"`
`** end`
`*e UserInterface Extensions Event Clicked Signal: "WidgetId:Value"`
`** end`
	How to register:
	`xfeedback register event/UserInterface/Extensions/Widget`
	
	Example:
	`<Event>`
	`  <UserInterface item="1">`
	`  <Extensions item="1">`
	`  <Widget item="1">`
	`  <Action item="1">`
	`  <WidgetId item="1">WidgetId</WidgetId>`
	`  <Value item="1">Value</Value>`
	`  <Type item="1">Type</Type>`
	`  </Action>`
	`  </Widget>`
	`  </Extensions>`
	`  </UserInterface>`
	`</Event>`

Two event versions that a control system can register to: one suited for terminal output mode, the other for XML output mode

Events Reference (page 2 of 4)

Example Events for Panel Update

The video device sends the following event when a new Control panel is applied:

LayoutUpdated—sent when a new panel is exported to the video device.

As a response to this event, the control system should send commands to initialize all widgets so that they reflect the true status of the room settings.

How to register:

```
xfeedback register event/UserInterface/Extensions/Widget/LayoutUpdated
```

Example:

Terminal output mode:

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

XML output mode:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <LayoutUpdated item="1"/>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

Example Events for Opening or Closing a Page

If you have given each of your pages a unique Page ID, the system can send events when a page is opened or closed.

EventPageOpened—sent when a page is opened

EventPageClosed—sent when a page is closed

The pages are like radio buttons, opening another page will close the current page. In that case both the EventPageClosed and the EventPageOpened will be issued.

How to register:

```
xfeedback register event/UserInterface/Extensions/PageOpened
xfeedback register event/UserInterface/Extensions/PageClosed
```

Example:

Terminal output mode:

```
*e UserInterface Extensions Event PageOpened PageId: "appletvpage"
*e UserInterface Extensions Event PageClosed PageId: "appletvpage"
```

XML output mode:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Page item="1">
        <Action item="1">
          <PageId item="1">appletvpage</PageId>
          <Type item="1">Opened</Type>
        </Action>
      </Page>
    </Extensions>
  </UserInterface>
</Event>
```

For an example of PageClosed, just substitute `Closed` for `Opened` in the example. This event will typically be used when you want the controller to take some action based on the event, in this case turning on (off) the AppleTV box.

Events Reference (page 3 of 4)

UserInterface Extensions Event Pressed

Sent by the video device when a widget is first pressed.

Equivalent to the UserInterface Extensions Widget Action event with Type "Pressed".

```
*e UserInterface Extensions Event Pressed Signal: Signal
```

Arguments:

Signal: String (0, 255) - The format of the string is "<WidgetId>:<Value>", where <WidgetId> is the unique identifier of the widget that triggers the event, and <Value> is the value of the widget. The range of allowed values depends on the widget type.

UserInterface Extensions Event Changed

Sent by the video device when changing a widget's value. This applies only to toggle buttons and sliders.

Equivalent to the UserInterface Extensions Widget Action event with Type "Changed".

```
*e UserInterface Extensions Event Changed Signal: Signal
```

Arguments:

Signal: String (0, 255) - The format of the string is "<WidgetId>:<Value>", where <WidgetId> is the unique identifier of the widget that triggers the event, and <Value> is the value of the widget. The range of allowed values depends on the widget type.

UserInterface Extensions Event Released

Sent by the video device when a widget is released (even if moving the finger out of the widget before releasing it).

Equivalent to the UserInterface Extensions Widget Action event with Type "Released".

```
*e UserInterface Extensions Event Released Signal: Signal
```

Arguments:

Signal: String (0, 255) - The format of the string is "<WidgetId>:<Value>", where <WidgetId> is the unique identifier of the widget that triggers the event, and <Value> is the value of the widget. The range of allowed values depends on the widget type.

UserInterface Extensions Event Clicked

Sent by the video device when a widget is clicked (pressed and released without moving the finger out of the widget).

Equivalent to the UserInterface Extensions Widget Action event with Type "Clicked".

```
*e UserInterface Extensions Event Clicked Signal: Signal
```

Arguments:

Signal: String (0, 255) - The format of the string is "<WidgetId>:<Value>", where <WidgetId> is the unique identifier of the widget that triggers the event, and <Value> is the value of the widget. The range of allowed values depends on the widget type.

Events Reference (page 4 of 4)

UserInterface Extensions Widget LayoutUpdated

Sent by the video device when the configuration file for the user interface extensions has been updated, such as when exporting a new configuration from the **UI Extensions Editor** to the video device.

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

In XML:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <LayoutUpdated item="1"/>
      </Widget>
    </Extensions>
  </UserInterface>
</Event>
```

UserInterface Extensions Widget Action

Sent by the video device when someone uses one of the controls on the user interface. Equivalent to the `UserInterface Extensions Event Type` event.

Depending on the action type, this event is equivalent to one of these events:

```
*e UserInterface Extensions Event Pressed
*e UserInterface Extensions Event Changed
*e UserInterface Extensions Event Released
*e UserInterface Extensions Event Clicked Events
```

In XML:

```
<Event>
  <UserInterface item="1">
    <Extensions item="1">
      <Widget item="1">
        <Action item="1">
          <WidgetId item="1">WidgetId</WidgetId>
          <Value item="1">Value</Value>
          <Type item="1">Type</Type>
        </Action>
      </Widget></Extensions></UserInterface></Event>
```

Arguments:

WidgetId: String (0, 40) - The unique identifier for the widget that triggered the event.

Value: String (0, 255) - The value of the widget. The range of allowed values depends on the widget type.

Type: <Pressed/Changed/Released/Clicked>

- **Pressed**: Sent when a widget is first pressed.
- **Changed**: Sent when changing a widget's value (only for toggle buttons and sliders).
- **Released**: Sent when a widget is released (even if moving the finger out of the widget before releasing it).
- **Clicked** : Sent when a widget is clicked (pressed and released without moving the finger out of the widget).

Chapter 5

Audio Console

Customizing the Audio Connections (page 1 of 2)

The **Audio Console**, available for systems using Codec Pro, can be found under *Customization* in the web interface of your video device.

This utility lets you define how audio inputs and outputs should be connected together using simple drag and drop.

Start by defining logical input and output groups. These can be given names as defined by you. Physical inputs and outputs are assigned to the logical inputs and outputs.

New logical groups may be added or removed at any time.

Changes applied to the settings are immediately saved and put into effect.

A physical output cannot be assigned to more than a single logical output group; however, a physical input, such as a microphone, can be assigned to more than one input. This comes in handy when working with local reinforcement, such as using a video device in lecture halls where the local audience also needs to hear what is being said through the microphone.

The **Audio Console** setup lets you utilize *Echo Control* on the part of the microphone signal that is sent to the far end. At the same time, it omits it for the part used locally. This is done by assigning that microphone to more than one logical group. Use *Direct On* for this, see the next page for more on this.

You may also apply noise reduction and equalizer settings to the microphone signal.

The Codec Pro has no dedicated line inputs. One or more of the line inputs may be used for line level input by deactivating phantom power and lowering the input gain (Level).

However, if these inputs are used for presentation audio delivery, a limitation with this approach is that the inputs are still muted when you press the microphone mute button.

For more information pertaining to seperate devices, see the *Using Extra Loudspeakers and Sound Reinforcement guide*. Go to:

► https://www.cisco.com/go/in-room-control-docs

Customizing the Audio Connections (page 2 of 2)

Using Audio Return Channels

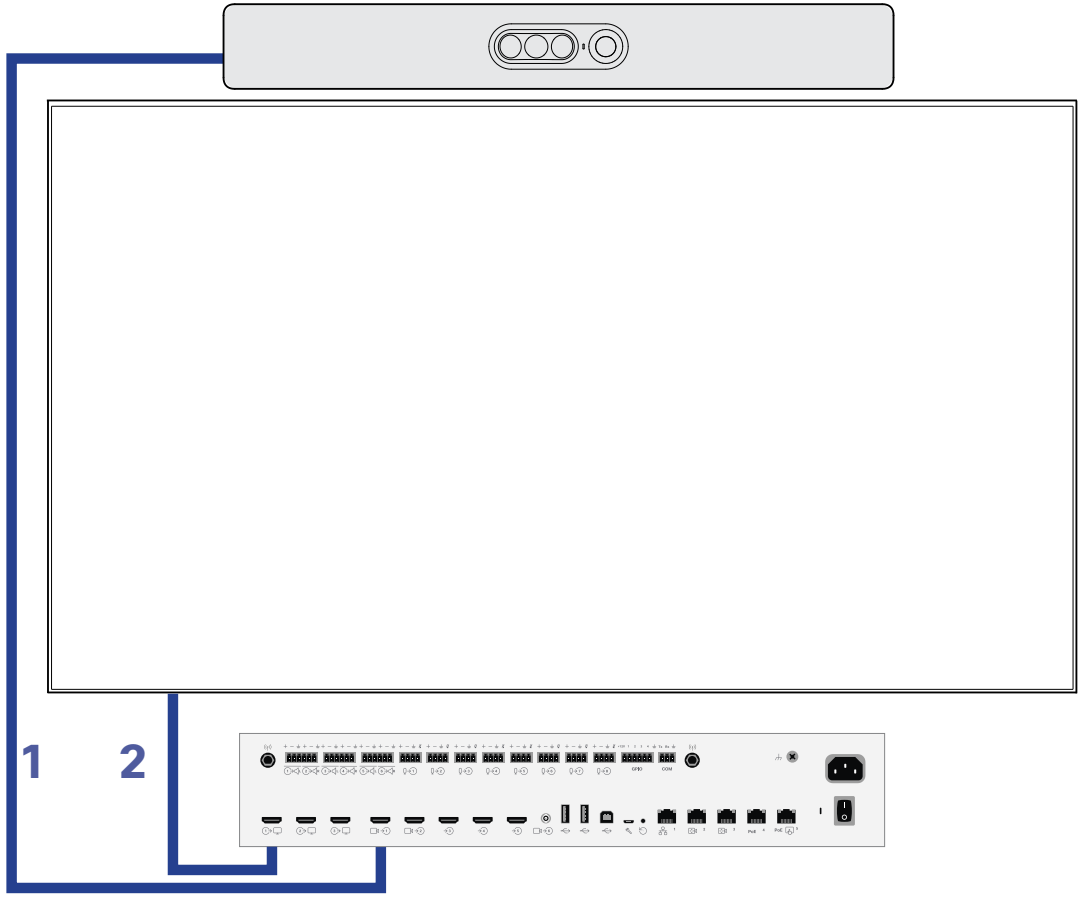
HDMI offers, under certain circumstances, the ability to transmit audio in either direction. When audio is sent in reverse, this is referred to as an Audio Return Channel (ARC). The Codec Pro supports using ARCs.

Consider the illustrated configuration, showing a Cisco QuadCam (top), a screen (middle), and a Codec Pro (bottom) connected via HDMI. The QuadCam will act as both camera and soundbar in this setting.

During normal use, the HDMI in 1 is used to provide the Codec Pro with video from the cameras of the QuadCam unit and the audio return channel of the same will be used to route the audio from the Codec Pro to the loudspeakers of the QuadCam.

In addition, if you want to use the setup as just a TV with a soundbar, the system will send audio from the monitor to the Codec Pro through HDMI out 1 (ARC) of Codec Pro and the Codec Pro will send that audio further to the QuadCam through HDMI1 in (ARC).

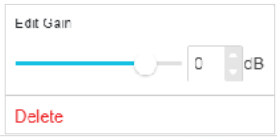
In order to be able to do this, the monitor must be CEC+ARC enabled. If your setup makes use of 4k video, make sure the monitor supports CEC+ARC in 4k format.



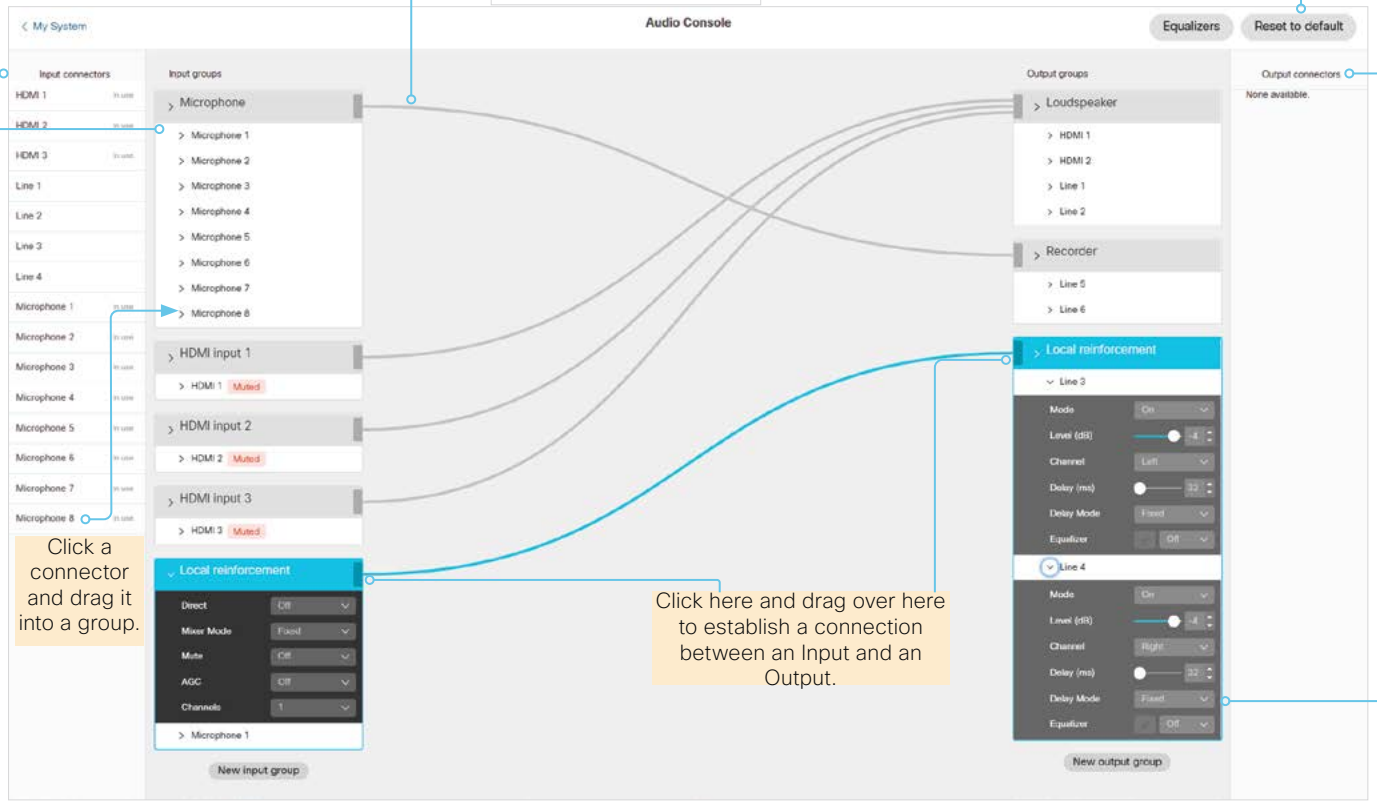
The Audio Console Panel

For systems using the Codec Pro, the **Audio Console** can be found under *Customization* in the Web interface of your device.

To delete a connection or adjust its gain, right-click on the "cable".



Reset to default will create a preset of connections based on connected devices detected.



The pool of physical *Input* connectors available.

The physical *Input* connectors are assigned to logical *Input* groups. These logical Input groups are created by you.

The name of a logical group is specified by you when clicking on *New input group*.

A given physical *Input* connector may be assigned to more than one logical group. Use drag-and-drop.

To remove a group or a member of a group, hover the mouse over the group or member. An X will appear. Click on this to remove the item.

Expand the logical group rather than the individual members when using it for local reinforcement.

Click a connector and drag it into a group.

Click here and drag over here to establish a connection between an Input and an Output.

This is the pool of physical *Output* connectors. Those that are not used are considered "inactive".

The logical *Output* groups are similar to the logical *Input* groups, but a physical *Output* cannot be assigned to more than one logical *Output* group.

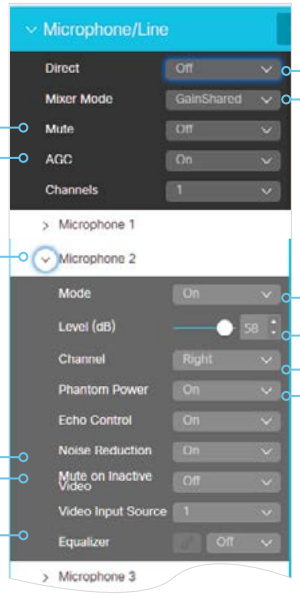
The Codec Pro offers Audio Return Channels (ARC) as additional choices.

We recommend that you set *Delay Mode* to *Fixed* to avoid potential delay caused by external screens.

In local reinforcement scenarios, set the microphones to *Direct* to bypass extra processing like *Echo Control*.

This will minimize latency and also serve to avoid that the Master volume control also affects the volume of the local presenter.

More on Setting up the Microphones



Mute the entire logical group.

Enable/Disable AGC (Automatic Gain Control) applying to the entire logical group.

Channels can be 1 (mono) or 2 (stereo). Applies to Codec Plus only.

You may expand a logical group to get access to settings on a group level.

In local reinforcement scenarios, make sure you set the Microphone(s) *Direct* to *On* to bypass extra processing like *Echo Control*. This will minimize latency and serve to avoid that the master volume control also affects the volume of the local presenter. For other use cases, leave it *Off*.

Mixer Mode can be *GainShared* or *Fixed*. If *GainShared*, the summed level of the microphone signals will never exceed a certain value, but their individual offsets will be retained. In *Fixed* mode, the levels are just summed together.

Mode: A microphone can be set to On or Off.

Level (dB) should be interpreted as *Gain* here.

If *Channels* has been set to 2 (at the Group level), you may use the *Channel* setting at the lower level to specify whether this particular microphone belongs to the group of left or right channel microphones.

The Codec Pro can do this with any type of input source, including microphones.

If the entire logical group has been set to *Direct* on group level, *Echo Control* will be set to N/A.

Phantom power:
This setting provides power submitted through the microphone cable. This can be switched off when microphones are used with a preamplifier providing line level signals.

Make sure you reduce the gain (typically by 40 dB or more) if you decide to use a Mic input as a Line input.

In the Codec Pro, a microphone input is a line input with *Phantom Feeding* activated.

Noise Reduction:
Setting this to *On* will reduce any continuous noise present in the room (e.g., fan noise). Impulsive noise will not be reduced.

Mute on Inactive Video: Use this feature in scenarios with, for example, a presenter's microphone. This microphone will be muted unless the associated camera (the one filming the presenter) actually sends video, provided that you have set up a camera/mic combination this way.

Equalizer: The *Equalizer* lets you choose among up to 8 predefined equalizer settings (or none, to indicate *Off*).

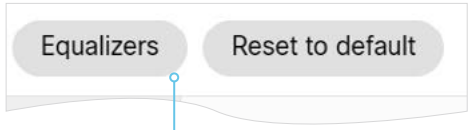
A graphical tool to set up the equalizer is described under Setting up the Equalizer.

Setting up the Equalizer

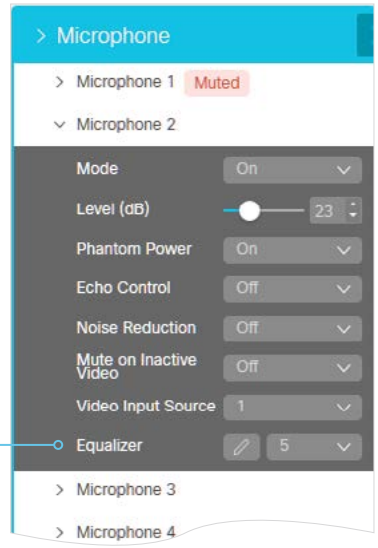
There are 8 user-definable parametric equalizer settings available.

A setting consists of up to 6 sections, each of which has its own filter type, gain, center/crossover frequency, and Q value.

Each section is shown with its own color. A white line shows the combined frequency response of the equalizer.



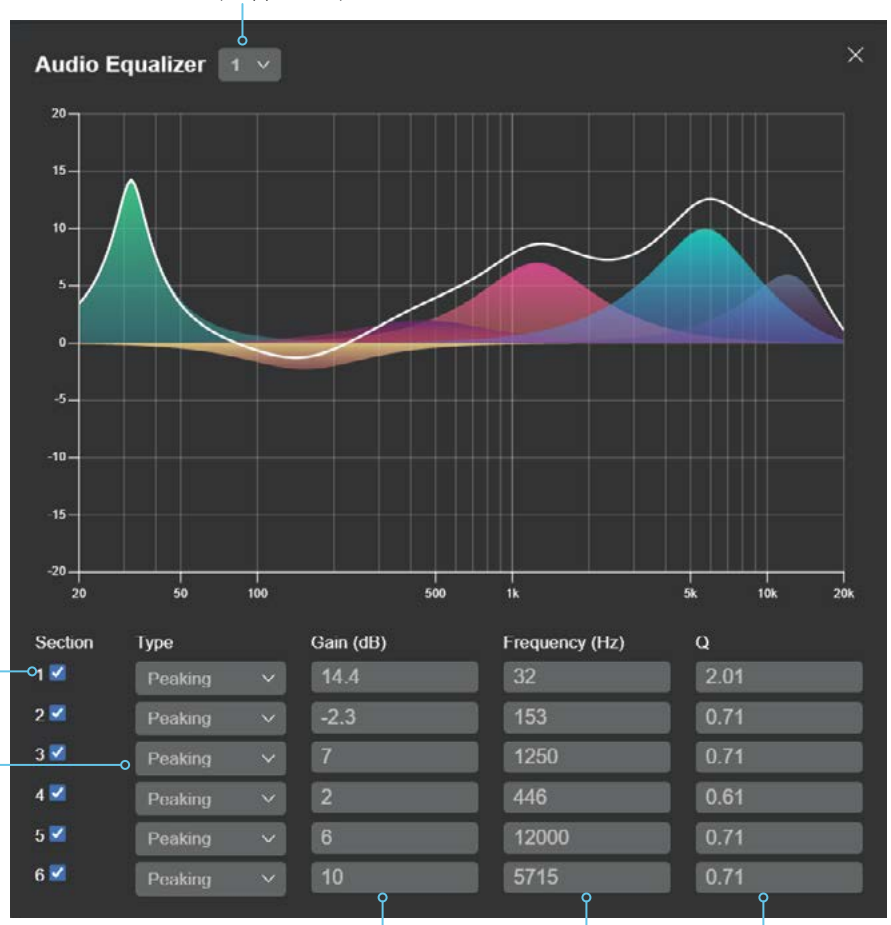
Gain access to the equalizer from here or here.



The effect of altering any of the parameters will become visible in the graph when you press Enter or select a different field.

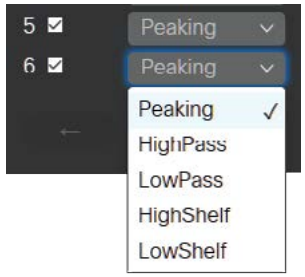
Your settings are saved automatically.

Go to the next setting (if applicable).



To activate or deactivate a filter section, click the corresponding checkbox.

Available filtering types



Value space for the Gain setting is 0 dB ±20 dB.

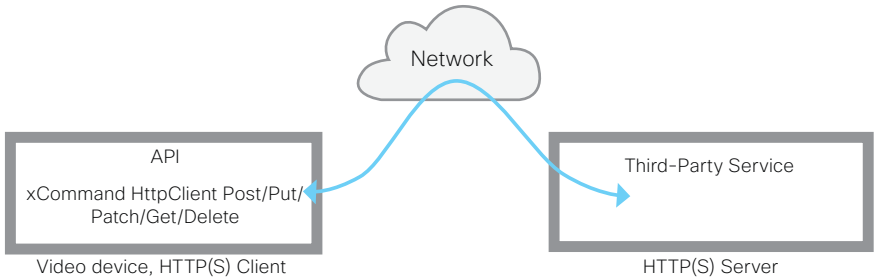
Value space for Frequency is 20–20 000 Hz.

Value space for Q is 0.1–10.0.

Examples

HTTP(S) Requests (page 1 of 3)

This feature makes it possible to send arbitrary HTTP(S) `Post` and `Put` requests from a video device to an HTTP(S) server.



By using API commands, you can send data to an HTTP(S) server whenever you want. You can choose what data to send and structure it as you like. In this way, you can adapt the data to an already established service.

Security measures:

- The HTTP(S) `Post/Put` feature is disabled by default. A system administrator must explicitly enable the feature by setting *HttpClient > Mode* to *On*.
- The system administrator can inhibit the use of HTTP by setting *HttpClient > AllowHTTP* to *False*.
- The system administrator can specify a list of HTTP(S) servers to which the video device is allowed to send data. For more information, see the `xCommand HttpClient Allow Hostname` commands.
- The number of concurrent `Post` and `Put` requests is limited.

List of Allowed HTTP(S) Servers

The system administrator can use these commands to set up and maintain a list of up to ten allowed HTTP(S) servers (hosts):

- `xCommand HttpClient Allow Hostname Add Expression: <Regular expression that matches the host name or IP address of the HTTP(S) server>`
- `xCommand HttpClient Allow Hostname Clear`
- `xCommand HttpClient Allow Hostname List`
- `xCommand HttpClient Allow Hostname Remove Id: <id of an entry in the list>`

If the list is not empty, you can send HTTP(S) requests to the servers in the list only. If the list is empty, you can send the requests to any HTTP(S) server.

The check against the list of allowed servers is performed both when using insecure (HTTP) and secure (HTTPS) transfer of data.

Allowing HTTPS without Certificate Validation

When sending requests over HTTPS, the video device checks the certificate of the HTTPS server by default. If the HTTPS server certificate is not found to be valid, you will get an error message. The video device will not send any data to that server.

We recommend using HTTPS with certificate validation. If this is not possible, the system administrator can set *HttpClient > AllowInsecureHTTPS* to *On* (`xConfiguration HttpClient AllowInsecureHTTPS: On`), which allows the use of HTTPS without validating the server's certificate.

HTTP(S) Requests (page 2 of 3)

Sending HTTP(S) Requests

Once the HTTP(S) Client Post feature is enabled, you can use the following commands to send requests to an HTTP(S) server. In the below text, *<Method>* is either `Post, Put, Patch, Get, or Delete`:

- `xCommand HttpClient` *<Method>* `[AllowInsecureHTTPS: <True/False>] [Header:<Header text>] [ResponseSizeLimit: <Maximum response size>] [ResultBody: <None/PlainText/Base64>] [Timeout: <Timeout period>] Url: <URL to send the request to>`

Adding header fields is optional, but you can add as many as twenty fields.

The `AllowInsecureHTTPS` parameter is effective only if the system administrator has allowed the use of HTTPS without validating the server's certificate. If so, you can send data to the server without validating the server certificate if the parameter is set to True. If you leave out the parameter, or set it to False, data is not sent if the certificate validation fails.

The `ResponseSizeLimit` parameter is the maximum payload size (bytes) that the device accepts as a response from the server. If the response payload is larger than this maximum size, the command returns a status error with a message saying that the maximum file size is exceeded. However, this has no effect on the server side. The request was received and processed properly by the server.

Use the `ResultBody` parameter to decide how to format the body of the HTTP response from the server in the command result.

You have three options:

- *None*: Do not include the body of the HTTP response in the command result.
- *Base64*: Base64 encode the body before including it in the result.
- *PlainText*: Include the body in the result as plain text. If the response contain non-printable letters, the command returns a status error with a message saying that non-printable data was encountered.

Use the Timeout parameter to set a timeout period (seconds). If the request is not completed during this period, the API will return an error.

Enter the payload (data) right after you have issued the command. Anything that you enter, including line breaks, is part of the payload. When done, finish with a line break ("\n") and a separate line containing just a period followed by a line break (".\n"). Now the command is executed, and the data is sent to the server.

All commands and configurations are described in detail in the API guide for your product.

HTTP(S) Requests (page 3 of 3)

Examples

Example 1: IoT Device control using HTTP Post.

The following is a macro function that turns on a light that is connected to a Philips Hue Bridge.

The body of the message is JSON in this example. It could be any format, depending on the expected format of the service that is receiving the messages.

```
function hue_command(data) {
  var url = 'http://192.0.2.10/api/'ZXlU4tUtQ23Pjbdyl-kiyCjTs0i5ANDEu1ypJq0-/
lights/1/state';
  var headers = 'Content-Type: application/json';
  var command = '{"on":true}';
  xapi.command('HttpClient Put',
      {'Url': url, 'Header': headers }, command);
}
```

You can do the same at the command line using the API:

```
xcommand HttpClient Put Header: "Content-Type: application/json" URL:
"http://192.0.2.10/api/'ZXlU4tUtQ23Pjbdyl-kiyCjTs0i5ANDEu1ypJq0-/lights/1/state"
{"on":true}
```

Example 2: Posting data to a monitoring tool using HTTP Post

```
xcommand HttpClient Post Header: "Content-Type: application/json" URL:
"https://mymonitoringserver.com/service/devicemonitoring"

{"Message":"A user reported an issue with this system","systemName":"BoardRoom
4th floor","softwareVersion":"ce9.6.0","softwareReleaseDate":"2018-06-
29","videoMonitors":"Dual"}
```

Example 3: Using a macro to sent HTTP Post

```
function sendMonitoringUpdatePost(message){
    xapi.command('HttpClient Post',
        {'Header': 'Content-Type: application/json',
         'Url':MONITORING_URL},
          JSON.stringify(Object.assign(
                {'Message': message}, systemInfo)));
}
```

Removing Default Buttons (page 1 of 3)

This feature adds the ability to hide default feature buttons in the UI while still exposing custom control buttons. This allows for a more customizable UI.

Even here, you will need local admin access to the device to use the xConfigurations.

More specifically, what this feature does is to add a series of configurations that allow you to hide/display certain feature buttons in the UI that have previously not been possible to hide, including the button to turn video On/Off.

Functional Overview

Enter the following command in the web interface of the video device to see what configurations are set:

```
xconfig  //UserInterface/Features
```

Enter the same command with the addition of a '?' to show the available options. For example:

```
xconfig //UserInterface/Features/ ?
```

For example, you may see:

? xConfiguration UserInterface Features Call End:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features Call JoinWebex:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features Call Keypad:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features Call MidCallControls:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features Call Start:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features HideAll:* <False**, *True*>

? xConfiguration UserInterface Features Share Start:* <Auto**, *Hidden*>

? xConfiguration UserInterface Features Whiteboard Start:* <Auto**, *Hidden*>

OK

On each line, the default configurations are indicated in **bold**. The text following "*?" is the command you can run. Replace the <...> with the value you will use. For example, to hide the *Call* button:

```
xConfiguration UserInterface Features Call Start: Hidden
```

If you choose hide the *Call* button, this will also hide the default UI feature for making a call or doing directory lookups / favorites / recent calls, etc. In addition, the *Add* button, which is used to add participants while in a call, will be hidden.

The MidCallControls are *Hold*, *Transfer* and *Resume*.

Hiding the *Share* button will hide the default UI for sharing as well as hiding the ability to preview sources in- and out-of-call.

Limitations

The feature applies to the *Call, MidCallControls,* and *Share* sets of buttons only.

You cannot hide other single buttons whose display is a result of certain use cases, such as *Meetings, Extension Mobility, Voicemail*, etc. These buttons must either be all displayed or all hidden.

If you choose to hide them all, only your custom-made buttons will be shown.
To do this, use:

```
xConfiguration UserInterface Features HideAll: True
```

You may circumvent this all-or-nothing problem through selective provisioning of the settings from the back-end.

Note! This feature is about removing buttons only. The functions themselves are not removed. For instance, although the *Share* button may be removed, the function will still be accessible via Proximity.

Removing Default Buttons (page 2 of 3)

An Out-of-Call Example

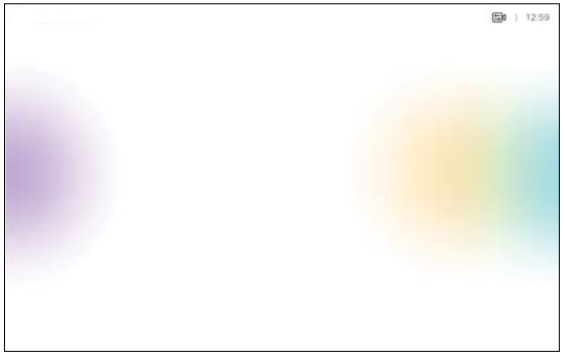
Assume that we want to create a scenario where users are limited to call a few specific rooms only. This could be the case in companies where external calls are never made. All calls are assumed to take place between this limited number of rooms.

Starting out with a standard UI like this:

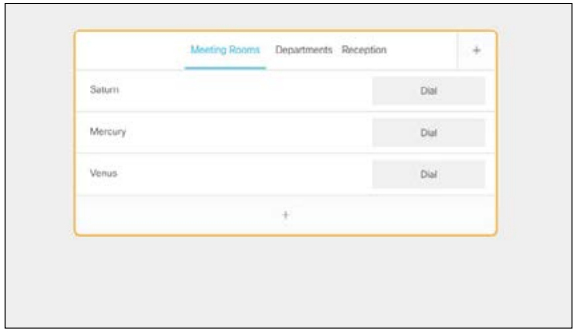
If we now issue the following command:

```
xConfiguration UserInterface Features HideAll: True
```

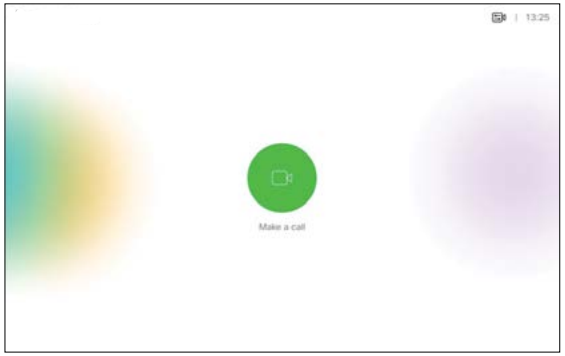
The UI will look like this:



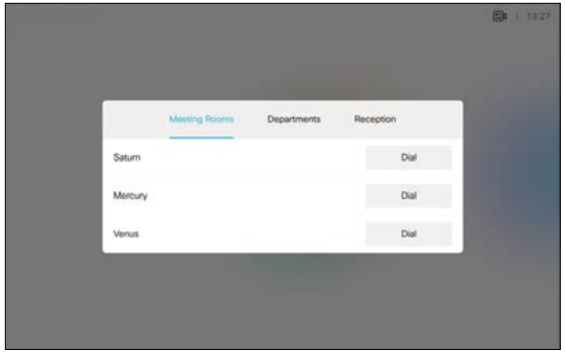
Now, let's open the **UI Extensions Editor** and create this panel:



The name of the panel is "Make a call". This will appear below the button when it is exported to the video device. For example:



If you now tap the *Make a call* button on the user interface, the following panel will appear:



As a user, there is not much that you will be able to do here, but that is intentional. No mistakes or random calls are possible. To call any of the three rooms, just tap *Make a Call* followed by *Dial* next to the name of the room to be called.

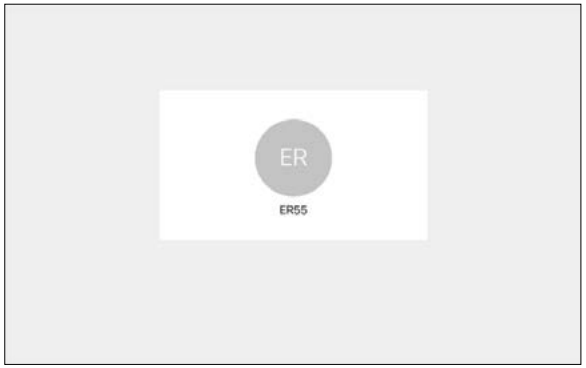
To actually create and use this panel, you will either need an external control device or a **Macro**. Macros are described in the Macros section of this document.

Since our new panel was for use outside of calls, we still need to define a panel or buttons for the user to have when in a call. Otherwise, ther will be no way to end the call. See An In-Call Examplefor a example showing how to set this up.

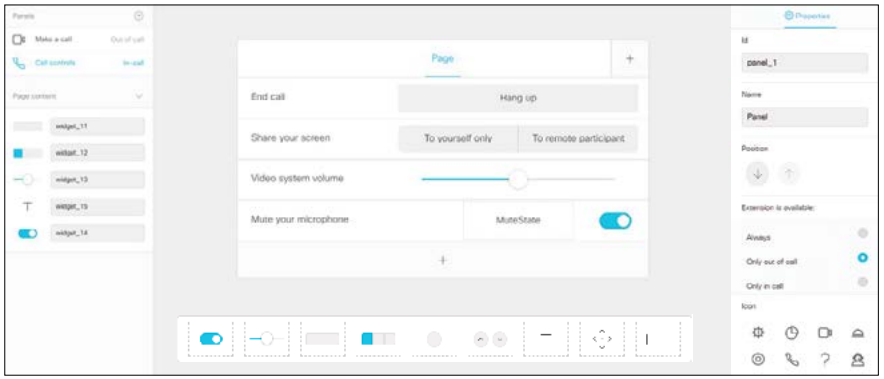
Removing Default Buttons (page 3 of 3)

An In-Call Example

As described on the previous page, we need to define in-call behavior as well. Otherwise, we will meet the following when in a call (no way to end the call):

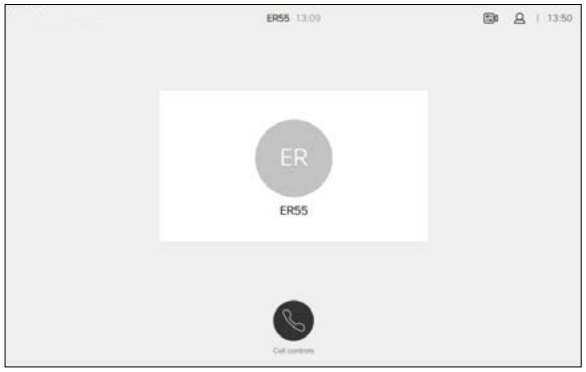


Let's create a setup:

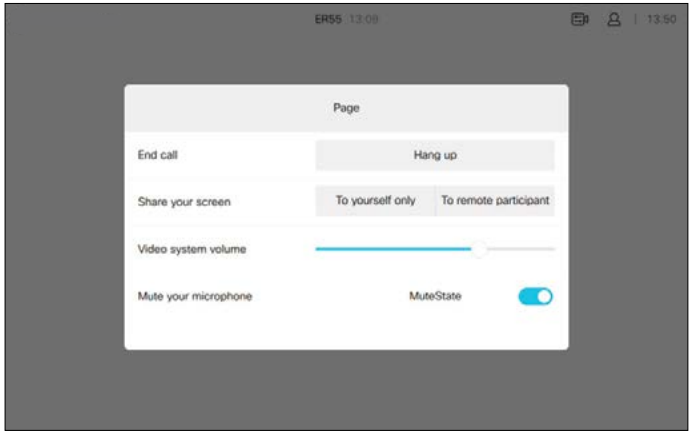


This button is set to appear in calls only.

Push this to the video device and the required functionality will be available:



In this example, once you tap the *Call control* button, the following will be displayed:



These are just a few examples of the many scenarios you can create.

Interactive Messages (page 1 of 3)

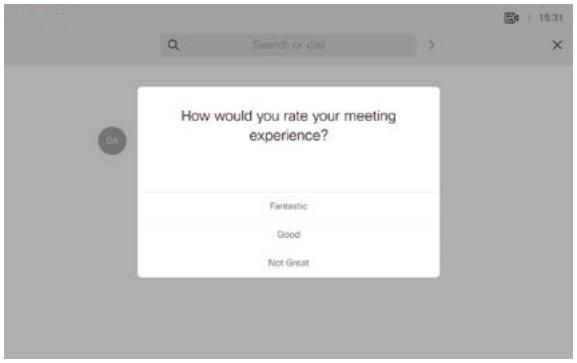
The Interactive Messages feature lets you create alerting and/or interactive messages on the user interface; thus, prompting the user to act accordingly.

If you want to create a sequence of messages where the next message depends on a choice made in the previous message, we recommend the use of macros to create events to act upon. Alternatively, you may use an external control device, which then will act upon the events created.

In order to submit inputs from the user, you should make use of HttpFeedback.

The `HttpFeedback` causes the device to post HTTP feedback messages (also known as "webhooks") upon changes to the API state (e.g., statuses, events, and configuration updates). The HTTP Post feedback messages will be sent to the specified ServerURL.

Example 1: Rating Your Experience



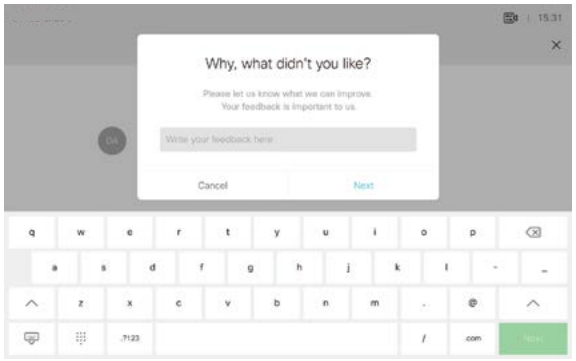

```
xCommand UserInterface Message Prompt Display FeedbackId:
"MeetingExperience" Text: "" Title: "How would you rate your meeting
experience?" Option.1: "Fantastic" Option.2: "Good" Option.3: "Not Great"


*e UserInterface Message Prompt Response FeedbackId: "MeetingExperience"
*e UserInterface Message Prompt Response OptionId: 1


<XmlDoc resultId="">
<Event>
 <UserInterface item="1">
  <Message item="1">
   <Prompt item="1">
    <Response item="1">
     <FeedbackId item="1">MeetingExperience</FeedbackId>
     <OptionId item="1">1</OptionId>
    </Response></Prompt></Message>
 </UserInterface> </Event>
</XmlDoc>
```

Interactive Messages (page 2 of 3)

Example 2: Request User Input



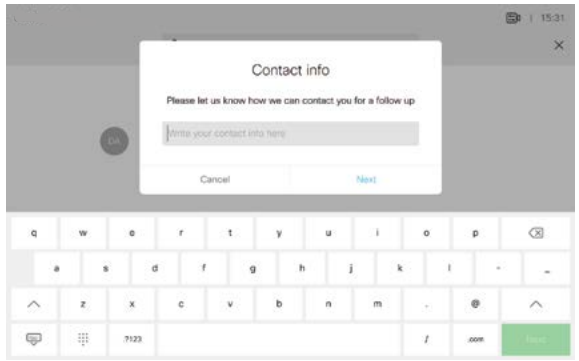

```
xCommand UserInterface Message TextInput Display FeedbackId:
"MeetingFeedback" Placeholder: "Write your feedback here" SubmitText: "Next"
Title: "Why, what didn't you like?" Text: "Please let us know what we can
improve. Your feedback is important to us."


*e UserInterface Message TextInput Response FeedbackId: "MeetingFeedback"
*e UserInterface Message TextInput Response Text: "Low resolution"


<XmlDoc resultId="">
<Event>
 <UserInterface item="1">
  <Message item="1">
   <TextInput item="1">
    <Response item="1">
     <FeedbackId item="1">MeetingFeedback</FeedbackId>
     <Text item="1">Low resolution</Text>
    </Response></TextInput></Message>
 </UserInterface></Event></XmlDoc>
```

When you create messages as shown here, you may specify the text on the "Next" button. The "Cancel" button; however, appears by default and its text cannot be altered.

Example 3: Getting In Touch With You



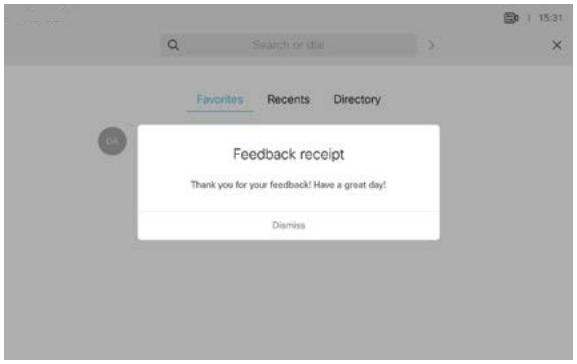

```
xCommand UserInterface Message TextInput Display FeedbackId: "ContactInfo"
Placeholder: "Write your contact info here" SubmitText: "Next" Title:
"Contact Info" Text: "Please let us know how we can contact you for a follow
up"


*e UserInterface Message TextInput Response FeedbackId: "ContactInfo"
*e UserInterface Message TextInput Response Text: "john@go.webex.com"


<XmlDoc resultId="">
<Event>
 <UserInterface item="1">
  <Message item="1">
   <TextInput item="1">
    <Response item="1">
     <FeedbackId item="1">ContactInfo</FeedbackId>
     <Text item="1">john@go.webex.com</Text>
    </Response></TextInput></Message>
 </UserInterface></Event>
</XmlDoc>
```

Interactive Messages (page 3 of 3)

Example 4: Feedback Receipt




```
xCommand UserInterface Message Alert Display Title: "Feedback receipt" text:
"Thank you for your feedback! Have a great day!"

*e UserInterface Message Alert Cleared

<XmlDoc resultId="">
<Event>
  <UserInterface item="1">
    <Message item="1">
      <Alert item="1">
        <Cleared item="1"/>
      </Alert>
    </Message>
  </UserInterface>
</Event>
</XmlDoc>
```

When you create messages using `Message Alert`, the "*Dismiss*" button will appear by default. The text on this button cannot be altered.

Example 5: Enter Your Webex Pin


```
xCommand UserInterface Message TextInput Display FeedbackId: "WebexPin"
InputType: Numeric Placeholder: "Please enter the host pin PIN" SubmitText:
"Submit PIN" Text: "Please enter the host pin PIN, followed by #. Not the
host: Press #" Title: "Webex Pin"

*e UserInterface Message TextInput Response FeedbackId: "WebexPin"
*e UserInterface Message TextInput Response Text: "1122#"

<XmlDoc resultId="">
<Event>
 <UserInterface item="1">
  <Message item="1">
   <TextInput item="1">
    <Response item="1">
     <FeedbackId item="1">WebexPin</FeedbackId>
     <Text item="1">1122#</Text>
    </Response></TextInput></Message>
 </UserInterface>
</Event>
</XmlDoc>
```

Third-party USB Control Devices (page 1 of 3)

This feature lets you use a third-party USB input device to control certain functions and behavior on a video device. Examples of such devices could be a Bluetooth® remote control (with a USB dongle) or a USB keyboard.

Note! To make this work you must define and implement any action the use of such input devices is meant to cause—there is no such as a library of actions readily available to pick actions from.

This feature is not meant to replace the touch screen, but it may complement parts of their functionalities, wherever convenient.

Why this Feature?

Examples of applications for this feature will typically be:

- In classrooms and during lectures, a small remote control can be used to wake up systems from standby mode and select which input source to present.
- Control of pan, tilt and zoom of a camera in situations where Touch 10 is an inconvenient alternative or not allowed to be used (for instance in operating rooms in hospitals combined with tele-medicine).

See Example Use of a Third-Party USB Input Device for a practical example.

Input Device Requirements

The input device must advertise itself as a USB keyboard. The term keyboard should not necessarily be understood literally here— a Bluetooth remote control with a USB dongle will do the trick.

Functional Overview

Pressing a button on the USB input device will generate an event in the API. Macros or third-party control devices can listen for such events and respond to them. This is similar to the buttons on the Touch 10 user interface. It is also possible to listen for the events via webhooks, directly in an SSH session.

You must implement the actions to be taken as response to these events. For example:

- Increase the volume of the video device when the *Volume Up* key is pressed.
- Put the video device in *Standby Mode* when the *Sleep* key is pressed.

Note! The support for third-party USB input devices is disabled by default. This means that you must explicitly enable it. Set the *Peripherals > InputDevice > Mode to On*.

Pressing and releasing a button will generate a `Pressed` and `Released` events, like this:

```
*e UserInterface InputDevice Key Action Key: <Name of key>
*e UserInterface InputDevice Key Action Code: <Id of key>
*e UserInterface InputDevice Key Action Type: Pressed
** end
*e UserInterface InputDevice Key Action Key: <Name of key>
*e UserInterface InputDevice Key Action Code: <Id of 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 input device is detected by the video device, it shows up in the *UserInterface > Peripherals > ConnectedDevice* status. Note that the input device may be reported as multiple devices.

Third-party USB Control Devices (page 2 of 3)

Example Use of a Third-Party USB Input Device

This example shows how to use the keys of a third-party USB input device (in this case a remote control) to control the standby function, increase and decrease the volume, as well as to control the camera of a room or desk device.

The macro created will listen for relevant events and carry out the associated actions using the API of the room or desk device.

Note! In the command examples below, the text in normal font is entered by you and the text in italics is the response received from the room device.

1. Sign in to the room or desk device on SSH. You need a local admin user.
2. Configure the device to allow the use of a 3rd party USB remote control.

   ```
   xConfiguration Peripherals InputDevice Mode: On
   ```

   *** end*

   OK

Note! 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

Note! 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 generates two different 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 are the events 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*

5. Write a macro that listens for the relevant *InputDevice* events and carries out the associated actions using the API of the device (this is shown on the next page).

   a. 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.

   b. 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 should stop.

   When the macro sees an event containing `KEY _ LEFT`, `KEY _ RIGHT`, `KEY _ UP`, or `KEY _ DOWN`, it executes the related commands.

(Continued on the next page)

Third-party USB Control Devices (page 3 of 3)

The parts related to the camera control function are identified in blue below.

```
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();=
```

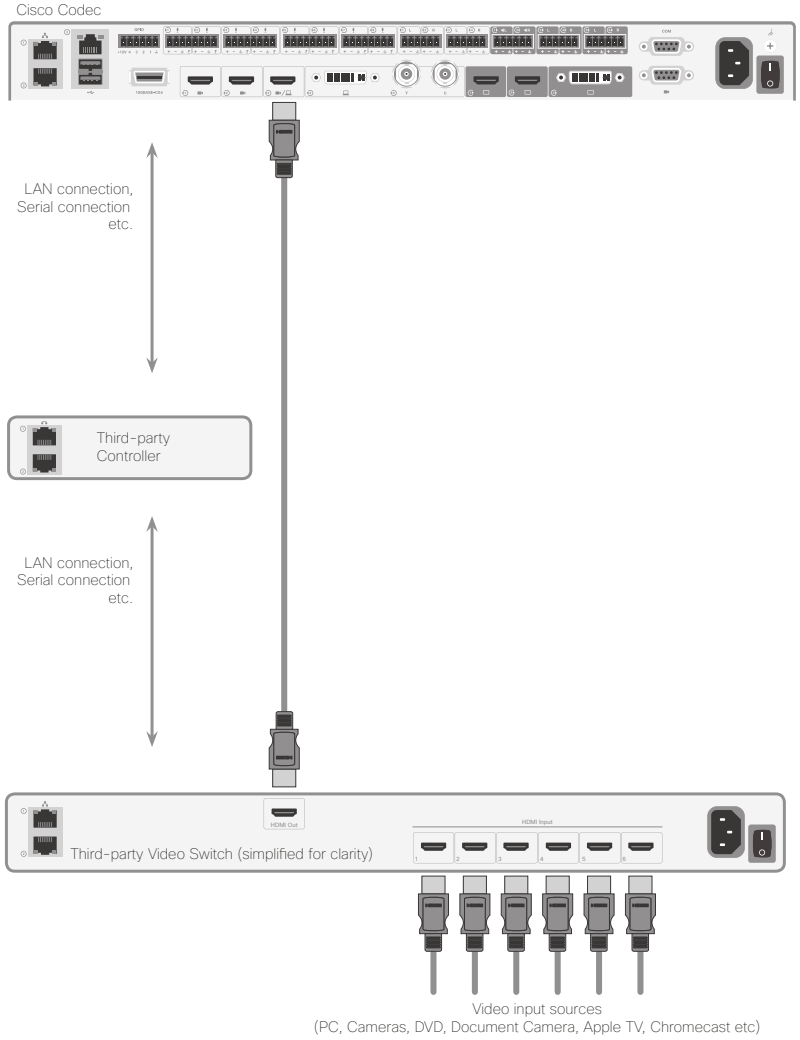
Use of a Video Switch (page 1 of 4)

Using a Third-party Video Switch to Extend the Number of Video Sources Available

The **UI Extensions Editor** can configure your system to show video sources from a third-party external video switch in the normal *Share Tray* view.

The sources will appear, and behave, as any other video source connected directly to the video device. For the user, this will be perceived as completely transparent and no video switch will seem to be involved.

See Video Switch Example for a simple example of a setup using the configuration.



Use of a Video Switch (page 2 of 4)

API Commands and Events

The video switch feature requires a third-party control system. The control system will use the API to synchronize the source states between the video switch and your system's user interface using a set of API events and commands.

In order to make this work when the user selects a video source, the video device must be set to issue a corresponding event. This, in turn, shall cause the controller to send appropriate commands to the video switch and the video device.

This event will be issued only if the controller has registered to the video device upon connection, requesting the following from the video device:

```
xFeedback register Event/UserInterface/Presentation/ExternalSource
```

The event issued will be as follows:

```
*e UserInterface Presentation ExternalSource Selected SourceIdentifier:
"XXXX"
```

Where "XXXX is a unique string ID used to identify this source when selecting or setting state.

There are six commands available to control the system:

Add: Adds video source identifiers, including ID of connector, the name to appear on the screen, a unique string ID to identify a source when selecting or setting state, and what type of icon to display on the screen for each source.

List: Returns the current list of external sources.

Remove: Removes a source from the list.

RemoveAll: Removes all of the sources from the list.

Select: Selects a specific source.

State Set: Changes the state of a source.

See the UserInterface Presentation ExternalSource API Commands for more details.

Use of a Video Switch (page 3 of 4)

UserInterface Presentation ExternalSource API Commands

UserInterface Presentation ExternalSource Add

This command establishes and defines an input source.

```
xcommand UserInterface Presentation ExternalSource Add ConnectorId:
ConnectorId Name: Name SourceIdentifier: SourceIdentifier Type: Type
```

Arguments:

ConnectorId: The ID of the video device connector to which the external switch is connected

Name: Name displayed on the screen

SourceIdentifier: A unique string ID used to identify this source when selecting or setting state

Type: Decides what icon is displayed on the screen, choose between: <pc/camera/desktop/document_camera/mediaplayer/other/whiteboard>

Example:

```
xcommand UserInterface Presentation ExternalSource Add ConnectorId: 3
Name: "Blu-ray"
SourceIdentifier: bluray Type: mediaplayer
```

UserInterface Presentation ExternalSource List

This command returns the current list of external sources.

```
xcommand UserInterface Presentation ExternalSource List
```

UserInterface Presentation ExternalSource Remove

This command removes a source from the list.

```
xcommand UserInterface Presentation ExternalSource Remove SourceIdentifier:
SourceIdentifier
```

Arguments:

SourceIdentifier is a unique string ID used to identify this source when selecting or setting state.

UserInterface Presentation ExternalSource RemoveAll

This command removes all sources from the list.

```
xcommand UserInterface Presentation ExternalSource RemoveAll
```

UserInterface Presentation ExternalSource Select

Starts to present the selected source if it is in ready state and has a valid ConnectorId. Also shows the item in the share tray as "Presenting".

```
xcommand UserInterface Presentation ExternalSource Select SourceIdentifier:
SourceIdentifier
```

Arguments:

SourceIdentifier is a unique string ID used to identify this source when selecting or setting state.

UserInterface Presentation ExternalSource State Set

Used to change state of the source with SourceIdentifier.

```
xcommand UserInterface Presentation ExternalSource State Set
SourceIdentifier: SourceIdentifier State: State [ErrorReason: ErrorReason]
```

Arguments:

SourceIdentifier: is a unique string ID used to identify this source when selecting or setting state

State: <Error/Hidden/NotReady/Ready> Ready is the only presentable state, hidden exists in the list but does not show in the share tray.

ErrorReason: Optional. Displays in the share tray if the state is set to `Error`.

Use of a Video Switch (page 4 of 4)

Video Switch Example

Controller sending:

```
xcommand UserInterface Presentation ExternalSource Add ConnectorId: 3
Name: "Blu-ray" SourceIdentifier: bluray Type: mediaplayer

xcommand UserInterface Presentation ExternalSource Add ConnectorId: 3
Name: "Apple TV" SourceIdentifier: appletv Type: mediaplayer

xcommand UserInterface Presentation ExternalSource Add ConnectorId: 3
Name: "TV" SourceIdentifier: tv Type: mediaplayer
```

The default state is `NotReady` (Fig. 1)

So, the next step for an integrator would be to set them to ready (Fig. 2).

```
xcommand UserInterface Presentation ExternalSource State Set State: Ready
SourceIdentifier: bluray

xcommand UserInterface Presentation ExternalSource State Set State: Ready
SourceIdentifier: appletv

xcommand UserInterface Presentation ExternalSource State Set State: Ready
SourceIdentifier: tv
```

If one of the sources is selected on the video switch the controller should send a command accordingly:

```
xcommand UserInterface Presentation ExternalSource Select
SourceIdentifier: tv
```

If the switch is connected on the chosen connector, it will start to present (Fig. 3).

When a user selects another source by clicking the other source item in the share tray, the video device will send the following event:

```
*e UserInterface Presentation ExternalSource Selected SourceIdentifier:
"appletv"
```

The *Controller* should listen to this event and display the selected source.

Note! The presentation will not start if the below setting has been set to `Manual`:

```
xconfiguration Video Input Connector [x]
PresentationSelection: <AutoShare, Desktop, Manual, OnConnect>
```

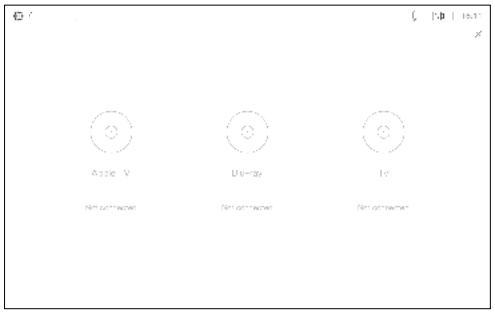


Fig. 1 Default state is NotReady.

Fig. 2 When Input sources have been set to Ready.

Fig. 3 If the switch is connected on the chosen connector it will start to present.

Example Strategies for Room Controls

Control of Lights

The combination of a slider and a toggle button could be used to control lights. The toggle button switches the lights on or off; the slider serves as a dimmer.

Consider the following strategy:

- Set a slider to minimum when the user turns the lights off.
- Set a toggle button to off when the user moves the slider to its minimum.
- Remember the value of the slider when the lights are turned off, so that you can return to this value when the lights are turned back on again.
- If the light is at 40 %, when the user switches it off, he or she would expect it to go back to 40 % (not maximum) when switching the lights on again.
- When the user selects one of the options in the group button (a light preset), set the sliders and toggle buttons accordingly.
- If the lights are changed away from a preset, for instance by changing a slider or toggle button, deselect all options in the group button.

Control of Temperature

The combination of a spinner and a large font text box (value) may be used to control temperature. Use the spinner to set the desired temperature, and the large font text box to show the current temperature.

For the best user experience keep the following in mind:

- Update the large font text box as the temperature in the room changes. Update the text field of the spinner when someone tap the up and down arrows
- Consult the Widgets chapter for details about how to update the spinner's text field and the large font text box.

Control of Blinds

You can either use a spinner, or up and down arrows from the Icons tab in the widget library.

Consider the following strategy:

- Tilt the slides as response to a short press on a direction arrow. If tilted all the way, move the blinds up or down incrementally.
- As response to a long press on a direction arrow, start moving the blinds in that direction. They do not stop until all the way up or down.
- Short press any button in order to stop the movement after a long press. Then no separate stop button will be needed.

Online Resources

YouTube

- https://www.youtube.com/playlist?list=PL_YnWo4XhzTe8CyOppyhheB8UN40mDYiX

xAPI samples

- https://github.com/CiscoDevNet/roomdevices-macros-samples

awesome-xapi: curated community resources

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

Intellectual property rights

Cisco contacts

On our web site you will find an overview of the worldwide Cisco contacts.

Go to: ▸ https://www.cisco.com/go/offices

Corporate Headquarters

Cisco Systems, Inc.

170 West Tasman Dr.

San Jose, CA 95134 USA