

Contents

TouchControl Server Setup (Windows)	11
TouchControl Server Setup (macOS)	17
iOS Device Setup	19
Locations and Activities	21
Locations	21
Activities	22
Windows Server:	22
Visible	22
Not visible	22
Not included	22
Activity Configuration	22
iPhone	23
iPad	23
Watch	23
Scale to fit	23
Zoom to Width	23
Visible	23
Safe Area	24
Shadows	24
Haptics	24
Mouse/Keyboard	24
Devices and Buttons	26
Devices	26
Buttons	27
Primary Buttons	27
IR buttons	27
Command buttons	27
AutoHotKey buttons	27
EventTrigger buttons	27
Global Caché buttons	27
HTTP Request buttons	27
iRTrans buttons	27
Composite Buttons	28
Macro buttons	28
Slider Buttons	28
Spinner Buttons	28
Gesture Pad buttons	28
Auxiliary Buttons	28
Link to Activity buttons	28
Labels	28
Web Views	28

<i>URL buttons</i>	28
<i>Feedback Client buttons</i>	28
<i>Feedback Listener buttons</i>	28
<i>Script buttons</i>	28
<i>Group buttons</i>	28
<i>Text Field buttons</i>	28
Repeating Buttons.....	28
Timer Buttons.....	29
Feedback Buttons.....	29
Adding Buttons.....	30
Configuring Buttons.....	31
2-Stage Buttons.....	31
IR Buttons (Windows server only)	32
Learning IR Codes.....	32
IR Transmit Repeat Factor.....	33
Manual IR Codes.....	33
Pre-Script & Post-Script.....	33
Command Buttons	33
Feedback Script.....	34
Pre-Script & Post-Script.....	34
iOS Device Commands.....	34
<i>Always On mode</i>	34
<i>Full Screen mode</i>	34
<i>Screen Brightness</i>	35
<i>Activity Locking</i>	35
<i>Screen Scrolling</i>	36
<i>Keyboard Control</i>	36
<i>Image Picker</i>	36
Server-side Commands (Windows server only).....	37
<i>Server Sleep</i>	37
<i>Screen Grabber Control</i>	37
<i>USB-UIRT IR Code Re-broadcast</i>	38
AutoHotKey Buttons (Windows server only)	38
AutoHotKey Feedback.....	39
Keystroke Helpers.....	40
QuickEdit.....	40
Pre-Script & Post-Script.....	40
EventTrigger Buttons	40
HEX Commands:.....	41
Pre-Script & Post-Script.....	41
EventGhost.....	41
Global Caché Buttons	41
Add a Global Caché interface in TouchControl Server.....	42
Add Global Caché buttons to your activities.....	42

Pre-Script & Post-Script.....	46
HTTP Request Buttons.....	46
Add HTTP Request buttons	46
Pre-Script & Post-Script.....	48
iRTrans Buttons	48
Add iRTrans buttons to your activities	48
"Edit" mode.....	49
"Learn" mode.....	49
"Import" mode.....	50
Pre-Script & Post-Script.....	51
Macro Buttons.....	51
Blocking vs. non-blocking pauses.....	51
MacroMessage.....	52
Press & Release macros	53
Release-only buttons.....	53
Slider Buttons	53
Action when sliding	54
Action at each stop.....	54
"Snap to touch" slider interaction.....	55
Action on release.....	55
TouchTips	55
Script.....	55
Spinner Buttons	56
Numeric: 0-9.....	56
Numeric: Min-Max	56
Buttons	57
Free Text.....	57
Grids	58
Gesture Pad Buttons	58
Hold To Repeat	59
2-Stage Buttons.....	60
Gesture Pad Mousepad (Windows server only)	60
Redirect Mouse & Keyboard Control	61
Swipe Velocity	62
Link to Activity Buttons.....	62
Dynamic Links.....	63
Link Pre-Script.....	64
Background Links.....	64
Popover Links	65
Custom transitions	65

Labels65

Web Views	65
Dynamic Variable Substitution	67
Refreshing a Web View	68
Alter a Web View's Identity.....	68
Web View Script	69
URL buttons.....	69
Feedback Client Buttons	69
Feedback Listener Buttons.....	70
Script Buttons	71
Group Buttons	72
TouchMotion	72
Group Edit Mode.....	74
Hide In Designer	74
Templates	74
Text Fields	75
Designing Layouts.....	79
Background Image	79
Adding Buttons	81
<i>Button Packs</i>	81
Selecting buttons.....	81
<i>Multi-select mode</i>	81
<i>Drag-select</i>	82
<i>Selecting group buttons</i>	82
Cloning buttons	82
Copy & paste	82
Resizing buttons	82
<i>Fast resizing</i>	83
<i>Proportional resizing</i>	83
Replacing buttons.....	83
Designer transparency	83
Rotating buttons.....	83
Configuring Buttons	84
Right-click menu	84
Enabling/disabling buttons	84
Button Images:	84
<i>Pressed Image</i>	84
<i>Icons</i>	84
<i>Reset size</i>	85
<i>Set background color</i>	85
<i>Disable animation</i>	85

Button text	85
<i>Show/hide text</i>	86
<i>Text size</i>	86
<i>Text color</i>	86
<i>Text font and alignment</i>	86
<i>TouchTips</i>	86
Size & Location	87
Delay touch.....	87
Auto Exec.....	87
Layout	88
<i>Bring to front/Send to back</i>	88
<i>Center horizontally/Center vertically</i>	88
Align	88
Stationary	88
Propagate	88
Properties	88
Design-Time Features.....	89
Snap to grid	89
Undo all changes	89
Find	89
Undo button	89
Button border color.....	89
Background variable.....	90
Designer Hotkeys (Windows server only)	90
Saving Layouts	91
<u>Interface Manager</u>	<u>92</u>
<u>Template Manager</u>	<u>96</u>
<u>Server Tools</u>	<u>97</u>
PC Remote Control (Windows server only).....	97
Listening (Windows server only).....	97
Import/Export.....	97
Reload Configuration.....	97
Migrate Activity Buttons (Windows server only)	98
Arrange Locations & Activities (Windows server only)	98
Import Global Caché Buttons From iLearn (Windows server only).....	98
Import Global Caché Buttons From Database	99
Find Unused Buttons (Windows server only)	99

Checking for New TouchControl Server Version (Windows server only)	100
TouchControl Server configuration backup/recovery (Windows server only)	100
Backgrounds and Button Packs	101
Background Images	101
Button Packs	102
Scripting	104
Editing Script	104
Feedback Script	104
The Return String	105
Button text/image/icon:	105
Execute a button	106
Additional timer button flags	107
Set a local variable	108
Set a global variable	109
Cancellation elements	110
Examples	110
Full Script Examples	111
Feedback Flags	112
Feedback Slicing	114
Updating activity background images via script	115
Determining device network status via script	116
Disable full-screen activity "go-back" swiping via script	116
Advanced Scripting	118
Button pre-script and post-script	118
Custom Button Properties	119
Built-In Button Properties	121
<i>For buttons that display text</i>	121
<i>For Slider buttons</i>	122
<i>For HTTP Request buttons</i>	122
<i>For all button types that render as a button on the screen</i>	122
<i>For Gesture Pad buttons</i>	123
<i>For Group buttons</i>	123
<i>For _deviceMotion buttons</i>	123
<i>For _mouseMoveButton buttons</i>	123
<i>For _macroMessage buttons</i>	124

<i>For all buttons on a watch activity</i>	124
<i>For Multi-Peer buttons</i>	124
Custom script libraries.....	124
Button Script Variables	126
Helper functions	128
Other available non-button-specific script variables and helper functions	131
Local/Global Variables, and State Variables via iCloud.....	133
Local Variables.....	133
Global Variables.....	133
State Variables and iCloud	134
Script Handlers	134
Handle socket disconnects/re-connects with script	134
Handle iPad rotation with script	135
Global Watchers	136
Apple Watch	138
Activity configuration	138
Watch Button configuration.....	139
Watch Haptics	142
Apple Watch interface styles for TouchControl.....	142
List 1	142
List 2	142
List 3	142
Grid.....	142
Style 1	142
Style 2	142
Style 3	143
Style 4	143
Style 5	143
Miscellaneous Topics	144
Sizes	145
Liquid Glass	146
Background Slideshow	147
Activity & Device Sharing	149

Network PING	150
Connectable.....	151
LocationManager	152
Device Battery Monitor.....	153
Speech Synthesis.....	154
Siri Integration	156
IFTTT Webhooks Integration	158
External Mousepad.....	159
Simple Service Discovery Protocol (SSDP).....	161
Integrated Global Caché IR Database	163
Device Motion Sensing.....	164
Designer Device & Button Search (Windows server only)	166
Device Search	166
Button Search.....	166
Grid Buttons.....	167
Examples.....	168
Multi-Peer (Nearby Networking).....	169
The Multi-Peer Session.....	169
Multi-Peer Button Properties.....	170
Sender-only EventTrigger Properties:	170
<i>MPSendDataMode</i>	170
<i>MPAutoAccept</i>	171
Sender and Receiver EventTrigger Properties:	171
<i>MPCConnectScript</i>	171
<i>MPDisconnectScript</i>	171
Activity Locking	172

Interactive Web Views	174
What you need	174
The web page	174
The web server	175
TouchControl & TouchControl Server	176
USB-UIRT Broadcast (Windows server only)	177
Server Configuration Management.....	178
Server Configuration Recovery	179
Web Remotes	180
TouchControl Server JSON HTTP API (Windows server only)	182
Examples of JSON API requests	183
Response Format	183
getlocations	184
getactivities	184
getdevices.....	184
getbutton.....	185
getbuttons	185
executebutton	185
Sample Code	185
Zero Config	187
Screen Grabber	188
Viewing Grabber Output.....	189
Grabber Protection (Windows server only)	190
Troubleshooting the Grabber.....	190
EventGhost (Windows server only).....	192
EventGhost Feedback.....	192
Wake-on-LAN.....	195
Access From the Internet	196

[Custom Slider Images 197](#)
[Global Caché "Smooth Continuous IR Commands" 198](#)

[TouchControl Server Setup \(Windows\)](#)

1. Download TouchControl Server from the [Download](#) page. Either choose to run the installer when you download, or save the installer to your computer and then run.
2. Run TouchControl Admin. After installing TouchControl Server, you will find a TouchControl folder in your Start menu. Within that folder you will find links to TouchControl Server and the TouchControl Admin program. Before running TouchControl Server the first time, you must run TouchControl Admin to enable TouchControl to communicate on your network, and to enable event logging.

IMPORTANT: When running TouchControl Admin, even if you are logged on in Windows with the Administrator account, you will need to right-click the executable and select “Run as administrator” to give it the needed permissions.

When you run TouchControl Admin, you will be presented with a dialog box showing your Windows user ID and the communications port that TouchControl will use to communicate.

- a. The Windows user ID must be the user under which TouchControl Server will be run on your computer. This should only be changed if you are not running TouchControl Admin under the same user that you will use to run TouchControl Server.
- b. The default port is 8822, and should only be changed if you know other software on your computer is using port 8822. You will also need to update the port configured in the iPhone app if changed. NOTE: If you have multiple TouchControl Servers, and you wish to access them from the Internet, you should assign each server a unique port number so that you can properly route traffic to each server from your network router.

All activity buttons and mouse/keyboard functionality use an additional port on your server for communication with your iOS device. The port chosen for this purpose will be the next higher open port on your system - i.e. if you choose port 8822 as the primary TouchControl port, then port 8823 will be used for all button clicks and mouse/keyboard control, unless that port is taken, in which case port 8824 will be tried, then 8825, etc., until an open port is located.

- c. The dialog will also show the IP address (or addresses) of your computer. Please

note what your IP address is, as it will be needed when running the iPhone app. If needed you can run the TouchControl Admin program again later to view your IP address. NOTE: TouchControl Admin may show multiple IP addresses for your computer. Make sure you select an IP address that is accessible by other devices on your network.

- d. Clicking the "Execute" button will perform the necessary administration tasks to enable TouchControl communication. When complete, you should be presented with "Success" popup message.
 - e. PLEASE NOTE: Running TouchControl Admin requires administrator access on your computer to perform the necessary actions to configure the network port. If you are not logged on as the system administrator, you will be asked by Windows to allow TouchControl Admin to elevate its permissions to administrator level. The TouchControl Server software does not require administrator permissions to run under Windows. On Windows Server/Windows Home Server, you must be logged on as administrator when running TouchControl Server for it to function properly.
3. Configure your firewall (if needed). TouchControl Admin, will also prompt you to allow it to add Windows Firewall rules to allow communication with TouchControl Server. If allowed, TouchControl Admin will attempt to add a rule allowing a range of ports to access the TouchControl Server program on your computer. The range of ports depends on the port you choose as the primary HTTP port (in step 2 above). Any firewalls that you have installed and active on your computer MUST allow communication over the TouchControl Server ports that were configured using TouchControl Admin in step 2 (default is port 8822 and the next higher port number as explained above). Please consult your firewall configuration instructions to open the needed ports for TCP traffic. If TouchControl Admin is unable to create the Windows Firewall rules for any reason, you will be notified of the issue, and asked to manually add the Firewall rules for the specified ports. If needed, [here are step-by-step instructions for adding a port to Windows firewall](#).
 4. Run TouchControl Server. The first time you run TouchControl Server, you will be asked if you wish to use ZeroConfig networking to allow TouchControl on your iOS device(s) to automatically find and configure TouchControl server. If you answer yes, you will then be asked to provide a name for this server. This name is used during ZeroConfig setup in the TouchControl app on your device, and should be a short, "friendly" name that clearly describes this server (such as "Laptop", or "Home Server", etc.).
 - a. To enable ZeroConfig networking, you must have Apple's "Bonjour for Windows"

installed and active on your TouchControl Server. If it is not found, TouchControl Server will alert you and allow you to either quit TouchControl Server and install Bonjour, or disable ZeroConfig networking via server settings. Bonjour for Windows may be downloaded from the Apple web site here:

<https://support.apple.com/kb/DL999>

- b. If you prefer not to use ZeroConfig networking, once you disable the feature in TouchControl Server, you will not be prompted again (until you re-enable the feature), and all networking updates must be made manually in TouchControl on your device.
- c. Once you enter and save the server's name, you will be presented with the Settings screen. All settings on this screen will contain their default values as follows:

Server Settings:

- **Server Port:** The communications port over which the iPhone app will communicate with your computer. Only change this if needed as described above.
- **Data Directory:** The default location for TouchControl data is the "...\Users\\Documents\TouchControl" directory. You may change this if desired.
- **Button Packs:** Image files for buttons are stored in "button packs" (ZIP files). This list displays the button packs that you have selected to use in TouchControl. You may use the Add and Remove buttons to manage this list. Please see the [Backgrounds and Button Packs](#) topic for more information.

Preferences:

- **Enable IRCommand2 Interface:** Instructs TouchControl Server to import the devices and buttons configured in the IRCommand2 software and makes them available to add to your remote control screens. If you do not have IRCommand2 installed, this option will be disabled. This feature is available as an upgrade via an in-app purchase in the iPhone app. Select Settings (gear icon) from the iPhone app nav bar and tap on the "Upgrade" option to purchase this feature. If this option is selected, IRCommand2 will start automatically with TouchControl Server, if it is not already running.
- **Animate layout designer panel:** By default, when showing or hiding the

activity layout designer panel, TouchControl Server will grow and shrink as needed in an animated fashion. De-select this option to disable the animation.

- **Block PC sleep/hibernate:** If enabled, TouchControl Server will keep your PC awake and available to respond to requests from your iOS device as long as TouchControl Server is running. NOTE: You may also use the wake-on-LAN feature to wake up your TouchControl Server system from your iOS device when needed.
- **Check for new version at startup:** If enabled (default), TouchControl Server will perform a check at startup to see if a new version of the software is available from the web site. If a new version is available, you will be prompted to automatically navigate to the download page on this site via your default browser to download the new version. IMPORTANT: If you'd rather not automatically perform the check for a new version at startup (for example if you run TouchControl Server in a lights-out environment and do not wish to be prompted at startup in the event a new version is available), you should disable this option. In this case you should periodically perform the version check manually, either by using the Help > Check for new version menu option, or by visiting this site.
- **Enable AutoHotKey QuickEdit:** This option will enable the ability to quickly edit all AutoHotKey button commands in a single, scrolling view when there are multiple AutoHotKey buttons available in a given device at design-time.
- **Enable ZeroConfig network setup:** If you disabled ZeroConfig during initial server setup, you can turn it on here, or turn it off if you wish.
- **Minimize to System Tray:** By default, TouchControl Server will minimize to the Windows task bar when minimizing the application (standard Windows functionality). If this option is enabled, the application will minimize to the system tray instead, and will disappear from the task bar when minimized.
- **Protect WebRemote:** This option will allow you to configure a password that must be used to access the WebRemote function from a browser.
- **Scale layouts in designer:** By default, when an activity loads into the layout designer, if it is too large to fit on your PC's screen, it will scale the layout (background and all button images) to a size that fits on the currently used monitor. With excessive scaling, it can sometimes become difficult to acquire precisely desired positioning when the

activity is rendered in full scale on an iPad. If this is the case, you can turn this setting off, and the layout designer will render at full size on your PC. Note that this could require that you drag the TC Server window around on your desktop to access all of the buttons that might be at the far edges of the layout, depending on the size of the layout vs. the size of your PC screen.

- Start screen grabber with server: This option will automatically start the TouchControl Screen Grabber when the server starts, positioning the grabber window in the same location it was when the grabber was last used.
- Toggle group edit: When editing group buttons, you can press and hold the “G” key to temporarily hide all buttons inside all groups for easier access to and manipulation of the groups themselves. Releasing the “G” key re-displays the contents of the groups. Enabling this toggle feature allows you to press and release the “G” key to turn on group edit mode, and then press and release the “G” key to turn it back off (effectively toggling group edit mode) rather than having to hold down the “G” key to edit the groups.
- Un-dock designer window: This option presents the designer window as a separate window independent of the main TouchControl Server window, allowing you to position it wherever you like, rather than always being “docked” to the right side of the server window.
- USB-UIRT broadcast: This option turns on a feature that will automatically re-broadcast any IR commands received by a connected USB-UIRT device back out onto your network via the UDP protocol, where TouchControl running on your iOS device(s) can listen for and respond to them. See [this topic](#) for more information.
- Use external script editor in designer: In the Script Manager interface within TouchControl Server, you have the ability to specify a program on your computer to use to edit the external script files included in the Script Manager, rather than using the built-in script editing text field. When an external editor is specified, enabling this option will also allow using that same program to edit script embedded within buttons in the main button config interface.
- Show location color selector: This option enables a color selector button next to the location drop-down control on the server’s main panel. With this enabled, the color selected for a given location will be used to color the location and activity buttons on the home screen in

the TouchControl iOS app when the “Colors” theme is chosen in the “Theme” settings within the app. The default color is black.

- Designer grid size: When the “snap to grid” feature of the designer panel is enabled, this option will allow you to set the size of the grid in pixels.

Click "Save" when done.

[TouchControl Server Setup \(macOS\)](#)

1. [Install TouchControl Server from the Mac App Store](#)
2. Run TouchControl Server. The first time you run TouchControl Server, you may be prompted to allow the app to access the local network. If prompted, answer Yes to give TouchControl Server the access needed to communicate with your iOS device(s). You will then be asked if you wish to use ZeroConfig networking to allow TouchControl on your iOS device(s) to automatically find and configure TouchControl server. If you answer Yes, you will then be asked to provide a name for this server. This name is used during ZeroConfig setup in the TouchControl app on your device, and should be a short, "friendly" name that clearly describes this server (such as "Laptop", or "Home Server", etc.).
 - a. If you prefer not to use ZeroConfig networking, once you disable the feature in TouchControl Server, you will not be prompted again (until you re-enable the feature), and all networking updates must be made manually in TouchControl on your device.
 - b. Once you enter and save the server's name, you will be presented with the Settings screen. All settings on this screen will contain their default values as follows:

Server Settings:

- **Server Port:** The communications port over which the iPhone app will communicate with your computer. Only change this if another process on your Mac happens to be using the same port.
- **Data Directory:** The directory that contains all of your TouchControl configuration data. You may create multiple data directories and switch between them using the drop-down list.

IMPORTANT: Due to the sandboxing security features of MacOS apps, the initial default location for TouchControl data is located within the app's sandbox container. Since Apple does not permit apps to save user data to the sandbox container, you must change this to a location outside the app's sandbox container path, such as under your Documents directory. Sandboxing will also not allow TouchControl Server to create this directory for you, so you will need to first create an empty directory (e.g. named "TouchControl" or whatever name you wish) in the location of your choosing, and then return to TouchControl Server and use the

“Browse” button to select that directory before you will be allowed to save settings and continue.

- Hide server at startup: This option will automatically hide the server window immediately after startup. The server icon will remain in the dock to restore the window.
- Launch Web Remote at startup: This option will automatically launch the Web Remote interface in your default Web browser on your Mac whenever the TouchControl Server app is started.
- Use external script editor in designer: This feature allows you to specify an external editor installed on your Mac to use when editing script within TouchControl Server, rather than using the built-in script editing text fields. Enter a program installed on your Mac in the “Editor” field to specify the external editor to use. For example, if you have the TextWrangler app installed on your Mac, simply enter “TextWrangler” in the “Editor” field.
- Designer grid size: When the “snap to grid” feature of the designer panel is enabled, this option will allow you to set the size of the grid in pixels that buttons will snap to when moved.
- Show location color selector: This option enables a color selector button next to the location drop-down control on the server’s main tab. With this enabled, the color selected for a given location will be used to color the location and activity buttons on the home screen in the TouchControl iOS app when the “Colors” theme is chosen in the “Theme” settings within the app. The default color is black.

Click "Save" when done.

[iOS Device Setup](#)

Note the minimum required iOS version to run TouchControl is iOS 9.3.

Please follow these steps to install and activate TouchControl on your iOS device.

1. Download TouchControl from the App Store and install on your device.
2. Make sure you have taken a look at the [Troubleshooting](#) page to make sure your server is accessible from your device, especially the link to manually open ports through Windows Firewall. Although Windows may have asked if you would like to allow access to TouchControl Server, you may still need to open the ports on your server manually, so probably worth walking through those procedures. At a minimum you need to make sure that you can access the test URL at <http://192.168.xx.xx:8822/touchcontrol/getserverinfo> from another device on your network (preferably via Safari on your iOS device) and see the server version display.
3. When you first start TouchControl on your iOS device, it will display a welcome screen with some general information about the app. Tap the "Continue" button to start the installation and configuration process.
4. If you have enabled ZeroConfig networking (discussed below) during TouchControl Server setup, you may see alerts that new servers were found and added to your configuration. After that, you will be asked if you would like to connect to a TouchControl Server.
 - a. Select "Yes" if you already have a TouchControl Server established and running and would like to connect to it and download your configuration. If you select "Yes" here, proceed to step 5.
 - b. Select "No" if you would like to put TouchControl into "Default Configuration Mode" on your iOS device. Default mode does NOT connect to your server, and includes a small number of sample activities that you can use without connecting to a server. Please tap the "?" icon in the navigation bar while in Default Mode to learn more about these sample activities, and for information on connecting to your server when you are ready to do so. If you select "No" here, your setup is complete, so do not continue to step 5.
5. If you have enabled [ZeroConfig networking](#) during TouchControl Server setup ([see here](#)):

- a. TouchControl will automatically detect any TouchControl Servers that are available on your network and will automatically configure the needed settings (IP address and port number) to connect and download your configuration. If TouchControl finds multiple servers, you will be presented with a dialog which will allow you to select the server that you would like to use for the initial setup on your device.
 - b. Note: If your TouchControl Server has multiple network adapters (i.e. Ethernet and wireless), ZeroConfig will find both networks and configure them as multiple servers (with the same name) in the client configuration. This is not an error and will allow you to contact your server when it is connected to your network in either wired or wireless modes.
 - c. Ensure that "Demo mode" under Personal Settings is disabled (not checked).
 - d. Select a server and tap the "Save Settings" button (iPhone) or the check mark icon (iPad) to continue.
6. If you have not enabled [ZeroConfig networking](#) during TouchControl Server setup:
- a. On the initial settings screen, tap "TouchControl Server" under Network Settings.
 - b. Enter a server name - this can be any "friendly" name that you'd like to give your server, such as "Home Server", "Laptop", "HTPC", or anything you want to uniquely identify the server, as you may have more than one TouchControl Server in the future.
 - c. Enter your server's IP address in the "Wi-Fi Server IP" field. The available IP addresses were listed at the bottom of the TouchControl Admin dialog, and there may have been multiple shown (e.g. wired and wireless), so you just need to make sure you use an IP address that is accessible by other devices on your network. Assuming you were able to get the test URL (above) to work, then use that IP address.
 - d. In the "Server Port" field, enter the port that you used in TouchControl Admin. The default is 8822 and is pre-populated for you, so only change it if you changed it in TouchControl Admin.
 - e. Tap "Save Settings" to return to the initial setup screen.

- f. Ensure that "Demo mode" under Personal Settings is disabled (not checked).
 - g. Tap "Save Settings" (iPhone) or the check mark icon (iPad) to kick off the initial search for your server.
7. You should see "Looking for Server..." and then "Server found! Loading new configuration..." on your iOS device screen.
8. You should then see TouchControl loading button packs and background archives.
9. You should then see the default TouchControl home screen with the included sample locations and activities, or any custom activities you have added on the server, displayed in a list of buttons.

You can then tap "Settings" in the nav bar on the main activities screen to browse the list of settings that are available to you on the iOS device.

Use the refresh button (circular arrow) on the right side of the app's navigation bar to refresh the configuration from the server to the device any time you have made changes on the server. This will download the updated configuration, and any changes you've made to images, HTML, script or other files located in the server's app directory. If no configuration changes have been made, or files updated, the previously-built configuration will be downloaded to the device.

Note: If you wish to force the server to rebuild the configuration, press and hold on the app's refresh button for two seconds. The server will rebuild the configuration and then send the configuration, as well as the background images archive (in the images/backgrounds.zip file) to your device.

[Locations and Activities](#)

[Locations](#)

The first task you should perform in TouchControl Server is to add a location. A location provides a grouping for activities within TouchControl. You may add as many locations as you wish. Add a location - such as "Home Theater", or "Family Room", or "Bedroom", etc. using the File > New Location menu option.

When the "Show location color selector" option is enabled in server settings, a color selector button will appear next to the location drop-down when a location is selected. The color selected here will be used to color the location and activity buttons on the home screen in the

TouchControl iOS app when the “Colors” theme is chosen in the “Theme” settings within the app. The default color is black.

Activities

Next you must add at least one activity. Activities consist of one or more devices that are logically grouped together to allow you to perform a task - such as "Watch TV", or "Watch Movie", or "Lighting". Add a new activity using the File > New Activity menu option.

After you have created at least one location and one activity, select the desired location from the Location drop down list, and then select the desired activity from the Activity drop down list. If this is the first time you have selected this unique location/activity pair, you will be asked if you would like to add this pairing to your configuration - click Yes

Windows Server:

Note: All activity names are available to add to any location. An activity name may also be added to more than one location. The activity list is divided into three sections:

Visible – those activities that exist in the currently selected location, and which will be visible in the app on the client device.

Not visible – activities that exist in the currently selected location, but are not visible in the app on the client device (see the "Visible" setting below).

Not included – activity names that have not been added to the currently selected location. Activities in the "Not included" list may be added to the currently selected location by simply selecting the activity from the list and following the prompts. When adding a new activity to your configuration, you will be prompted to optionally create that activity from another existing activity, effectively allowing you to use any existing activity as a template for new activities. If you expect to create many activities that all share the same basic design, you could create a "template" activity with only the common design elements and then use that activity as the template when creating new activities, saving potentially extensive design/layout effort.

Mac Server:

Within the activity popup list, activities that are visible are denoted with the “✓” symbol, and activities that are not visible on your device are denoted with the “X” symbol.

Activity Configuration: After selecting an activity, on the Windows server a "Configure →" button will appear to the right of the activity name. Clicking this will display a small configuration panel for that activity with the below options. On the Mac server, these options will automatically appear when an activity is selected.

iPhone - designates this activity as targeted for the iPhone/iPod (default). When this option is selected, the layout designer panel will initially size to 320x420, optimized for the iPhone/iPod. Layouts designated as iPhone/iPod will always zoom to fill the iPad screen.

iPad - designates this activity as targeted for the iPad. When this option is selected, the layout designer panel will initially size to 768x960, optimized for the iPad (portrait). Layouts designated as iPad will not be available in the iPhone/iPod app.

Watch – designates this activity as targeted for the Apple Watch. When this option is selected, the “Style” drop-down will be enabled, allowing you to select the layout of the activity on the watch. See [Apple Watch](#) for more information.

Scale to fit – this option will force the layout to automatically scale to the size of the device screen in all orientations (note iPhone/iPod only supports portrait orientation).

You can also use this option make activities designed for smaller devices render at full screen on larger devices, or activities designed for larger devices render at full screen on smaller devices. Note that the "Stretch to full" option in the iOS app is also available, and will also provide a similar experience for smaller layouts on larger screens. The difference between "Stretch to full" and "Scale to fit" is that "Stretch to full" will only scale smaller activities up to larger screen sizes (as that feature was added specifically as a stop-gap measure for users moving from a smaller device to a larger device, until the activities could be re-designed), and thus stretches/scales all activities in that manner. "Scale to fit" on the other hand, can scale smaller activities to larger screens, but can also scale larger activities to smaller screens, and is on an activity-by-activity basis, as determined by you when selecting this option in the designer.

Zoom to Width – this option forces activities to scale to the width of the device they are running on, and proportionately scale the height as well, which effectively "zooms" the activity to the size of the device (based on device width). Your activity background image will always fill the entire screen, and the activity may scroll vertically if the zoom process results in an activity that is taller than the device's screen. This is useful when using the same activities on multiple devices with different screen dimensions.

Visible – if this option is de-selected, the associated activity will not show up in the

app's activity list, however, the activity will still exist, allowing you to link to the activity from other activities, while not taking up room on the main activity list/screen. This lets you create drill-down menus of activities, if desired.

When creating an activity strictly to use as a template, you should mark the activity as not visible, so that the partial activity does not display on your device. (Remember to "turn on" the visible setting of your newly created activity if you wish, as new activities will inherit all configuration properties of any template used to create them.)

Safe Area – if this option is selected, the associated activity will automatically adjust the top and bottom margins of your activity to layout within the "safe area" on full-screen iPhone and iPad devices. This allows you to design your activity layouts with buttons at the top and bottom edges, but Touch control will automatically adjust the layout so that the top is below the front-facing camera assembly (the "notch") and the bottom is above the "home bar". This will also help avoid the rounded corners of these devices. The activity background image will continue to extend to the screen's top and bottom edges for a finished look. If you'd rather take advantage of this space on the device screen, then leave the "Safe Area" option unselected, which is the default.

Shadows – To enable button shadows for specific activities, but not for all of them, select the Shadows setting in the server's activity configuration, and turn OFF the Button Shadows options in TouchControl settings on your device. Leave that option ON in settings on your device if you wish to use button shadows for all buttons in all activities regardless of this setting in the server designer.

Haptics – To enable button haptics for specific activities, but not for all of them, select the Haptics setting in the server's activity configuration and turn OFF the Sound & Vibration Haptics option in TouchControl settings on your device. Leave that option ON in settings on your device if you wish to use button haptics for all buttons in all activities regardless of this setting in the server designer.

Mouse/Keyboard – (Windows server only) displays a "Mouse" button on the navigation bar on the iPhone when using this activity. Pressing the "Mouse" button will display the default TouchControl mouse pad and keyboard interface on the iPhone. This interface allows you to control the mouse cursor on your computer, and send keystrokes and keyboard shortcuts/commands to applications/windows on your computer as well.

NOTE: If using full-screen activities on the iPhone, the Mouse button will not be available, as the navigation bar will be hidden.

By default, the mousepad/keyboard will control the TouchControl Server where your configuration was loaded from, but it can also be configured to control a different TouchControl Server PC. To point the mousepad/keyboard to an alternate server, set the following global variable in script prior to opening the built-in mousepad:

```
_global.mouseKeyboardServer = '192.168.xxx.xxx:8822';
```

Use the alternate server's IP address and server port here (the default server port is 8822). See the [Scripting](#) and [Local/global variables](#) topics for more information on setting global variables via script.

Devices and Buttons

Devices

Activities are groupings of devices, so you must add one or more devices that you wish to control to any given activity. "Devices" in TouchControl Server are really groupings of buttons that logically work together in some fashion. Devices can mirror physical equipment, like a TV, or HD DVR, or DVD Player, or a lighting system, or power window shades, etc., but can really be any set of buttons that perform any function. When controlling applications on a computer, for example, if you have an XM Radio Online player on your computer that you'd like to control, a device could be named "XM Radio", or if you would like to just group together some keyboard shortcuts, a device might be named "Keyboard".

Add one or more devices to your configuration by clicking the Add button under the *Available Devices* list. If you selected a location/activity pair earlier, you may then add the newly created device to that activity by highlighting the device in the Available Devices list and clicking the → button to add it to the *Devices In This Activity* list. Only buttons from devices added to an activity can be used on a remote layout screen.

NOTE: Windows users, if you have selected to use the IRCommand2 interface on the settings page, all of your IRCommand2 devices will be added automatically for you, and will be designated by displaying "[IRC2]" before the device name.

Windows users, if you create a "device" in TouchControl Server which will mirror a real physical device (such as a DVD player), you may also set a default physical "host" device for that logical device. Right-click on any device in the list and click "Default host..." from the popup menu and you will be presented with a panel allowing you to select the physical "host" device that you will likely use for at least the majority of buttons within that logical device. The physical "host" devices are configured using the Interface Manager feature of TouchControl Server, so [please see this topic](#) for more information on adding physical host devices. Once a default host is selected, that will be the default host displayed when configuring any buttons contained within that device. Please note this is only a default to help speed up the button configuration task. You may override this at any time for any given button during button configuration.

You may also change the host defined for buttons in a device all at once by right-clicking on a device in the "Available devices" list and selecting "Replace host...". This will provide a list of currently configured hosts, and after selecting one, TouchControl will replace the host on buttons within the device with the newly selected host. If the device contains buttons of varying types, this will only replace the host on relevant buttons within the device. That is, if you select an HTTP Request host, for example, it will only update HTTP Request buttons in the device with that new host, leaving all other button types unchanged. This option is useful if you

replace one physical device with another device, but wish to simply "re-target" the buttons from the old device to point to the new device.

NOTE: (Windows only) To easily locate a device in the Available Devices list, simply select any entry in the list and start typing any part of the desired device's name. This will immediately filter the list to only the devices containing the typed string (which can appear at any location in the found devices). To filter the list to only devices starting with a given character or string, simply press and hold the first character you wish to search for, and the list will immediately be filtered to display devices whose name starts with the entered characters. Once you've found the device you are looking for, just press "Enter" to open the device and display its buttons, or press Ctrl-Enter to add the device to the currently open activity. On Mac server, typing a device name will jump to that device in the list.

(Mac only) If you right-click on a device in the Available Devices list, and then select the "Property..." option, a dialog will appear that will allow you to add a property to, or remove a property from, all buttons in that device at once.

Buttons

To control a device, you must add buttons which correspond to the buttons on the device's original remote, or in the case of PC control, that correspond to functions you'd like to perform on the system.

* = Supported on Apple Watch

Primary Buttons

IR buttons* (Windows server only) will transmit an IR signal (via USB-UIRT only) to the device when the button is clicked.

Command buttons* will execute programs and processes on your computer.

AutoHotKey buttons* (Windows server only) will execute AutoHotKey scripts on your computer. AutoHotKey is distributed and installed with TouchControl Server (in-app upgrade required).

EventTrigger buttons* will trigger events (i.e. send data) to a variety of different servers or software applications, such as networked A/V equipment, EventGhost, automation controllers, or any other device or server that can communicate via a socket over TCP/IP.

Global Caché buttons* will send IR, serial, or relay (contact closure) commands to your Global Caché iTach or GC-100 devices (in-app upgrade required).

HTTP Request buttons* will send HTTP requests (GET or POST) to web servers/services (for example the J.River Media Server or UPNP devices).

iRTrans buttons* will send commands to your iRTrans device (in-app upgrade required).

Composite Buttons

[Macro buttons](#)* allow you to create macros consisting of multiple other button types which are then executed in order when the macro button is pressed, including pauses between buttons as needed.

[Slider Buttons](#) let you add an iOS slider control to your layouts, and execute commands when sliding left, sliding right, or specify a specific command to execute at each stop on the slider bar.

[Spinner Buttons](#) let you add an iOS spinner, or "picker" control, to your layouts (similar to the iOS date picker), and execute commands when spinning the wheel. Spinners can also use a table layout, or a grid layout.

[Gesture Pad buttons](#) allow you to create buttons that respond to single- and multi-touch gestures (swipes, taps, and rotations).

Auxiliary Buttons

[Link to Activity buttons](#)* allow you to link from one activity screen to another.

[Labels](#)* allow you to display text and or images on your remote layouts.

[Web Views](#) allow you to embed a web view, or "window", within a remote layout to view web pages or HTML content that you supply.

[URL buttons](#) will launch the web browser on the iOS device to view a configured web page, or can be used to launch other iOS apps that allow it.

[Feedback Client buttons](#) allow TouchControl to open a connection to a specified device on your network and monitor that connection for feedback, and process any data received with custom script.

[Feedback Listener buttons](#) allow TouchControl to open a UDP port on your device and listen for broadcast messages from other devices on your network, and process any messages received with custom script.

[Script buttons](#)* allow TouchControl to run a block of script, without connecting to a remote device or sending any command. Use this button type when you simply need to run script locally within TouchControl on your iOS device.

[Group buttons](#) allow dropping other buttons on top and positioning on your layouts as a single entity. Groups may also be tied to a template, allowing you to reuse groups in multiple locations, but maintain them globally.

[Text Field buttons](#) render as a multi-line, scrolling text box allowing you to enter/interact with their contained text via the device keyboard, or scripting.

Repeating Buttons

If you'd like any of the primary button types to repeat the command when you hold down on the button on the screen, check the "Repeat" option to the right of the button type on this panel. This is a per-button setting, as some buttons should not repeat their commands. Hold-to-repeat buttons will perform their function immediately when pressed - and then if held for at least 1/2 second, will start repeating the function until the button is released. Set the repeat

interval from .1 sec. (the default) to 600 sec. (in .01 sec. increments) as desired to adjust the speed of the repeating commands. (Note the up/down control on the user interface allows changing the interval by .05, but you may manually enter any value with increments of .01.) Using the repeat interval is useful if you experience lag from your IR transmitter, AutoHotKey, EventGhost, etc. (i.e. commands continue to repeat after you have released the button in TouchControl). Altering the interval at which the buttons repeat allows you to more closely match the actual duration of the command executed by each repeat of the button. Note that it may take some trial and error to get the button repeat to match the actual command duration.

Timer Buttons

Any of the primary button types (IR, Command, AutoHotKey, EventTrigger, Global Caché, or HTTP Request) may also be executed via a timer. After selecting the "Repeat" option (above), you then may select the "Timer" option, which will allow you to set the button to execute either once after the timer expires, or each time the timer fires at a set interval. When using the "Timer" feature, the repeat interval value becomes the timer's interval, and as above, can be set from .1 sec. to 600 sec., in .1 sec. increments. Timer buttons are started by tapping the button on the iOS device screen, or by executing the button using the "[#]" script return string element (see [Scripting](#) for more info). Timer buttons can be stopped by 1) tapping a currently running timer button again, 2) by once again executing the button from script using "[#]", or 3) by exiting the activity containing the timer button. Linking forward to another activity will not terminate timer buttons, as the activity containing the timer still exists in that instance. Timer buttons may also be set as the AutoExec On Load button for an activity, to ensure the timer starts as soon as the activity loads, if desired.

Feedback Buttons

Command, AutoHotKey, EventTrigger, Global Caché, and HTTP Request buttons also allow you to specify if you'd like to receive feedback from those button executions. Selecting this option will force TouchControl on your device to wait after each button click for feedback from the device or program that you are controlling. Make sure that the resulting action from the button click actually returns feedback from the action, or TouchControl will timeout waiting for the feedback to arrive.

IMPORTANT: If you are controlling a device or server that always returns feedback from each button click (such as a Denon receiver, for example), you should always select the "Feedback" option for any buttons used with that device/server. Using feedback buttons ensures that the feedback from the device is always emptied from the queue after each button click and does not accumulate. If you are not interested in processing the feedback from any button, just don't provide feedback script to process it, and it will effectively be ignored.

Alternately, when controlling EventGhost, you specify which button clicks should return

feedback using the Feedback action within the TouchControl EventGhost plugin. In that case you should only use feedback buttons for commands that you have defined to return feedback, because looking for feedback when none is generated will cause the app to pause and timeout while it waits for feedback, drastically degrading performance.

Adding Buttons

To add a button to your configuration, highlight a device in the "Available Devices" list and click the "Add Button" button.

On the "New Button" panel, first supply a name for the new button. The name given a button will also be used by default for the text/caption for the button on your remote layout screen.

Next, select the button type from the many available types. [Primary Buttons](#) are the main buttons that do the work in TouchControl. These buttons are configured (later) with the actual commands that will be sent to the remote devices. [Composite Buttons](#) are actually made up of one or more "primary" buttons, via various configuration methods. So, commands that you wish to send using any of the composite button types must first be configured via a primary button. [Auxiliary Buttons](#) allow you to perform various "auxiliary" tasks on your remote layouts, including running script, displaying text as a label, linking to other activities, launching URLs or other iOS apps, or displaying web pages within a window on your remote screen. You may also specify whether to automatically display the built-in TouchControl mouse pad or keyboard screen after you execute a given button command. This is simply a time saver primarily for HTPC users, in case there are commands that you execute that you know you will always want to navigate the cursor or send text immediately following the command.

By default, a button will display its name as the button's text on your device's screen (for those button types that support text labels). This may prove a challenge if you would like to embed any type of formatting in the button's text on the screen, such as line feeds, etc. (as button names are also used to reference buttons in scripting). Therefore, you can set alternate text for any button by selecting one of the "Alternate Button Text" options on the new/edit button panel. The "Text" option allows you to enter simple text with embedded escape characters (such as \n for line feed, etc.). The "HTML" option allows you to enter any desired HTML that you'd like to display on the button, including tables, divs, inline styles, etc. The "HTML" option gives you nearly complete control over the button's text content, including loading images to display on top of the button's background image. Even the button images that you use as normal button backgrounds can be loaded via HTML onto any button to mix text and images in unique and interesting ways. To load one of your button pack images onto a button via HTML, simply use the tag as follows:

```
<img src='images/buttonpackname/imagefilename.png'/>
```

(note the buttonpackname and imagefilename are case sensitive).

TouchControl will load this image directly from TouchControl's on-device cache, so no external (off-device) loading is necessary (although that's possible too by specifying an external location for the image source).

When setting alternate text for a button, you may also specify one of the button's defined properties. To specify a property value as alternate text, enter `_property.myPropertyName` in the alternate text field. When the button renders on the iOS device screen, the specified property value will display as the button's text or [TouchTip](#). This allows you to have the same button displayed multiple times on the same layout with different text.

Configuring Buttons

After you've added one or more buttons, highlight a button and click the Set Data button (Windows) or Configure button (Mac) - or double-click the button in the list, to configure the button's functionality. You may add as many buttons as you wish before defining any of their functions using Set Data.

If you have selected to use the IRCommand2 interface (Windows only) on the settings page, all of your IRCommand2 buttons will be available automatically for you when you select an IRCommand2 device in the device list, and will display "IRC2" as the button type. The function performed by IRCommand2 buttons is defined within the IRCommand2 interface.

On Windows server, right-clicking on a button in the buttons list will allow you to edit the given button, which will allow you to re-define the core functionality of the button (e.g. change from an IR button to a command button, set repeat, timer, and/or feedback options, etc.). On Mac server, the button configuration is always available via the "Button Settings" tab at the top of the button config window for any button. The right-click menu will also give you the option to duplicate (copy) the selected button, and specify which device the new (copied) button should reside in, and provide a new button name, if desired.

NOTE: To easily locate buttons from any device in your configuration, simply select any entry in the "Available Devices" or the "Buttons for selected device" lists and either right-click and select "Find Button...", or just press Ctrl+F. This will open the Find button panel. Simply begin typing any part of a button name in the "Button name" field, and the result list will automatically filter to any buttons whose name contains the entered string. Highlighting and selecting a button in the results list (use arrow buttons or click with mouse to highlight, then click "OK" or double-click or press enter to select) will automatically select that button's device in the devices list, and automatically display and select the button in the buttons list.

2-Stage Buttons

An additional method of displaying buttons in your layouts is by creating a "2-stage" button. 2-

stage buttons allow you to place a button on a layout, and when that button is tapped, a “real” button appears over your layout which you can tap to execute the intended command. This could be useful if you have a layout with many buttons on an iPhone or iPod, and don’t have room for all of the buttons’ full text. A small image or hot-spot could be used for each button, and then when tapped, a “clickable” button would appear with the full button text. 2-stage buttons also display a dismiss icon (red “X”) that allows you to dismiss the button without actually executing it. This could also be useful for a button that has a long running macro (e.g. starting up your theater, adjusting lights, opening/closing blinds, etc.), and you sometimes accidentally tap it at the wrong time. Using a 2-stage button, you would in essence have the chance to confirm or dismiss the actual execution of the button before possibly executing a destructive command or macro.

2-stage buttons are implemented via the gesture pad, so add a gesture pad button to any layout, and when configuring the pad, select the “Tap” gesture, and then select the “2-Stage” option. Select the “action” for the tap gesture just as you would for any other gesture pad. The button selected as the tap gesture “action” will be the button used for the 2-stage command.

Note: If you have not purchased the IRCommand2, AutoHotKey, Global Caché or iRTrans upgrades in the iOS app, you can add those buttons in TouchControl Server, but they will not be accessible on your remote screens on the device. You can, however, add, configure, and test all of these buttons from within TouchControl Server to ensure they work properly before purchasing the upgrades on the device.

Now you're ready to start [Designing Layouts](#).

IR Buttons (Windows server only)

Learning IR Codes (for use with USB-UIRT only)

Once you have added one or more IR buttons, you are ready to teach TouchControl the IR signals (codes) that it will transmit to your equipment. Make sure you have the original remote available for the device that you'll be controlling, then highlight a button in the "*Buttons for selected device*" list and click the **Set Data** button, which will display the IR code learning screen. Aim the device remote's IR transmitter at the USB-UIRT device's IR receiver, click the **Start Learning** button on the TouchControl screen, and press and hold the corresponding button on the remote until you see the IR code displayed in the window. If the learning task fails, a message will alert you to this, and you can repeat these steps to try again. Until you find the right orientation and distance for learning the signals, it may take more than one try before successfully learning the codes. After the code has been successfully learned, click the **Save** button to save the code to the TouchControl configuration. If you want to immediately test the code, point the USB-UIRT at your equipment, and press the **Test** button. The device should

perform the desired function. If it doesn't, you may need to re-learn the code, as it may not have detected the entire code stream.

If you have selected to use the ***IRCommand2*** interface on the settings page, your ***IRCommand2*** buttons will automatically use the IR codes (or any other functions) defined in ***IRCommand2***, *so no re-learning needed!*

IR Transmit Repeat Factor

The IR transmit repeat factor controls the transmission of the IR signal from the USB-UIRT device. Some devices require multiple IR signal bursts to ensure the signal is received successfully, while other devices require that the IR signal is transmitted only once for the command to function as expected. Allowing the repeat factor to be set for each button gives you ultimate control over how the signal is transmitted. For example, if you wish for a signal to be transmitted multiple times - to increase/decrease volume more than one notch at a time, or to move to the top or bottom of a list of menu items, for example - you can increase the repeat factor to accomplish this. The default factor is 2, which should allow the majority of IR signals to function as expected.

Manual IR Codes (for use with USB-UIRT only)

As an alternative to learning codes from an existing remote, you may also manually insert IR codes by pasting them directly into the code learning window in TouchControl. The codes used by TouchControl are in the Pronto CCF format, which can be found in various places - one of the most popular being <http://www.remotecentral.com/>, which hosts thousands of remote codes for hundreds of different devices. Visit the "Files" section of that site, find the desired code for your device, and copy and paste it into the TouchControl window. This can be very handy if you have, for example, lost the original remote for a device.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See the [Advanced Scripting](#) topic for more information.

Command Buttons

Command buttons allow you to execute commands, processes, executable, batch files, etc., on your TouchControl Server PC, directly from your iOS device. To set the command to be executed by a command button, select the button in the list and click the "Set Data" button, which will open a panel that allows you to enter a PC command or program to be executed when clicking on that button on your remote layout. In addition to entering commands manually via the keyboard, you may also drag and drop files, shortcuts, URLs, text, etc. on to the command text box to automatically configure the button to launch that item. Use the

additional command arguments text box to enter any additional arguments that you'd like to pass to the command or program that is defined in the command box.

Feedback Script

If you enable the "Feedback" option for a command button, you can add JavaScript to the button configuration that will run whenever feedback is received from the command executed by this button. Note that feedback for Command buttons is only provided when running an executable (.exe) or a batch file (.bat) on your TouchControl Server PC. See the [Scripting](#) topic for more information.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See the [Advanced Scripting](#) topic for more information.

iOS Device Commands

You can use a command button to perform various functions to control the iOS devices themselves. For example, you can adjust the device's back light, enable or disable "Always On" mode (device sleep), etc.

Always On mode

These commands enable or disable the device's built-in sleep timer. This also directly updates the "Always On" setting within TouchControl's app settings. Use the following commands in the "Command" field:

{enable sleep}	- disable "Always On" mode
{disable sleep}	- enable "Always On" mode
{toggle sleep}	- toggle "Always On" mode

Each of these commands will also update the variable `_sleep` to the value "enabled" or "disabled", which can be accessed/used within any pre-, feedback, or post-script to determine the current sleep mode.

Full Screen mode

These commands allow you to put an individual activity into or out of full screen mode, independent of the "Full Screen Activities" option in TouchControl app settings:

{fullscreen:on}
{fullscreen:off}

Each of these commands will also update the variable `_sleep` to the value "enabled" or "disabled", which can be accessed/used within any pre-, feedback, or post-script to determine the current sleep mode.

Screen Brightness

These commands adjust the device's hardware back light. When the back light is fully dimmed using this feature, a single tap or swipe on the screen must first be performed to bring the device out of "full dim" mode and enable normal screen touches within the app. Use the following commands in the "Command" field:

{screen full bright}	- set brightness to 100%
{screen full dim} restore	- set brightness to 0%, and enable tap/swipe
{screen set bright:nn} set bright:80}	- set brightness to specified percent (e.g. {screen
{screen set bright+nn}	- increase brightness by specified percent
{screen set bright-nn}	- decrease brightness by specified percent

Each of these commands will also update the variable `_brightness` to the current screen brightness after the command has executed, which can be accessed/used within any pre-, feedback, or post-script to determine the current brightness level.

NOTE: This brightness feature is intended to be an app-specific setting. However, there is currently a bug in iOS that will prevent the device from returning to the global iOS brightness setting when exiting the app using the device's home button. Therefore, if you find that the screen brightness at the iOS home screen is incorrect, you can simply press the sleep (top) hardware button on the device, and then wake the device to restore the correct iOS brightness setting.

Activity Locking

These commands allow you to "lock" an activity optionally requiring a password to "unlock" the activity, allowing navigation away from the activity screen. Use the following commands in the "Command" field:

{screen lock:on:password}	- lock the activity, requiring the user to enter the supplied password to unlock
{screen lock:on}	- lock the activity, requiring you to programmatically unlock the activity
{screen lock:off}	- programmatically unlock the activity

Please see [this page](#) for more information on activity locking.

Screen Scrolling

You can also scroll the screen of a scrollable activity directly from Command buttons on your iOS remote layouts. Simply enter the following in the command field when configuring your buttons:

{scroll page right}	- scroll the width of the device's screen to the right
{scroll page left}	- scroll the width of the device's screen to the left
{scroll page up}	- scroll the height of the device's screen up
{scroll page down}	- scroll the height of the device's screen down
{scroll page right+up}	- scroll to the right and up one page
{scroll page right+down}	- scroll to the right and down one page
{scroll page left+up}	- scroll to the left and up one page
{scroll page left+down}	- scroll to the left and down one page
{scroll top}	- scroll to the top edge of the layout
{scroll bottom}	- scroll to the bottom edge of the layout
{scroll left}	- scroll to the left edge of the layout
{scroll right}	- scroll to the right edge of the layout
{scroll to:(+/-)x,(+/-)y}	- *where x,y = upper-left screen coordinates you want to scroll to

* If the x and/or y coordinates are preceded by "+" or "-", the screen will scroll by the number of pixels relative to the current location. Otherwise the coordinates are absolute pixel values.

Keyboard Control

The following special device commands can be used with text field buttons to manipulate the iOS keyboard:

{keyboard show:myTextField}	- shows the keyboard, placing cursor focus in the specified text field
{keyboard hide}	- hides the keyboard

Image Picker

This command displays the iOS photo library image selection screen, allowing you to select an image to use within your own TouchControl activity at runtime:

{imagepicker}

Enable the command button for Feedback and your feedback script will receive the path to the selected image in the `_feedback` variable. For example, to set a button's background image to the selected image, use the following feedback script within the command button:

```
_setImage('myImageButton',_feedback);
```

Note that the selected image is copied from your device photo library and stored within the TouchControl app data store on your device. To access that same image at a later time, store the location contained in the `_feedback` variable to iCloud state using the `_state` variable scripting feature.

Server-side Commands (Windows server only)

You may also use a command button to perform various functions to control various features of TouchControl Server.

Server Sleep

These commands enable or disable TouchControl Server's ability to block the PC from sleeping. Use the following commands in the "Command" field:

{server sleep:disable}	- block the PC from sleeping
{server sleep:enable}	- allow the PC to sleep normally
{server sleep:reset}	- reset to the current TC Server "Block PC sleep/hibernate" setting

Note that these commands will not change the TouchControl Server "Block PC sleep/hibernate" setting, thus allowing you to temporarily override the server setting. Either use the `:reset` command above, or restart TC Server to re-sync the PC's sleep setting with the TouchControl Server setting value.

Screen Grabber Control

You may start, stop, restart, hide, show, move, and re-size the screen grabber directly from Command buttons on your iOS remote layouts. Simply enter the following in the command field when configuring your buttons:

{grabber start}	- start the grabber
{grabber stop}	- stop the grabber
{grabber restart}	- restart the grabber
{grabber hide}	- hide the grabber
{grabber show}	- show/flash the grabber
{grabber pointer:on}	- turn Capture Mouse Pointer on

{grabber pointer:off} - turn Capture Mouse Pointer off
{grabber move:(+/-)l,(+/-)t} - move the grabber
{grabber move:(+/-)l,(+/-)t,(+/-)w,(+/-)h} - move/re-size the grabber

When moving the grabber, "l" and "t" are integers indicating the left and top of the grabber window in screen coordinates, and "w" and "h" are integers indicating the width and height of the grabber window. If any of these are preceded by "+" or "-", it will move or re-size the grabber window by the number of pixels relative to the current position or size. As an examples, if you desire to only alter the size of the grabber, simply enter:

{grabber move:+0,+0,100,100}

...or to only alter the left position...

{grabber move:+25,+0}

USB-UIRT IR Code Re-broadcast

If you are using the "USB-UIRT Broadcast" feature to broadcast IR codes received by the USB-UIRT out to iOS devices on your network running TouchControl, the following command will force TouchControl Server to re-broadcast all unique IR codes received since the server was last started, in the order they were last received by the USB-UIRT:

{rebroadcast ir}

And the following command will clear the re-broadcast queue:

{rebroadcast clear}

Look for more special device commands in future releases of TouchControl.

AutoHotKey Buttons (Windows server only)

Note: AutoHotKey integration is available as an upgrade via an in-app purchase within the TouchControl app on your device. If you have not purchased that upgrade (found under "Settings"), any AutoHotKey buttons you add to your remote layouts will not be accessible on the device. Purchasing the upgrade will enable any existing and future AHK buttons to appear.

AutoHotKey (AHK) buttons allow you to enter AHK scripts directly into TouchControl, and those scripts will be executed when you tap on AHK buttons on your remote screens on the phone. The AutoHotKey executable is distributed with TouchControl server, so you don't need to install it separately (but it is fine if you already have it installed, or wish to also install it yourself for other purposes). After adding an "AutoHotKey" type button, clicking the "Set Data" button for it will open a panel that allows you to enter the AHK script to be executed. You will also find a "Record" button, which will launch the AutoScriptWriter executable which comes with AutoHotKey to record your keystrokes and mouse interaction. Follow the instructions in the prompts when using this feature. You may also test your AHK scripts immediately while editing them. Simply click the "Test" button at any time to test the script as it is displayed in the input panel - no need to save the script first.

AutoHotKey Feedback

AutoHotKey buttons can be configured to receive feedback from your AutoHotKey scripts. When you create an AutoHotKey button in TouchControl Server, you are given the option to check the "Feedback" box, which tells the device app that after it executes the AutoHotKey script, it should wait to receive some text back from AutoHotKey before continuing. That text can then be used to replace the caption on any of your activity buttons or labels, replace the image and/or icon associated with any button or label on your layout, execute other buttons on a layout, etc. To configure and use AutoHotKey feedback:

1. Create an AutoHotKey button and select the Feedback option
2. Use the "Set Data" button to enter the AHK script that will be executed by the button
3. At the end of the script add the following command:

return, "buttonName^buttonText"

Where "buttonName" is the name of the label or button that will receive the feedback value, and "buttonText" is the string value that will be placed on the button or label. Either of those values may be static text, or may be dynamically generated from your AutoHotKey script. Please consult the AutoHotKey documentation for information on generating string values.

Please note that when you designate a button as an AutoHotKey feedback button, when the button is pressed TouchControl will wait up to 10 seconds to receive the feedback. If no feedback is received within that time, TouchControl will not populate your feedback label or button caption, and will simply continue normally. So, it is important that you remember to add the return statement to the end of the AutoHotKey script executed by any AutoHotKey feedback button.

If you wish to process the feedback from this button using custom script, select the "Feedback Script" option and enter/paste your JavaScript in the provided text box. See the [Scripting](#) topic for more information regarding feedback and feedback scripting.

Keystroke Helpers

To help speed the manual creation of AutoHotKey scripts, a "Keystroke Helpers" dialog is available to quickly enter common AutoHotKey keystroke macros. Just click the "Keystroke helpers" checkbox to display the dialog. The keystroke helpers dialog contains three tabbed pages of keystroke macros that you can enter into your script with a single click of a button. The "Ctrl/Alt/Shift" page includes all of the different Ctrl, Alt and Shift variants (such as "{Ctrl Up}", "{LShift Down}", etc. The "Win Keys" page contains helpers for many of the standard Windows keys, and the "FKeys" page contains helpers for {F1} through {F24}.

To hide the "Keystroke Helpers" dialog, simply un-check the "Keystroke helpers" checkbox, or just Save or Cancel your script.

QuickEdit

When you select a device that has one or more AutoHotKey buttons defined, a "QuickEdit" button will display just above the list of buttons for that device which will open a "batch" editing window, giving you quick access to the scripts for all AutoHotKey buttons in that device. This can be helpful when you have many AutoHotKey buttons that perform similar actions and you'd like to copy/paste scripts from one button to another without opening each button configuration separately.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See [this page](#) for more information.

EventTrigger Buttons

To use an EventTrigger button, you must first add a remote "EventTrigger" server or device using the Interface Manager option within Settings in TouchControl Server. EventTrigger (ET) buttons allow you to send data to "event servers" such as EventGhost, or TCP/IP-enabled devices such as A/V receivers, etc., directly from TouchControl on your device. Then when configuring the EventTrigger button, select the desired server or device from the drop-down list.

To configure an EventTrigger button, simply enter the text string that you want to send to the device or server into the EventTrigger Command field in the button configuration. Also set the "Command Terminator" selection to whatever is required for the server or device that you will be controlling. For example, the TouchControl EventGhost plugin requires that all event strings be terminated with a line feed, and Denon AVR receivers require all command strings be

terminated with a carriage return. Please consult your server/device documentation to determine the correct termination character to use, if any.

EventTrigger buttons can be used to send wake-on-LAN messages to your network devices. [Look here](#) to find out how!

HEX Commands:

EventTrigger buttons can also send raw HEX commands to devices. To send a HEX command, enter the following in the EventTrigger command textbox when configuring a button:

0x:64A783FD546D3265

...where "0x:" is a required prefix, and the remaining data is your HEX command, with no spaces or other special characters. HEX commands must contain an even number of characters (not including the "0x:" prefix).

You may also send mixed ASCII and HEX commands. To send a mixed command, enter the following in the EventTrigger command textbox when configuring a button:

MYCOMMAND\x02\x5F\x00\x2F\x41\x0D

...where "MYCOMMAND" is the ASCII portion of the command, and each HEX digit is prefixed with "\x" to denote it's HEXed-ness. ASCII characters may reside at anywhere within the command and may exist in multiple locations in the command.

Note that when entering HEX or mixed commands, the "Command Terminator" character is ignored, and any specific terminator required must be included within the command itself. If you wish to include the literal string "\x" in your command data without converting it to HEX, enter the string as "\\x". If you wish to include both "\x" (HEX) data and the "\\x" literal string within the same command, precede the entire command with the "0x:" prefix as shown above.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See [this page](#) for more information.

EventGhost

You can easily control EventGhost using the EventTrigger interface within TouchControl. Please see the [EventGhost](#) topic for more information.

Global Caché Buttons

Note: Global Caché (GC) integration is available as an upgrade via an in-app purchase within the TouchControl app on your device. If you haven't purchased that upgrade (found under "Settings"), any GC buttons you add to your remote layouts will not be

accessible on the device. Purchasing the upgrade will enable any existing and future GC buttons to appear.

Note: TouchControl is designed to work with GC-100 devices with firmware versions of 3.0 or greater. If your GC-100 device's firmware version is prior to 3.0 and you encounter problems when attempting to communicate with TouchControl you may contact Global Caché support for information on how to upgrade your GC-100 device.

The Global Caché interface in TouchControl allows you to control Global Caché (GC) iTach & GC-100 devices directly from your iOS device, without passing through your TouchControl Server. This interface is available within TouchControl on your iOS device as an in-app upgrade, but like all other TouchControl functionality, can be fully configured and tested in TouchControl Server before purchasing the device upgrade.

[Add a Global Caché interface in TouchControl Server](#)

1. Select *Tools* -> *Settings* from the TouchControl Server menu.
2. Click the Interface Manager button. The "Interface Hosts" panel will open.
3. Click "Add New", and enter the required information for your Global Caché device(s) and click "Save" and then click "Save" again.
4. Finally click "Save" again to save and close the TouchControl Server settings.

[Add Global Caché buttons to your activities](#)

1. On the main TouchControl Server panel, select a TouchControl device (or create a new device and select it in the device list), and click "Add Button".
2. On the "New Button" panel, give the button a name, select the "Global Caché" button type, and optionally set the button as repeating, set the repeat interval, set it as a timer button, and/or enable for feedback as desired.
3. Click "Add" to create the button. (Note that when adding Global Caché buttons, all will display as type "GC" in the button list, whether your device is an iTach or GC-100, as it may be possible to share GC buttons between those devices in some situations.
4. Once the new button is added, highlight the new button in the Buttons list and click "Set Data". The Global Caché Command panel will display.
5. The default GC device (entered in Interface Manager above) will initially display for each GC button. To set the desired device, simply choose it from the Device Name drop down list. If the device you want to control with this button is not listed, click the "Manage" button and add the needed GC device, and then select it when you are returned to this panel.

6. Select the "IR", "Serial", or "Relay" radio button, depending on which type of command you will be sending to the GC device.

GC commands may be added to the button using the following methods:

- If you are entering a Serial command, simply enter or paste it into the empty textbox. Also set the "Command Terminator" to the required terminator for the serial device you will be controlling, or set the terminator to "Nothing" if no terminator is required.
- If you are using a GC IR command that you have obtained from some other source, click the "Edit" button and enter or paste it directly into the empty textbox. The module, connector, repeat and frequency settings will be automatically detected from the command you enter/paste. Change them as needed/required (please consult the Global Caché documentation at <http://www.globalcache.com/downloads> for more information on the required settings for your device).
- You may also learn GC IR commands through an iTach device directly in TouchControl Server:
 1. Click "Learn" to open the IR learn panel.
 2. Set the module, connector, repeat, and separation settings as desired (consult your GC device documentation for the correct values for these settings).
 3. Select "Auto-trim" to force TouchControl Server to automatically trim redundant data from the resulting learned IR command.
 4. Click the "Start" button.
 5. Point your device's original remote at the IR receiver on the iTach device and press the desired remote button to learn. If the command is not detected within 15 seconds, the learning process will timeout and you will need to try again.
 6. When the command is learned, the IR data window will populate with the GC IR command data, and the module, connector, repeat and frequency settings will be detected and populated (it may take more than one attempt to successfully learn a given command).
 7. Click the "Save" button to generate and display the full GC IR command and close the learn panel.
 8. Click "Save" again to save the command to your configuration and close the GC button panel.
- You may import an IR command from an existing TouchControl button, or from raw hex code data (in Pronto CCF format):
 1. Click "Import" to open the import panel.
 2. Select "TouchControl Button" to import from an existing button, or select "Raw HEX Code" to import from CCF hex codes.

3. Set the module, connector, and repeat count you'd like to use for the command (consult your GC device documentation for the correct values for these settings).
 4. If importing from an existing button, select the desired button in the device/button tree view. Only buttons that already contain IR command data will be available to import from.
 5. If importing from CCF hex codes, paste/enter the codes into the code window.
 6. Click the "Import" button.
 7. The resulting GC code will be displayed in the code window.
 8. Click "Save" to save the command to your configuration and close the GC button panel.
- Also, for GC IR buttons, click the "Find" button to open a panel to download ready-to-use codes from Global Caché online IR code database (ControlTower). This panel will allow you to search through device brands, types (i.e. "TV", "Receiver", "Amplifier", etc.), models, and functions to find the exact code needed for the button you are configuring. Over 138,000 code sets are freely available via this feature. You may set the module and connector on this panel, or adjust it on the edit panel, which is displayed after you select a code to use from the online database.
 - Relay (contact closure) commands are added using the provided interface. Using the "Edit" button, you may "Set" the relay state to "1" or "0" and choose the GC module and connector address that you'd like to set. Or you may "Get" the relay state from a specified connector address, in which case you can use feedback script to process the returned state and act upon it accordingly.

Once you have set the command for the GC button, you may drag and drop it on any remote activity layout and configure it as needed.

When configuring Global Caché buttons, you may also specify script variables for the module and/or connector addresses (rather than the normal static numeric values for these settings). Then at run-time, TouchControl will substitute the value of the script variables for the module and/or connector addresses, allowing you to dynamically change the target module and/or connector on the fly. This would primarily be used along with the dynamic IP address variable feature, which allows you to dynamically change the IP address of a remote interface host (such as a Global Caché device) at runtime. For example, if you have multiple Global Caché adapters, each of them controlling a satellite TV set-top box (which would use the same commands), but those boxes are controlled through different module and/or connectors on the various GC adapters, you could set both the dynamic IP address variable, and the dynamic module/connector variables, allowing you to use the exact same remote control layout, but target different devices in different rooms simply by updating a few script variables on the fly.

To set dynamic module and/or connector address variables, select the "Dynamic" check box under the module and/or connector addresses, and enter %varname% in the field provided - including the beginning and ending percent signs, and where "varname" is some unique variable name that you will set via script a run-time. It is your responsibility to ensure that the variables are set, and that the variable values contain numeric values equal to the module and/or connector addresses that you require. If the variables are not found, whatever static value that is set in the button config will be used instead. If the variables are found but do not contain valid module and/or connector values, the button commands will not work.

Similar to the TouchControl "EventTrigger" button type (to control IP devices, for example), when GC buttons are activated on your TouchControl remote screens, the commands are sent directly to the GC device over the network, and do not pass through your TouchControl Server. Therefore, for activities that use GC buttons exclusively, you do not need to have your TouchControl Server running to use them. All other features of GC buttons work similar to other TouchControl button types (as documented here). You may test them in TouchControl Server, add them to macros, set delay touch, use them as AutoExec/AutoAppear/AutoResume/AutoExit buttons, etc., and the free PC Remote feature in TouchControl Server supports Global Caché buttons as well. Please make sure you've read the documentation for your Global Caché device and understand how it works. Support for GC devices can be found at <http://www.globalcache.com>.

GC serial buttons also have the ability to receive feedback from the devices they control. If you wish to process the feedback from this button using custom script, select the Feedback Script option and enter/paste your JavaScript in the provided text box.

GC serial buttons can also send raw HEX commands to devices. To send a HEX command, enter the following in the GC serial command textbox when configuring a button:

0x:64A783FD546D3265

...where "0x:" is a required prefix, and the remaining data is your HEX command, with no spaces or other special characters. HEX commands must contain an even number of characters (not including the "0x:" prefix).

You may also send mixed ASCII and HEX commands. To send a mixed command, enter the following in the EventTrigger command textbox when configuring a button:

MYCOMMAND\x02\x5F\x00\x2F\x41\x0D

...where "MYCOMMAND" is the ASCII portion of the command, and each HEX digit is prefixed with "\x" to denote it's HEXed-ness. ASCII characters may reside at anywhere within the command and may exist in multiple locations in the command.

Note that when entering HEX or mixed commands, the "Command Terminator" character is ignored, and any specific terminator required must be included within the command itself.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See [this page](#) for more information.

HTTP Request Buttons

HTTP Request buttons allow you to send commands/data to HTTP devices, servers or services directly from TouchControl on your device.

Add HTTP Request buttons

- Use [Interface Manager](#) (in TouchControl Server Settings) to add an HTTP Request server to your configuration.
- Add a new button to one of your devices, and select "HTTP Request" as the button type, and optionally select repeating as desired.
- Configure the new button using the "Set Data" button.
 - Select the server you wish to control using the "Host" drop-down list. If the server you wish to control is not listed, click the "Manage" button to add the server, and then select it when you are returned to this panel.
 - Select whether the HTTP Method for the request will be "GET", "POST", or "PUT".
 - Select the "HTTPS" option if the request should use the HTTPS protocol.
 - Enter the path to the web service on in the "Path" field. This is the part of the service URL that follows the host name/IP and port number. This should not start or end with a slash. For example, enter "MCWS/v1/Playback/Stop" to trigger the "Stop" command in J.River Media Server. The Path parameter may be empty if needed.
 - For GET HTTP requests, enter the query parameters for the service, if any, in the "Query" field. This should not start with a "?". For example, enter "Zone=2&ZoneType=Id" for the J.River "Stop" command if you wish to control a different zone. The Query parameter is not required.
 - For POST or PUT HTTP requests, enter any parameters in the "Body" field. The parameters should be in the format:

field=value&field2=value2&field3=value3&etc...

- HTTP Request buttons also support submitting XML data in the "Body" field. This enables the use of services such as UPNP or JSON-RPC, which require posting XML data to the destination server. Simply enter the XML/JSON data directly into the "Body" field when the POST method is selected.
- If you have designated your HTTP request button as a "feedback" button (when creating the button in TouchControl Server), the response from the HTTP request will be returned to the client app. If you would like to process that feedback using custom feedback script, select the Feedback Script checkbox, and enter your custom feedback script as desired in the "Feedback Script" field.
- You can access the HTTP status code from the request using the **_httpStatusCode** script variable within the HTTP Request button's feedback or post script.
- If your HTTP request requires any custom HTTP headers, those headers may be added to the request using the button properties feature. Button properties can be accessed either by right-clicking a button in the Buttons list, or after a button has been dropped onto a layout by right-clicking on the button within the layout and selecting "Properties..." from the popup menu. All HTTP Request buttons will have a built-in property named "HTTPHeaders". Select this property, and enter the custom headers for the HTTP request in the following format:
headername1=headervalue1^headername2=headervalue2^headername3=header value3^etc...
- Once you've entered the desired headers in the value field, click the + button to add the headers to the button's properties list, then click Save to exit the properties dialog.
- If your HTTP request required authentication, the needed credentials (username and password) may be added to the request using the button properties feature as well. All HTTP Request buttons will have a built-in property named "HTTPAuth". Select this property, and enter the authentication credentials in the following format:
username:password
- Once you've entered the desired credentials in the value field, click the + button to add the credentials to the button's properties list, then click Save to exit the properties dialog.

You may test the button from TouchControl Server by clicking the "Test" button on the button configuration panel.

This feature is not used to load web pages to view on your device. To load a web page in a remote activity screen, use the new "Web View" button type. Also, to launch the browser on your device and load a specified URL, use the "URL" button type.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See the [Advanced Scripting](#) topic for more information.

iRTrans Buttons

Note: iRTrans integration is available as an upgrade via an in-app purchase within the TouchControl app on your device. If you haven't purchased that upgrade (found under "Settings"), any iRTrans buttons you add to your remote layouts will not be accessible on the device. Purchasing the upgrade will enable any existing and future iRTrans buttons to appear. The iRTrans interface in TouchControl allows you to control iRTrans LAN devices directly from your iOS device, without passing through your TouchControl Server. This interface is available within TouchControl on your iOS device as an in-app upgrade, but like all other TouchControl functionality, can be fully configured and tested in TouchControl Server before purchasing the device upgrade. TouchControl does not support iRTrans USB devices.

IMPORTANT: The minimum required iRTrans firmware version supported by TouchControl is 1.11.00.

Use [Interface Manager](#) (in TouchControl Server Settings) to add an iRTrans host to your configuration.

Add iRTrans buttons to your activities

- On the main TouchControl Server panel, select a TouchControl device (or create a new device and select it in the device list), and click "Add Button".
- On the "New Button" panel, give the button a name, select the "iRTrans" button type, and optionally set the button as repeating, set the repeat interval, set it as a timer button, and/or enable for feedback as desired.
- Click "Add" to create the button. (Note that when adding iRTrans buttons, they will display as type "TRANS" in the buttons list).
- Once the new button is added, highlight the new button in the Buttons list and click "Set Data" (or double-click the button in the list). The iRTrans Command panel will display.
- The default iRTrans device (entered in Interface Manager above) will initially display for each iRTrans button. To set the desired device, simply choose it from the Host drop down list. If the iRTrans device you want to control with this button is not listed, click the "Manage" button and add the needed iRTrans device, and then select it when you are returned to this panel.
- iRTrans commands may be added to the button using the following methods:

"Edit" mode

This method may be used to select commands stored internally in your iRTrans module.

- After clicking the "Edit" button, you may then select the type of command to use for this button. Selecting the "snd" or "sndr" command types will access your iRTrans module and list all internally stored remotes and their associated commands. Simply navigate the remote/command list and select the command you wish to use for this button.
- Alternately, selecting the "sndhex", "sndccf", or "sndccfr" command types will present you with a blank input box where you may manually enter or paste the command to use for this button. Please consult your iRTrans device documentation for more information on the correct format and content for these types of commands. When selecting "sndhex", "sndccf", or "sndccfr" command types, you do not need to include the command type within the command itself - it will be automatically added for you and shown to you after saving the command.
- Lastly, selecting the "[other]" command type will allow you to enter a fully custom command as needed.
- When in edit mode, each of the different command types also have additional options you may select if desired to further configure the commands (listed under the "Options" title to the left of the command input box on the edit panel). These settings have default values that will be used unless you change them. Please consult your iRTrans documentation for more information on these options and their use.

"Learn" mode

You may also learn iRTrans IR commands through the iRTrans LAN device directly in TouchControl Server:

- Click "Learn" to open the IR learn panel.
- The "sndhex" option will be automatically selected, which is required for any learned HEX codes.
- Click the "Start" button.
- Point your device's original remote at the IR receiver on the iRTrans device and press the desired remote button to learn. If the command is not detected within 15 seconds, the learning process will timeout and you will need to try again.

- When the command is learned, the IR data window will populate with the iRTrans IR command data (it may take more than one attempt to successfully learn a given command).
- Change any of the optional settings to the left of the IR code window. See your iRTrans documentation for more info on these settings. The default settings should work in the majority of cases.
- Click the "Save" button to generate and display the full iRTrans IR command and close the learn panel.
- Click "Save" again to save the command to your configuration and close the iRTrans button panel.

"Import" mode

You may import an IR command from an existing TouchControl button, or from raw hex code data (in Pronto CCF format):

- Click "Import" to open the import panel.
 - Select "TouchControl Button" to import from an existing button (Windows only), or select "Raw HEX Code" to import from CCF hex codes (on macOS, importing raw codes is the only available option).
 - If importing from an existing button, select the desired button in the device/button tree view. Only buttons that already contain IR command data will be available to import from.
 - If importing from CCF hex codes, paste/enter the codes into the code window.
 - Click the "Import" button.
 - The resulting iRTrans code will be displayed in the code window.
 - Change any of the optional settings to the left of the IR code window. See your iRTrans documentation for more info on these settings. The default settings should work in the majority of cases.
 - Click "Save" to save the command to your configuration and close the iRTrans button panel.
- Once you have set the command for the iRTrans button, you may drag and drop it on any remote activity layout and configure it as needed.

Similar to the TouchControl "EventTrigger" or "Global Caché" button types, when iRTrans buttons are activated on your TouchControl remote screens, the commands are sent directly to the iRTrans device over the network, and do not pass through your TouchControl Server.

Therefore, for activities that use iRTrans buttons exclusively, you do not need to have your TouchControl Server running to use them. All other features of iRTrans buttons work similar to other TouchControl button types (as documented here). You may test them in TouchControl Server, add them to macros, set delay touch, use them as AutoExec/AutoAppear/AutoResume/AutoExit buttons, etc., and the free PC Remote feature in TouchControl Server supports iRTrans buttons as well. Please make sure you've read the documentation for your iRTrans device and understand how it works. Support for iRTrans devices can be found at <http://www.irtrans.com/>.

Pre-Script & Post-Script

Enter any desired pre- or post-script for the button. See [this page](#) for more information.

Macro Buttons

Macros (multiple commands sent with one button click) can easily be created by adding a button of type "Macro" to any device. Macros are created by combining other existing buttons in a list, which are then executed in order when the macro button is activated (pressed). To create macros from existing buttons, use the "Button Macro" option at the top of the Macro panel (accessed by selecting a macro button and clicking the "Set Data" button). Using this option will present an interface for visually generating macros by selecting existing buttons (which already have commands defined) and adding them to the macro using the "+" button. You may also add pauses (both the 1 sec. pause - "!", and the .1 sec. pause - "."), rearrange the macro commands, and remove commands from the macro by clicking the appropriately labeled buttons.

When adding pauses, you can configure those pauses to repeat up to 600 times in succession. This keeps you from needing to add a sequence of multiple pauses to make your macro pause some lengthy amount of time. To repeat a pause, add a pause to your macro, then right-click on the pause symbol ("!" or ".") in the macro list, and use the provided numeric control to set the number of times that pause will occur in succession at that location in the macro. The pause repeat count will be displayed next to the pause symbol.

Blocking vs. non-blocking pauses

By default, pauses you add to macros are "non-blocking", meaning that when the macro reaches the pause, any other button presses that have occurred while the macro was processing up to the pause will process during the pause. If you would rather all other button presses wait until after the macro has completed, you can make the pauses "blocking". To create a "blocking" pause, right-click on the pause symbol ("!" or ".") in the macro list, and use select the "blocking" option on the panel that appears. An "X" will appear next to the pause symbol to alert you that it is now a blocking pause.

Macros can include IR, Command, AutoHotKey, EventTrigger, Global Caché, HTTP Request, URL, or Link buttons. Macros may also contain other macro buttons. Note that any macros that contain other macros that have already been included via another macro will be ignored to eliminate the possibility of an endless loop.

IMPORTANT: Be careful when executing macro buttons from a slider, gesture pad, or from other repeating buttons, especially if the macro includes long-running buttons, buttons that request and process feedback, or includes pauses. Commands can be sent very quickly when being executed in this manner, and if commands must execute in a given order, or wait for feedback to arrive and then be processed, fast/repeated executions of macros can cause unexpected results. For this reason, TouchControl will not allow more than one instance of a given macro to be running at any given time. If a macro is running and another instance of the same macro is attempted to be executed (such as when executing from a slider, gesture pad, etc.), that subsequent instance of the macro will not run. It will not queue up and run later – it simply will be skipped.

MacroMessage

If you would like to display a message on the screen while a long-running macro is executing, enable the "MacroMessage" feature. When enabled, TouchControl will look for a specifically-named Label button on your activity to display over all other buttons on your layout. This label will appear when the macro begins executing, and will disappear when the macro ends.

With this feature enabled, TouchControl will look for a Label button named "_macroMessage" (note the button MUST be a Label type button, and the name is case-sensitive) on your layout. If found, that button will be re-sized/positioned over your layout using size/position parameters that you specify. The _macroMessage button can contain any text that you wish by configuring the text of the label, and may be visually configured (image, colors, etc.) just as any other label using the configuration options on the pop-up menu when right-clicking on the label on your layout. To provide size and position parameters, add the "DisplayFrame" property to the label (right-click on the label and select Properties..., then select the "DisplayFrame" property from the "Name" drop-down list). The value for the DisplayFrame property must be in the following format:

left,top,width,height

The parameters for this property may contain exact pixel positions, (such as 100,100,200,300), or may contain the following size and position values/keywords:

percentage - when supplying a percentage value (e.g. 80% or .8) in the left or top location, positions the label the given percentage from the left or top of the

screen, and when used in the width or height location, sizes the label at the given percentage of the screen width or height.

center - when used in the left location, centers the label horizontally on the screen, and when used in the top location, centers the label vertically on the screen (case-sensitive)

full - when used in the width location, sizes the label to fill the screen horizontally, and when used in the height location, sizes the label to fill the screen vertically (case-sensitive)

Using the `DisplayFrame` property allows you to add the `_macroMessage` label to your layout as a small, disabled button and locate it anywhere on your layout, and `TouchControl` will re-size and re-position it according to your `DisplayFrame` configuration at run-time. As an example, use `0,0,full,full` to cover the entire screen on any device with the `_macroMessage` label while a macro is executing.

Press & Release macros

An additional type of macro is the "Press & Release" button macro. Press & Release macros are simple two-step macros that execute on command on button press, and another command on button release. These macros, created in the same manner as normal macros (above), allow you to add a single button to the press action, and a single button to the release action. Press & Release buttons may be held "down" as long as desired, to space out the press & release actions as long as necessary for your use. This can be especially useful, for example, when used with `AutoHotKey`, to send key "down" commands (i.e. `{LCTRL DOWN}`) to your computer on press, and then key "up" commands (i.e. `{LCTRL UP}`) to your computer on release.

Release-only buttons

A variation on the Press & Release macro, you may optionally specify an activity only for the "release" action in a Press & Release macro to generate a button that only sends its command once you release it. Once you press the button, as long as your finger does not move, releasing the button will send the button's command. If, while holding down the button, your finger moves, this will cancel the command. This can be useful to provide a safeguard for those buttons that you may accidentally press while grasping the device (but not intending to press any button), but which can cause headache or havoc if executed at the wrong time - such as "All Off". You can also specify a "Pressed image" for the button if desired so that it changes state while you are pressing it to give you some visual feedback.

Slider Buttons

"Slider" buttons are visual controls used to select a single value from a continuous range of values. Sliders are displayed as bars, with an indicator, or thumb, indicating the current value of the slider which can be moved by the user to change the setting.

Sliders are implemented in TouchControl as a “composite” button type, meaning they are buttons that are made up of, or execute the commands of, other pre-defined buttons. The first step in using a slider is to create buttons that execute the commands that you want the slider to execute when you move the slider’s thumb. Sliders can be used, for example, as an easy, visual, quick, and intuitive way to change volume for a device.

IMPORTANT: Be careful when executing buttons from a slider that take a long time to complete, that request and process feedback, or when executing macros from sliders. Commands can be sent very quickly when moving a slider back and forth, and if commands must execute in a given order, or wait for feedback to arrive and then be processed, fast/repeated executions of long-running buttons or macros can cause unexpected results. For this reason, TouchControl will not allow more than one instance of a given macro to be running at any given time. If a macro is running and another instance of the same macro is attempted to be executed (such as when executing from a slider), that subsequent instance of the macro will not run. It will not queue up and run later – it simply will be skipped.

When configuring a slider button, you select the slider’s minimum, maximum, and increment values. These define the starting and ending values for the slider, as well as the values at each position along the slider’s bar. You may also select an initial value for the slider, which to force the slider’s thumb to a given position when it is initially displayed.

The action triggered by the slider at each position can be defined using two different methods:

Action when sliding

This method allows you to define a primary TouchControl button (IR, Command, AutoHotKey, EventTrigger, or HTTP Request) which will be executed at each position along the slider’s bar when sliding in each direction. Simply select the device/button to execute from the drop-down lists for both the “Left” and “Right” slider directions.

Action at each stop

This method allows you to define a single, primary TouchControl button (IR, Command, AutoHotKey, EventTrigger, or HTTP Request) which will be executed at each “stop” or “position” along the slider’s bar. Simply select the device/button from the drop-down lists. The configured code for this button contains a special flag that, when used as a slider action, is dynamically replaced with the slider value before the button is executed. For example, to adjust the volume on a Denon receiver, a volume button used with a slider would have code that looks like this:

MV%value%

In this example, “%value%” is replaced at runtime with the value from the slider, resulting in the command MV30 (for example) being sent to the receiver when the slider passes over or lands on the value “30”. So, wherever you place the string %value% in the button’s command will be replaced with the value from the slider at each stop/position before executing the specified button command.

If the device you are controlling requires HEX values as commands (such as the Insteon SmartLinc network adapter, and many others), you may force TouchControl to convert the slider's value to HEX by using the substitution string "%0x:value%" instead. This will convert the slider's integer value to its two-character HEX equivalent before substituting it into the command (i.e. the integer value 0 would convert to "00", 10 would convert to "0A", 255 would convert to "FF", etc.).

"Snap to touch" slider interaction

When using the "Action at each stop" slider type, you may now specify that the slider should "Snap to touch", which allows you to touch the slider at any location, and the slider value will "snap" to that location. Without this feature enabled, the slider only updates as you slide your finger across the slider bar.

Action on release

When configuring any slider, you may now specify an additional action to occur when you release the slider. For example, if you would like to execute one command while sliding the slider, and then execute a different command only once that uses the slider's final value, you could use the "Action on release" to perform that final action. Or alternately, if you don't want to execute any command while adjusting the slider, but then execute a single command when you are finished, you may select the "Action at each stop" slider type and select "[none]" for the action's device, then specify the command to execute on release only. This basically allows you to freely slide the slider without executing any commands, and then just execute a single command on release. This would be useful if the command you want to execute is either long-running, or possibly runs feedback script that would result in "jerky" slider operation.

TouchTips

Slider buttons can also show a TouchTip, which is a small "bubble" that appears above your finger when you slide the thumb image back and forth over the slider. For slider buttons, the TouchTip displays the slider's current value. To enable the TouchTip, just right-click on the slider in the layout designer and select "Show TouchTip".

Script

The “Script” button will display a field that will allow you to enter JavaScript which will execute when you release the slider with your finger (after the "action on release" executes), and also

when you execute a slider via script using the [#] script return string element. See the [Scripting](#) topic for more information.

Note that if you use full-screen activities and you add a slider button to a layout, the default activity swipe gestures that take you back to the previous screen could interfere with the swiping gesture required to move the slider. If this is the case, you may wish to [disable activity swiping via script](#).

Spinner Buttons

“Spinner” buttons use a spinning-wheel or slot-machine metaphor to show one or more sets of values. Users select values by rotating the wheels so that the desired row of values aligns with a selection indicator.

Spinners are implemented in TouchControl as a “composite” button type, meaning they are buttons that are made up of, or execute the commands of, other pre-defined buttons. So, the first step in using a spinner is to create buttons that execute the commands that you want the spinner to execute when you spin the wheel. For example, 0-9 channel buttons, or volume up/volume down buttons, etc.

Select the type of spinner you would like:

Numeric: 0-9: This type of spinner adds the numbers 0 through 9 to each segment of the spinner’s wheel. You may set any number of sections from 1 to 10. Using channels again as an example, your cable/satellite box may allow up to 4-digit channels, so you would set the number of sections of the spinner to 4. This would create a spinner that would allow you to select any 4-digit number from 0000 to 9999, and then send those numbers using the corresponding channel digit buttons. This effectively creates a mini macro sending the four button clicks as a series of commands. This spinner type automatically looks for buttons named “0” through “9” to execute for each value selected on the spinner, so those need to be created and available before using the spinner. Those buttons do not need to be added to your activity.

Numeric: Min-Max: This type of spinner adds the numbers from a minimum that you specify to a maximum that you specify, with the given increment that you specify. Using volume as an example for this type, say you have a receiver that allows you to set the volume from 0 to 90. You would set the “Min” value for the spinner to 0, and the “Max” value to 90. For the most granular control over the volume, you would set the “Increment” to 1 to give you 91 possible volume levels to scroll through. Or, if you’d like the volume to be a little more responsive, you could set the Increment to a larger number so that the volume would change more for each “click” of the spinner. If the practical volume range

is really more like 30 through 60, you could make those the min and max to cut down on the number of options available on the spinner. This type of spinner also gives you the ability to display the numbers (min to max for the given increment) on the spinner, or alternately display black bars that grow from left to right for each value on each row of the spinner. This just gives you an alternate visual option to display on the spinner rather than the numbers. For this type of spinner, a single button is specified which is executed at each stop of the spinner wheel. The configured code for that button contains a special flag that, when used as a spinner action, is dynamically replaced with the spinner value before the button is executed. For example, to adjust the volume on a Denon receiver, a volume button used with a spinner would have code that looks like this:

MV%value%

In this example, “%value%” is replaced at runtime with the value from the spinner, resulting in the command MV30 (for example) being sent to the receiver when the spinner lands on the value “30”. So, wherever you place the string %value% in the button’s command will be replaced with the value from the spinner at each stop. This button does not need to be added to the activity.

Buttons: This type of spinner lets you select multiple buttons to display on each row of the spinner, and then those buttons will be executed whenever the spinner lands on the corresponding value. No command substitution is done for these buttons – they are simply executed as they are configured. This gives you an option for adding more buttons to your activity than may fit on the screen, while still giving an easy way to access and execute the buttons.

Free Text: Free Text spinners allow you to enter any text you wish for the spinner selections, and then execute custom script when an entry is selected. Free Text spinners can also be displayed as a scrolling table list, with a check mark indicating the selected entry (rather than the standard spinner's normal selection bar). The table option allows you to display a larger list on the screen than normal spinners, which are limited in their maximum height by the operating system on the iOS devices.

Free Text spinners can also display images in their rows (in either spinner or table mode).

The buttons used as commands for any of the first three types of spinners can be any of the “primary” button types, including IR, Command, AutoHotKey, EventTrigger, or HTTP Request. Those buttons may also be feedback buttons and contain feedback script, and the feedback and script will be processed just as if the buttons were normal stand-alone buttons.

Spinner buttons can either be placed freely within a remote layout (just like any other button), or they can be "docked" to the bottom of the activity view. Spinner buttons which are "docked" can also be set to "auto hide", which initially hides the spinner at the bottom of the screen, displaying only a thin bar with the spinner's name, and then "popping up" from the bottom of the view when that bar is tapped.

In the case of "Numeric: 0-9" spinners or "Custom" spinners, when the spinner is showing on the screen it will have a "Go" button above it which will send the entire series of digit-button commands (0-9), or the custom button command when pressed. This allows you to set all sections of the spinner to the desired digits and then execute them all together, or re-execute the command without first having to change the spinner to re-select the entry(s). If the spinner is set to auto-hide, the spinner will hide itself after it sends the button commands.

In the case of "Numeric: Min-Max" spinners, the configured button is executed immediately when the spinner comes to rest on a row/value. If the spinner is set to auto-hide, the spinner will hide itself when the button bearing its name immediately above it is tapped.

Grids

A unique button presentation feature is available - referred to as a "grid" - which allows you to lay out multiple buttons in a free-flowing, independently-scrolling grid layout within your activities. The grid is configured as a spinner button, using the "Buttons" mode, which allows you to add buttons from your configuration to the spinner (or grid in this case), and then selecting the "Display as grid" option in the spinner configuration will trigger the spinner to display as a grid on your iOS device. Please see [this page](#) for more information.

You may set a spinner's background color, select a background image for a spinner, and/or set the spinner's text color via the normal designer menu options once added to an activity layout.

Gesture Pad Buttons

Gesture Pads are buttons that recognize different gestures on the iOS device screen. The supported gestures are:

- Swipe Up: Swipe your finger in an upward direction over the gesture pad
- Swipe Down: Swipe your finger in a downward direction over the gesture pad
- Swipe Left: Swipe your finger to the left over the gesture pad
- Swipe Right: Swipe your finger to the right over the gesture pad
- Rotate Left: Place one finger on the gesture pad and rotate it counterclockwise
- Rotate Right: Place one finger on the gesture pad and rotate it clockwise

- Tap: Tap your finger once on the gesture pad
- Double Tap: Tap your finger twice on the gesture pad

For swipes and rotates, the gesture pad will send continuous commands during the life of the swipe or rotation (as long as your finger continues to move on the device's screen). That is, it will act like a hold-to-repeat button, repeating commands during the gesture. Taps will only send a single tap when the gesture is recognized.

IMPORTANT: Be careful when executing buttons from a gesture pad swipe or rotation that take a long time to complete, that request and process feedback, or when executing macros from gesture pads. Commands can be sent very quickly when swiping or rotating, and if commands must execute in a given order, or wait for feedback to arrive and then be processed, fast/repeated executions of long-running buttons or macros can cause unexpected results. For this reason, TouchControl will not allow more than once instance of a given macro to be running at any given time. If a macro is running and another instance of the same macro is attempted to be executed (such as from a gesture pad swipe or rotation), that subsequent instance of the macro will not run. It will not queue up and run later – it simply will be skipped.

After adding a gesture pad to your configuration, when you click the "Set Data" button for the pad, you will be presented with a configuration dialog in which you will configure the command to be sent for each desired gesture. Select the checkbox next to each gesture that you'd like the gesture pad to recognize, and click the "Action >>" button to select its action. Gesture pad actions are other buttons that you've already configured with IR, command, AutoHotKey, EventTrigger, Global Caché, or HTTP Request functions. Therefore, you must configure individual buttons with the commands you'd like the gesture pad to send, but those other buttons do not need to be placed on your activity layouts. For example, to control the volume, you'd create "Vol+" and "Vol-" buttons configured with the desired IR, command, AutoHotKey, EventTrigger, etc. data, and then create a "Volume" gesture pad which uses "Vol-" as the rotate left action, and "Vol+" as the rotate right action.

Hold To Repeat

Up, down, left and right swipe gestures on a gesture pad can optionally be configured for "Hold to repeat". This feature allows you to swipe in a given direction, and when you pause the swipe but leave your finger on the screen, the gesture pad will continue to send the action commands for the current swipe direction without further swipe movement required. Lifting your finger off the screen or continuing the swipe motion will cancel the repeating commands. Each distinct swipe direction may be independently configured for hold to repeat. Repeated commands are sent at the rate defined by the gesture pad's "Repeat interval" setting.

2-Stage Buttons

2-stage buttons allow you to place a gesture pad on a layout, and when that gesture pad is tapped or double-tapped, a “real” button appears over your layout which you can tap to execute the intended command. This could be useful if you have a layout with many buttons on an iPhone or iPod, and don’t have room for all of the buttons’ full text. A small image or hot-spot could be used for each button, and then when tapped, a “clickable” button would appear with the full button text. 2-stage buttons also display a dismiss icon (red “X”) that allows you to dismiss the button without actually executing it. This could also be useful for a button that has a long running macro (e.g. starting up your theater, adjusting lights, opening/closing blinds, etc.), and you sometimes accidentally tap it at the wrong time. Using a 2-stage button, you would in essence have the chance to confirm or dismiss the actual execution of the button before possibly executing a destructive command or macro.

2-stage buttons are implemented via the tap and double-tap gestures on a gesture pad. When you select the “Tap” or “Double Tap” options during gesture pad configuration, you may optionally select the “2-Stage” option, and select the “action” for the tap gesture just as you would for any other gesture. The button selected as the tap/double-tap gesture “action” will be the button used for the 2-stage command.

Gesture Pad Mousepad (Windows server only)

Gesture Pads can also act as stand-alone mousepads. When configuring the gestures for a Gesture Pad, select "Mousepad" and the entire gesture pad will automatically expose the gestures needed to perform mouse and keyboard operations. The following gestures are supported on Gesture Pads in mousepad mode:

- One-finger swipes for mouse movement
- Two-finger swipes for scrolling (horizontal and vertical)
- One-finger tap for mouse left-click
- Two-finger tap for mouse right-click
- One-finger long-press for mouse drag (one-finger long-press or one-finger tap to end dragging)
- Two-finger long-press to show keyboard
- Two-finger pinch zoom in/out

One or more Gesture Pad Mousepads may be added to any activity layout, and can be resized and/or rotated, and can have any image background, just like any other button. You may also drop other buttons around or on top of your Gesture Pad Mousepads within your layouts to create a completely custom mouse/keyboard-enabled activity.

Script helper functions are also available to show/hide the mousepad keyboard and text input field (in addition to using the two-finger long-press). The following functions can be used in a script button or in any script in any other button type to show or hide the mousepad keyboard:

- `_showGPKeyboard('padName');` - Show the keyboard for a specified gesture pad mousepad
- `_hideGPKeyboard();` - Hide the gesture pad mousepad keyboard
- `_toggleGPKeyboard('padName');` - Show/hide the keyboard for a specified gesture pad mousepad

Redirect Mouse & Keyboard Control

Gesture Pad Mousepads can also be used to control the mouse and keyboard on computers other than the primary TouchControl server system. All Gesture Pads have a built-in property named "MouseServer". Set this property to a value equal to the IP address and TouchControl Server port of the Windows PC you want to control as follows:

192.168.1.100:8822

The port can be found on the settings page of TouchControl Server on the system you want to control. When the activity with this Gesture Pad Mousepad renders, all mouse movements and keyboard commands will be sent to the specified server. Therefore, the requirement for this is that TouchControl Server be running on the alternate PC. It can be minimized to the system tray, etc., to keep it out of the way, and does not need to be used for any configuration or other function if not needed.

The alternate system for the gesture pad Mousepad can also be set or changed at runtime. Simply use script to change the "MouseServer" property, either setting a new IP:Port to target a different system, or set the property to a blank string to default back to the primary TouchControl Server (the server where the client's configuration is refreshed from). Set the property via script as follows:

```
_setProperty('myGesturePad', 'MouseServer', '192.168.1.101:8822'); // set to alternate server
```

```
_setProperty('myGesturePad', 'MouseServer', ""); // default to primary server
```

This would allow you to use one mousepad to control multiple systems, switching between them with just a tap of a button (which would run the above script).

Swipe Velocity

When configuring a gesture pad to use swipe gestures, or as a gesture pad mousepad, enabling the "Use swipe velocity" option will use your swipe velocity to adjust the resulting actions - commands will be executed at a faster rate as the velocity of a swipe increases, and the velocity of the mouse movement on your computer screen will be increased as you swipe faster on the mousepad on your iOS device. There is currently no option to adjust the swipe velocity setting. NOTE: Please be aware that when using full-screen activities on the iPhone/iPod, other swipe gestures (swipe down and/or swipe right) are added to the activity window to allow you to return to the main activities home screen (since there is no navigation bar with the "Activities" button to take you back to that screen). When using any Gesture Pad with a swipe gesture (including a Gesture Pad Mousepad), these swipe gestures will be disabled, as they can interfere with the swiping action of the Gesture Pad button. In this case, use the long-press gesture on your activity background (i.e. not on any button) to display the navigation bar which will allow you to return back to the main activities screen or last activity.

Link to Activity Buttons

Link to Activity buttons allow you to add buttons to your layouts that, when tapped, will load another TouchControl activity. When configuring a link button, you must select which other activity will be opened when the button is tapped, or you may select "[go back]" as the link action, which will simply return you to the previous screen.

Link to Activity buttons, by default, will retain the history of the current activity so that when you "go back" from the new activity, you will be back to the original activity that contained the link button. However, you may also specify that the link button should not retain history (unselect the "Retain History" option), in which case when you return from the linked-to activity, you will return directly to the "Activities" home screen (or the activity *before* the last activity, if you have linked multiple times to get to the currently displayed activity).

One important item to note about Link buttons is that if you have many activities that link to each other, TouchControl will ensure that there is only one instance of any given activity "alive" at any given time. For example, assume you have activities A, B, and C. If you link from activity A to activity B, they are both "living" within the app. If you then go "back" from activity B to activity A, activity B is gone and only activity A is "alive". Makes sense. So now assume you link from activity A to activity B, and then from activity B to activity C. Now you have three activities in the "stack" of living activities within the app - A->B->C. But then instead of going "back" from activity C to activity B, you link *forward* to activity A again. In this case, instead of loading a new instance of activity A, TouchControl finds the existing activity A in the "stack" of activities and moves it to the top. So now you still have three activities in the stack, but they are now ordered B->C->A. So, if you go "back" from A, you'll find C, then back from C to B, the back from B to the main activities screen. A very important thing to understand about this is that

when activity A is moved from the bottom of the "stack" to the top, it maintains its "state" as far as any scripting variables, or AutoExec, or UI updates are concerned. So, if you expect an AutoExec on Load button to execute each time you enter activity A, for example, in this case it will not execute, because the "state" of activity A has been preserved, and the AutoExec on Load would have already executed when A was originally loaded. One way around this, and something that might be suggested if you have many links from activity to activity and possibly have multiple "paths" of getting to any one activity, is to turn off the "retain history" setting on your link buttons, so that you always only have one activity living in the app at any given time. This will not only ensure that you get a "fresh" instance of any given activity each time you enter it, but will also help with memory consumption so that you don't have many activities loaded simultaneously that you may not go back and use during a given session. Another option is to use an AutoExec on Appear button, which executes each time an activity appears on the screen, whether or not it is already living somewhere down in the stack of activities.

Dynamic Links

When configuring a Link button, you may select the "Dynamic" option, which allows you to enter a `%varname%_local` or `_global` dynamic substitution variable name (rather than selecting a static activity), which will be queried at run-time to determine the activity to link to. The variable name must exist in the `_local` or `_global` script object, and the value of the `_local/_global.varname` variable must contain an activity in the following format:

location^activity

...where *location* is the name of any location in your configuration, and *activity* is the name of any activity within that location, joined by the "^" character. An example of the above would involve simply entering:

`%myLinkActivity%`

into the Link to Activity field, and then at run-time, create a variable in script as follows:

```
_local.myLinkActivity = "Theater^Watch TV";  
_global.myLinkActivity = "Theater^Watch TV";
```

Pressing the link button on your layout will then link to the "Watch TV" activity in the "Theater" location (as an example).

You may also manage the location and activity separately in the activity link field as follows:

`%locationVar% - %activityVar%` <-- uses two separate variables

%locationVar% - MyActivity <-- uses a variable for the location and specifies a static activity name
MyLocation - %activityVar% <-- uses a variable for the activity and specifies a static location name

In the above examples, the variables (`_local/_global.locationVar` and `_local/_global.activityVar`) must contain only the location or activity name (respectively), and the location and activity entered in the field must be separated with " - ", just as all other location - activity pairs in the drop down "Link to Activity" list.

Note that the location and/or activity names referenced in the `_local` or `_global` variables must match exactly, including upper/lower case, etc. It is your responsibility to ensure that the `_local/_global.varname` variable contains the desired location and/or activity names prior to executing the link button at run-time. If the variable does not exist or does not contain a valid location and/or activity name, the link button press will be ignored.

Link Pre-Script

When configuring a link button, you can supply pre-script which will run immediately before the link is executed. This could be useful in conjunction with the above "dynamic link" option to modify the destination of the link before it is executed. Also, just as with other button pre-script, including the string '[*]' embedded somewhere in the value returned from the script will cancel the button command, which in this case will cancel the link action.

Background Links

Link to Activity buttons may also be specified as "Background", in which case the linked-to activity will become a "background" activity. Background activities are activities that appear "behind" other activities – or in the "background". These activities give you quick access to additional buttons that may not fit on a main activity screen if you would like to keep your activities to a single screen without the need to scroll. Any activity linked to using a "Background" link button will cause the main activity screen to "curl" up to the top of the screen, revealing the linked-to activity as if it were sitting behind the main activity. While the background activity is displayed, no other activity can be linked to (so a background activity can't have a background activity).

To return to the main activity (un-curl the page down from the top of the screen), you can use the one-finger long press or swipe gestures, or you may add a "Link to activity" button that is configured for "[go back]". This is because the page curl animation also hides the navigation bar, so the standard "back" button is not available in this mode.

Popover Links

Link to Activity buttons may also be specified as "Popover" links (for the iPad only), in which case the linked-to activity will be displayed using the iPad's built-in popover functionality - which displays the activity in a temporary "window" over top of the currently displayed activity. The popover will automatically size itself to the content found within the popover activity - sizing to the background image by default. If no background image is specified for the popover activity, the resulting popover will size itself to the bounds of the buttons found within the activity, allowing you to create very small or very large popovers as desired. If the linked-to activity is larger than will fit within the resulting popover, the activity will simply scroll within the popover.

To enable this functionality, when creating a link button, simply select the "Popover" option to the right, and the activity selected for the link will be displayed within the popover when the link button is tapped. To dismiss the popover, simply tap anywhere outside the popover, as all gestures on the current activity are suspended while the popover is visible on the screen.

Custom transitions

When you link to activities, you can specify custom transition animations to show the new activities on the screen. Slide in from any direction, flip from any direction, curl up or down, zoom in, or dissolve (fade in). Custom transitions are defined within the Link buttons that link to the activity, and in addition to the type of transition, also allows setting the duration (speed) of the transition animation. Note that [go back] link buttons cannot have custom transitions, as exiting an activity simply reverses the transition used to link to the activity. Also, activities loaded from the home activities screen will use the default sliding transition.

Labels

The "Label" button type isn't really a button at all. A label just lets you place any text and/or image you'd like anywhere on your activity screen. When creating a Label, you are given a text box in which you can enter the text to display on the label. You can also leave the text box empty, and dynamically populate the text of the Label from button feedback or other script values. Labels can be made any size, rotated, and given any background image, just like any other button.

You may specify any label as a "Marquee" label when creating the label button. This will cause the label to repeatedly scroll its text from right to left within the bounds of the label on the iOS device's screen. (This option is not supported in the PC Remote feature on your server, or in WebRemotes).

Web Views

Web Views are not actually buttons, but rather an embedded view within your activity layout in which you can load a web page, images, raw HTML, or anything you can load over HTTP, that your device has access to. Simply add a button of type "Web View" to any device, and set the URL you wish to load into the web view in the button configuration. This differs from URL-type buttons in that the web page loads within TouchControl, rather than launching an external browser.

You can also specify that the web view should automatically reload at a given interval. When defining the web view "button", simply select the "Refresh" option, and set the refresh rate anywhere from 0 second to 600 seconds between refreshes. A refresh rate of 0 (zero) will cause the web view to continually refresh with no pause in between. You can also manually refresh a web view by double-tapping it on your device screen.

By default, web views may be manually reloaded by double-tapping the web view on your iOS device screen. To disable this functionality, simply un-check the "Double-tap to refresh" option when creating or editing the web view button.

Web views may also be defined as "Interactive", which allows the links, buttons, and/or script in the contained web page to interact with TouchControl buttons which also exist in the activity layout. This allows you to design at least a portion of your layout using HTML, CSS, XSL, etc., yet still retain use of TouchControl features not supported in HTML, such as custom TouchControl scripting, global and state variables, executing macros, link buttons, etc., along with executing all other primary button types. Please see [this page](#) for more information on using interactive web views.

You may also supply raw HTML to a web view (rather than a URL) if you wish, which will be loaded directly into the web view and displayed on your remote activity screen. If the data provided to a web view button STARTS WITH the character "<", it will be treated as raw HTML, otherwise it will be treated as a URL. The following is an example of using raw HTML to load an image and provide custom page styling:

```
<body style="margin:0">  
  
</body>
```

You may also load an image into a web view from a button pack that you have supplied to TouchControl for button backgrounds. To load an image from a button pack, simply use the button pack name and image file name as follows:

```

```

To simplify the use of the web view button with the TouchControl Screen Grabber, there are several substitution strings that you may use within the web view URL or HTML that automatically map to your TouchControl Server's address, port, etc. The substitution values available are:

%touchcontrolserver/grabber% = resolves to your server's built-in HTTP server address, including the path to the screen grabber service

%touchcontrolserver% = resolves to your server's built-in HTTP server address (including address & port)

%touchcontrolhost% = resolves to your server's IP address

%touchcontrolport% = resolves to your server's HTTP listener port

Using these substitution values, TouchControl will automatically substitute the correct IP address (or hostname) and port number depending on whether you are in Wi-Fi or WAN coverage, and will automatically update as you roam between networks. If you are currently displaying an activity that contains a web view using these values, you may need to back out of the activity and re-enter it to acquire the correct network-specific address.

Dynamic Variable Substitution

You may also use dynamic variable substitution (i.e. %varname%) in a Web View button's HTML or URL to dynamically pass data to Web Views, or web pages loaded within Web Views via script.

When using raw HTML in a Web View, simply specify any `_global` or `_local` variable name (i.e. `_global.myVar` or `_local.myVar`) within percent signs (i.e. `%myVar%`) within the Web View's HTML, and that string will be replaced with the contents of the referenced variable when the Web View loads, and also any time the Web View is reloaded using "[#]" execution from another button's script.

When specifying the URL of a web page in a Web View, you may pass data to the specified URL via a query string contained within the `_global` or `_local` variable. For example, your URL could be specified like this:

<http://localhost/html/myWebPage?%myQueryString%>

The variable `_global.myQueryString` or `_local.myQueryString` should then contain a valid query string, such as:

"item1=value1&item2=value2&item3=value3", etc.

Your web page can then use its own script to process this query string. Note that using the "http://localhost/html" hostname and path for a web page triggers TouchControl to load the web page from the local device, as discussed here.

Refreshing a Web View

Web Views can also be refreshed, or their content updated programmatically via script. Simply return the following from any script to update a web view:

```
return '[#]mywebview';
```

This will simply refresh the current content in the web view.

```
return '[#]mywebview==http://mynewurl.com';
```

This will load a new page (URL) in the web view.

```
return '[#]mywebview==<body>New webview content</body>';
```

This will load new raw HTML into the web view.

Alter a Web View's Identity

By default, a web view on a given device will present the same identity to web pages as the mobile Safari browser running on that device. That is, a web view within TouchControl running on an iPhone will present itself as an iPhone to web pages, and when on an iPad, will present itself as an iPad. This is performed internally and automatically using the User-agent HTTP header. However, if you would like to alter the identity that a web view presents to web pages loading within it, you can alter the User-agent header as needed. This could be useful, for example, if you wish to present a Web View on an iPad that is sized like an iPhone, and load the iPhone-specific version of a web page or web site in the Web View. (This assumes that the web site presents a different version of its web pages on an iPhone vs. an iPad - Twitter.com is an example of a web site that does this.)

To accomplish this, all Web View buttons have a built-in UserAgent property. To set it, right-click on any Web View button, select "Properties...", and select the "UserAgent" from the Name drop-down list. Then simply enter the new user agent value you would like to use in the Value field, and click the "+" button to add this property to the Web View button. To remove the property, just highlight the property in the list and click the "-" button. Or to change an existing value, highlight the property in the list, which will populate the Value field with the property's value, change the value, and click the "+" button again to update the property with the new value.

Web View Script

Enter any desired JavaScript into the "Script" field, which will be executed when the web view button is referenced from another button's script via the "[#]" return string element. See the [Scripting](#) topic for more information.

URL buttons

URL buttons can be used to launch the web browser on the iOS device to view a configured web page, or to launch other iOS apps on the device that expose a URL scheme. Various sites, such as gadgethacks.com track URL schemes used by iOS apps.

Feedback Client Buttons

Feedback Client buttons allow TouchControl to connect to a TCP endpoint (IP address/port number) on your network and “monitor” that location for feedback messages. Data received over that persistent connection is then processed by the script provided within the button’s configuration. This allows TouchControl to react to changes to your environment in real time - as devices or services are updated by outside sources, rather than only after a command is sent from TouchControl. For example, a Feedback Client button could monitor the IP connection on an AV receiver, updating the volume status in real time as the volume knob is turned manually on the receiver itself. TouchControl can also send commands to the monitored device over that same persistent connection. Any feedback generated from those commands are processed by script attached to the button initiating the command, if it exists, or by the Feedback Client button’s script if no feedback script exists for the button executing the command. This allows you to override the default behavior of the Feedback Button’s script on a per-button basis, if you wish. To create and configure a Feedback Client button:

1. Determine the device you wish to monitor/control. A Feedback Client button may connect to any type of device that exposes a TCP port that accepts incoming connections (typically defined as either EventTrigger or Global Caché in TouchControl Server). If you know that an entry already exists for that device you wish to monitor in Interface Manager within TouchControl Server settings, go to step 5. If no server/device entry exists, proceed with step 2 to create one.
2. Choose Tools – Settings from the TouchControl Server menu, then click the “Interface Manager” button, and then click the “Add New” button to create a new interface entry.
3. Select the type of entry to add. If you have no other need to send commands to this connection via other buttons, select “Feedback Client”, otherwise you may wish to use one of the other device types as described in step 1.

4. Give the entry a unique name, provide the devices IP address and Port (and WAN Port if needed to access via the Internet), and select TCP as the protocol. Click “Save” to save the entry. Click “Save” two more times to exit server settings.
5. On the main TouchControl Server screen, select the device you would like to add the Feedback Client button to and click the “Add” button.
6. Select the “Feedback Client” option in the “Auxiliary Buttons” section, give the button a unique name, and click the “Add” button to generate the new button.
7. Highlight the new button in the buttons list and click “Set Data” (or double-click the button in the list) to open the button configuration panel.
8. Select the device this button will connect to in the “Host” dropdown list and enter the desired feedback script in the provided input field. Click “Save” when you are done. Note for future reference that you may also access the Interface Manager settings from the button configuration panel by clicking the “Manage” button next to the “Host” dropdown list.
9. The new button will display in the buttons list with a type of “CLIENT”. You may now drag and drop the new button to any activity layout.

Feedback Client buttons attempt to connect to their defined remote host after the activity they are contained in has loaded/rendered on your device, including any Auto Exec on Load button you may have configured. You will be alerted to any connection errors at that time. Note that if the device you are connecting to only allows a single connection on the defined port at a time, TouchControl will acquire that connection, and any other remote sources attempting to connect to that same port on the remote device will likely fail. If this is undesirable, you may wish to use a standard EventTrigger button type instead (with no feedback monitoring).

Feedback Client buttons are not supported on Apple Watch.

Feedback Listener Buttons

Feedback Listener buttons allow TouchControl to open a local UDP port on your iOS device that waits for UDP messages from any other device/service on the network. Data received on the UDP port is then processed by the script provided within the button’s configuration. This allows TouchControl to react to changes to your environment in real time - as devices or services are updated by outside sources, rather than only after a command is sent from TouchControl. For example, a Feedback Listener button could receive broadcast messages from an automation system indicating that a light has been turned on or off, or that the brightness level has changed, updating the light’s status in real time on your remote control screen. To create and configure a Feedback Client button:

1. Choose Tools – Settings from the TouchControl Server menu, then click the “Interface Manager” button, and then click the “Add New” button to create a new server/device entry.
2. Select the “Feedback Listener” type and give the entry a unique name.
3. If your listener needs to join a multicast group, enter the multicast IP address in the supplied input field. This is optional, and by default (without the multicast address), the listener will automatically respond to any data broadcast on the specified port.
4. Specify the port to which the UDP data will be broadcast. No WAN port needed. The protocol is automatically set to UDP.
5. Click “Save” to save the entry. Click “Save” two more times to exit server settings.
6. On the main TouchControl Server screen, select the device you would like to add the Feedback Listener button to and click the “Add” button.
7. Select the “Feedback Listener” option in the “Auxiliary Buttons” section, give the button a unique name, and click the “Add” button to generate the new button.
8. Highlight the new button in the buttons list and click “Set Data” (or double-click the button in the list) to open the button configuration panel.
9. Select the “Feedback Listener” device created in step 1 above in the “Host” dropdown list, and enter the desired feedback script in the provided input field. Click “Save” when you are done. Note for future reference that you may also access the Interface Manager settings from the button configuration panel by clicking the “Manage” button next to the “Host” dropdown list.

The new button will display in the buttons list with a type of “LISTEN”. You may now drag and drop the new button to any activity layout.

Feedback Listener buttons will begin listening for UDP messages after the activity they are contained in has loaded/rendered on your device, including any AutoExec on Load button you may have configured. Note that TouchControl cannot send commands over a Feedback Listener connection, as it only receives and processes incoming UDP messages. Use an EventTrigger button with a Host that is defined with the UDP protocol to send UDP messages and broadcasts.

Feedback Listener buttons are not supported on Apple Watch.

Script Buttons

Script buttons allow TouchControl to run a block of script, without connecting to a remote device or sending any command. Use this button type when you simply need to run script

locally within TouchControl on your iOS device. Script buttons may be used just like any other primary button type (i.e. added to macros, set as autoexec, given alternate text, etc.). See the [Scripting](#) and [Advanced Scripting](#) topics for more information.

Group Buttons

Group buttons allow you to drag and drop other buttons on top of them, creating a collection of buttons that can be positioned, copied, pasted, and configured as a single entity. To create a group button, simply add a new button to any device and select the "Group" option, provide a button name, and click "Save". When dragging and dropping the group button onto a layout, you may select any background image (or hotspot/no image) for the group, just as with other button types. Groups have a dashed border on the layout designer so you can easily tell them from other normal buttons. Group buttons may not contain other groups, but they may contain any other type of button available in TouchControl, and only one instance of any group button may exist at any given time on an activity layout.

TouchMotion

Groups may also be moved around your iOS device's screen using your finger. While designing your group button in TC Server, right-click on the group and select "TouchMotion", and then select the direction(s) you'd like to be able to move the group on your device's screen. Selecting either "Vertical" or "Horizontal" will allow movement in only the selected direction. Selecting both "Vertical" and "Horizontal" will allow movement in only one of those two directions at any given time - as long as your finger is touching the screen, and then movement in the opposite direction with a subsequent touch. Selecting "All directions" will allow free movement anywhere on the screen. You may also select "Constrained" to prevent the group from being moved off the edge of the layout. Otherwise, the group may be moved off the layout to the extent allowed by the touch recognition. When using "Vertical" and/or "Horizontal" modes, you may also "flick" the group. A short, flicking motion with your finger will send the group sliding to the opposite edge of the activity, stopping when it reaches the opposite edge. Note that TouchMotion movement is only possible when touching the group itself (the group's background). If your group is completely covered with buttons, leaving none of the group itself showing, you will not be able to move the group with your finger.

You may also specify if the group should be "pinned" to the edges of your layout. This allows you to create interactions where the group that you are moving always snaps to the left, right, top or bottom of your layout when you move it. The distance the group needs to be moved before it snaps to the opposite edge in the direction of the move is based on the size of both the group and the layout itself. Moving just a little and then releasing will snap the group back to where it started. Move it a little more and it will snap to the opposite edge. If "Constrained" move is turned off, you can snap the group off the edge of your layout, leaving enough of the group visible to move it back into the layout. The "Pinned" option is only available when you

have selected horizontal and/or vertical movement (i.e. not available when "All directions" is selected).

A group button can also have "TouchMotion Script", including pre-script, move script, and post-script.

TouchMotion "pre-script" runs immediately when you touch the group with your finger, before the move process has started.

TouchMotion "move script" runs as the group is moved across the screen with your finger (or when the group ends its movement after a "flick").

TouchMotion "post-script" runs when you remove your finger from the group after a move, or if the touch/move process is interrupted/cancelled by any other factor (such as dragging your finger off the screen, a pop-up message, etc.).

Select a group button in the buttons list on the main TouchControl Server screen and click "Set Data" (or double-click on the button in the list) and enter your script in the fields provided. The following additional script variables are available to TouchMotion scripts:

<code>_moveDirection</code>	"left", "right", "up", "down", "all", or "none"
<code>_moveDelta</code> script	the distance the group was moved since the last run of the TouchMotion
<code>_groupX</code>	the screen pixel coordinate of the left edge of the group
<code>_groupY</code>	the screen pixel coordinate of the top edge of the group
<code>_groupW</code>	the width of the group in pixels
<code>_groupH</code>	the height of the group in pixels
<code>_groupTouchX</code>	the horizontal pixel location within the group that you touched to move it
<code>_groupTouchY</code>	the vertical pixel location within the group that you touched to move it

- A group configured to move in "All directions" will always return a `_moveDirection` of "all" if it was moved in any direction
- A `_moveDirection` of "none" is returned if a group is tapped but not moved
- Together `_groupX` and `_groupY` make up the point on the screen representing the upper left corner of the group
- Together `_groupW` and `_groupH` make up the size of the group

- Together `_groupTouchX` and `_GroupTouchY` make up the point in the group you touched

Note that if the script that you enter here generates excessive processing demands (such as executing other buttons that send commands over the network that expect feedback, etc.), it could affect the smoothness of movement of the group on the screen.

Here are a couple of videos showing both group buttons and animations...

[Groups & animations - iPad](#)

[Groups & animations - iPhone](#)

Download these sample activities from the download page.

Group Edit Mode

While designing an activity layout that contains groups, pressing and holding the "G" key ("g" for "group") will temporarily hide the contents of all groups on the layout, displaying only the group's outline, background and group name. This is useful if your group is completely covered by one or more buttons, leaving no background available to access the group's right-click menu, or to move/resize the group itself with the mouse. Releasing the "G" key will re-display the contents of all groups. If holding the "G" key down proves problematic on your computer, or you prefer not to have to hold the "G" key for group edit mode, enable the "Toggle Group Edit" option in server setting to switch the "G" key behavior to a toggle, turning on group edit mode with one press of the "G" key, and turning it off with a subsequent press of the "G" key.

Hide In Designer

Groups may also be hidden in the designer during normal design and configuration by right-clicking on a group and selecting "Hide In Designer". This is useful if you have groups that will sit over other groups on your layout during normal use (for example if a group will be initially hidden, and then shown as a popup over your main layout), but you'd like to have access to the underlying buttons for design & configuration purposes. Note that this setting only affects the groups visibility while designing in the server, and has no affect at run time on the device. The normal "enabled" setting still affects run-time visibility, or any alpha (transparency) value applied via script. Once a group is hidden in the designer, using the "group edit" mode described above will temporarily display the hidden group, allowing you to move/resize the group, and/or providing access to the group's right-click menu to configure the group or un-hide it in the designer.

Templates

Any group may also be saved as a template, allowing you to easily create other identical groups in a single step. To create a group template, add a group to any layout, add the desired buttons to the group, configure the group and all contained buttons as desired, and then right-click on the group and select "Save Group Template". This will prompt you to provide a template name,

and then click "OK" to save a template of the selected group. The template then becomes available at a global level to create new groups within your configuration. To update a group template, simply make the desired changes to a group and again select "Save Group Template" from the pop-up menu, and then select the existing template name from the drop-down list and click "OK". The template with that name will now be updated to the current configuration of the group you saved it from.

To create a new group from an existing template, simply drag a group button from the buttons list, drop it on your layout, and when you are presented with the list of images to select for the group, the first list of images in the image selector panel will be "Group Templates". Switch to this list to view thumbnails of all the existing group templates, and select the desired template. The new group will be populated with the buttons from the template, and configured accordingly. The new group will initially be located at the coordinates of the original group used to create the template (useful, for example, if you want a group of buttons to show up at exactly the same location on every activity layout), but can then be moved to any location you wish. By default, when creating a group from a template, the new group will be "linked" to the template, and any future updates to the template will be propagated to any groups "linked" to it. This also means that the contents of new groups created from templates are initially "locked", and cannot be modified directly. If you'd like to "un-link" a group from its template, just right-click on the group and de-select the "Linked To Template" option. Once un-linked, the group's contents may be updated, and changes to the group's original template will not affect this groups appearance or configuration. If you'd subsequently like to re-link the group to its original template, just right-click and re-select the "Linked To Template" option, save, close, and re-open the layout, and the group will once again mirror the template. Note that a group may only be linked to the template that it was originally created from. If you wish for a group to use a different template, simply delete the group and re-create it from the new template.

You may view all existing templates in your configuration by selecting "Template Manager" in server settings. A panel will appear showing all existing group names. Double click on a group name to view a thumbnail image of the group, or select a template name and click the "Delete" button to remote the template from your configuration.

Text Fields

Text Field buttons render as multi-line, scrolling text boxes, allowing you to enter/interact with the contained text via the device keyboard, and scroll with your finger. And since they scroll, they can contain more text that would fit on a standard button/label, or even on your device's screen. Text fields can be used for such purposes as maintaining a log of messages sent to/from devices, generating blocks of text to send to destinations on the network, or any other requirement for capturing and/or displaying large amounts of unstructured text.

There are various ways to interact with text fields. The device keyboard is likely the primary way, automatically popping up when you tap inside the text field to give it focus, or using special device commands enabled via Command buttons (discussed later). You may also disable a text field so that the keyboard does not appear when touching the text field, but still allowing you to scroll the contents with your finger, and you may add text to, and retrieve text from a text field via script, with script helper functions (discussed later).

Text fields can have most of the same properties as normal buttons, including text color, text size, background color (when using a hot-spot for the button image), and a background image. Note that for text fields, any specified background image will be tiled behind the text, and will scroll with the text. The default text color is black, and the default background color (when using a hot-spot) is white. You can set the initial text contained in a text field by populating the alternate text field when creating/defining the button. Note that text fields cannot have HTML text, as other buttons can. If you need a large amount of scrollable HTML, use a web view button instead. A text field's initial text can also be set using a property value, just as other buttons, by specifying `_property.propertyName` in the alternate text field.

The following script helper functions are available to interact with text field buttons via script:

<code>_getText('myTextField')</code>	- returns the text contained in a text field
<code>_setText('myTextField','new text')</code> any existing text)	- sets the text contained in a text field (replacing
<code>_appendText('myTextField','new text')</code> text in the field	- appends the new text to the end of the existing
<code>_prependText('myTextField','new text')</code> existing text in the field	- prepends the new text to the beginning of the
<code>_appendLine('myTextField','new text')</code> of the existing text	- appends the new text plus a line feed to the end
<code>_prependLine('myTextField','new text')</code> beginning of the existing text	- prepends the new text plus a line feed to the
<code>_scrollTop('myTextField')</code>	- automatically scroll to the top of the text field
<code>_scrollBottom('myTextField')</code>	- automatically scroll to the bottom of the text field
<code>_setDisabled('myTextField,true/false')</code>	- enables/disables keyboard interaction with the text field (while maintaining scrolling capability)

Note that when using `_setText`, `_appendText` or `_appendLine`, the text field will automatically scroll to the end of the new text. When using `_prependText` or `_prependLine`, the text field will automatically scroll to the beginning of the new text. The `_append/_prepend` helper functions may also be used with any other button types as well.

The following special device commands can be used with Command buttons to manipulate the iOS keyboard:

{keyboard show:myTextField} - shows the keyboard, placing cursor focus in the specified text field
{keyboard hide} - hides the keyboard

Note that tapping inside a text field will automatically show the keyboard, and tapping outside the text field on the activity background will automatically hide the keyboard.

By default, a text field will present the default iOS keyboard when activated. All text fields have a built-in property named "KeyboardType", allowing you to change the presented keyboard for a specific text field. Simply set the KeyboardType property to the value "Numpad" to display the numeric keypad for the specified text field.

By default, a text field will present a keyboard with a "Return" key, which will add a line feed into the text field when pressed. All text fields have a built-in property named "ReturnKeyType", allowing you to change the name and action of the return key as follows:

- Set the ReturnKeyType property to the value "Done" to present a "Done" key, which will dismiss the keyboard when pressed, as well as run any specified return key script (see below).
- Set the ReturnKeyType property to one of the values "Go", "Next", or "Send" to present a key with the same name, which will run any specified return key script when pressed (see below), but not dismiss the keyboard.

Custom script can also be specified which will run when the return key is pressed (either with the default "Return" key, or any of the above-mentioned custom return key types). To specify script to run when pressing the return key, all text field buttons have a built-in property named "ReturnKeyScript". Simply set the value of this property to the custom script you would like to run when pressing the return key. If executing more than a single line of script, it is suggested to add the script to a function within a script library (Tools - Settings - Script Manager), and simply call the function from the ReturnKeyScript property.

By default, spell check is turned off for text field buttons. To enable spell check, all text field buttons have included a built-in property named "SpellCheck". Simply set this property to the value "On" to enable spell check for the specified text field.

Note that any of the above-mentioned properties can also be set at runtime via script. For example, to turn spell check off, simply execute the script:

```
_setProperty('myTextField', 'SpellCheck', 'Off')
```

Sample activities are available on the [download page](#) which demonstrates several of the features of text field buttons.

- One activity demonstrates using a text field to send character keypresses to a Roku device using the Roku's keypress HTTP API. This feature can be integrated into any activity, and can be used to send text to any device that includes a similar API.
- Another activity creates a chat client using two text fields, a UDP broadcast EventTrigger button, a Feedback Listener, and various other buttons using script/commands mentioned above. To use the chat client sample activity, two devices running TouchControl must be on the same internal LAN network to broadcast and receive the UDP chat messages. The features found in this sample could also be used to send simple alert messages to other devices running TouchControl on your network. Use your imagination, and enjoy!

Designing Layouts

Once you have some buttons created and configured, you are ready to layout your TouchControl screen(s). Click the "Show Layout" button to open the layout designer panel. This is your canvas for designing your universal remote screens. iPhone/iPod layouts are initially sized to match the displayable area on an iPhone, and iPad layouts are initially sized to match the displayable area on an iPad in portrait mode. When initially opening the design panel for an iPad layout in portrait mode, a button will be available to the left of the layout panel that will allow you to switch the layout to landscape mode. The button will then toggle to allow you to switch back to portrait mode. Once you add a background image to the layout (see below), however, the layout will be locked to the dimensions of the background image.

Within TouchControl Server settings you will find an option to "Un-dock designer window". This setting allows the server layout designer window to open as a separate window on your desktop (the default is to expand the main TouchControl window to encompass the designer window). When this is enabled, the layout design window will open each time where it was located the last time it was closed. Only one layout designer window may be open concurrently. Click the "?" in the title bar of the designer (to the left of the "Save" button) to open the "QuickHelp" window, providing you with help for keystroke behaviors, key/mouse combinations and many other advanced designer features. The help window can be positioned in any location on your screen, will remain on top, and will always re-open in its previous location. Note that you must have internet access for the "?" to appear and help content to load.

Background Image

- If you wish, you may add a background image to your layout by clicking the Background button (Windows server), or right-clicking on the layout background and selecting "Background Image" from the popup menu (Mac server). A few backgrounds have been included in the initial install for you to try. If you'd like to add your own background, please see the [Backgrounds And Button Packs](#) page on this site for more information.
- Note that you may add a background of any size. If the background is larger than the default iPhone screen size of 320x480, or the default iPad screen size of 1024x768, the design layout panel will expand to show the entire background.
 - On the server settings screen you will find an option to "Scale layouts in designer." With this option enabled, if the background image is too large to fit on the dimensions of your computer screen at your given resolution, the

image will be scaled down to fit on your screen. This is a design time feature only, and the image will be at full resolution on the iOS device display. If this option is not enabled, the layout designer will expand to the dimensions of your screen and present scroll bars to allow access to the entire layout.

- To add a new background image to TouchControl Server, simply select "Tools -> Import Background Image..." from the menu, and select the image from your computer when prompted. This will add the image to the background collection and make it immediately available for use in your layouts.
- If you already have an activity open in the designer layout panel when you import a background image, and the new background does not show up in the list when attempting to select a background image, you may need to click the "Refresh" button just above the background image list to force the new background image to appear in the list.
- You may also dynamically set the background image used for an activity at run time by supplying a script variable that holds the name of the background image to use. To do this, right-click on a blank spot on your activity background in the server designer and select "Background Variable...". This will present you with an input field in which you may enter the name of a script variable which will hold the background image file name at run time. This can be any variable you choose, as it will be your responsibility to ensure the variable is set via script you provide. When the activity is rendered on your iOS device, it will look for the referenced file within the backgrounds.zip file (so the background image must already exist in the backgrounds.zip archive) and use that image to render the activity. This will occur before the activity is drawn on the iOS device screen, so there will be no visual replacement of images noticeable. You should still add a background image to your activity in the designer as usual, which will be used if the variable does not contain a valid image file name when the activity is opened on the device.

The background variable may also contain a color, allowing you to set the color of the activity background when no background image is being used. This can sometimes be easier than creating a solid color background image just to get a specific color for your activity background. To set a color, populate your background variable with either a hex color value (i.e. '#f2f2f2'), or the name of a built-in color preceded with '#' (i.e. '#black', '#blue', '#cyan', etc.). TouchControl's built-in color names are 'black', 'white', 'blue', 'red', 'green', 'yellow', 'orange', 'purple', 'brown', 'cyan', 'magenta', 'gray', 'lightgray', 'darkgray', and 'clear'. This allows you to quickly change the activity background color by simply updating the background variable's value.

Alternately, if you simply want to set a color for the background and don't need to change it dynamically later, you can just enter the `#color` directly into the "Background Variable" field (e.g. `#black`, `#blue`, `#f2f2f2`, etc.), bypassing the need to create and maintain a separate variable.

Adding Buttons

To add a button to the layout, highlight the desired device in the *Devices In This Activity* list, and click and drag the desired button from the Buttons list to the layout canvas and drop it on the layout panel. On the Mac server, you may also highlight the button you wish to add in the list, and click the "Add button to layout" button at the bottom of the server window, and then click the location on the layout panel where you'd like to add the button. When you add the button to the panel, you will be presented with a list of available images for the button. Button images are grouped into "Button Packs" which are selected via the drop-down list at the top of the image list. Simply click on the desired image and it will be added to the layout panel where you can then drag it into the desired position and re-size as needed. Note that the first image shown in each button pack is actually a "hot spot", which is a transparent button that can be positioned over any portion of the background, can be given a solid background color and/or rounded corners if desired, and can be adjusted to the desired size.

Button Packs: Some default button images are included with the TouchControl Server installation, but the fun starts when you add your own images for your device buttons. You do this by packaging your images into "button packs" and supplying them to the TouchControl program. Please see the [Backgrounds and Button Packs](#) topic for more information.

- If you have recently added a new button pack and the new pack does not show up in the list when attempting to select a button image, you may need to click the "Refresh" button just above the button image list to force the new button pack to appear in the list.

Selecting buttons

To configure a button that has been dropped on a layout, select it by simply clicking it with your mouse.

Multi-select mode: You may also select and configure multiple buttons simultaneously by holding the CTRL key while clicking on the buttons you wish to select. This is referred to as multi-select mode. Each multi-selected button will highlight with a magenta border to indicate it is in multi-select mode. Simply click on a blank spot on your background to deselect the selected group of buttons, or CTRL-click again on any multi-selected button to remove it from the multi-select group.

Drag-select: In addition to using CTRL-click to select multiple buttons, you may also click on any blank spot on your layout background and start dragging. The background will turn black while dragging so you can see the drag rectangle outline regardless of what background image you may be using. Buttons will be selected as soon as the drag rectangle touches them (i.e. buttons don't have to be fully contained in the drag rectangle to get selected). To drag-select multiple buttons inside a group, hold the "s" key (for "select") before starting the drag, and release the "s" key when finished drag-selecting buttons (this is because a group is itself a button, and a normal click-and-drag with the mouse will simply move the group on the screen).

Selecting group buttons: Note that you may not select both a group button and any of its contained buttons at the same time. If you select a group button (using either CTRL-click or the drag-select method), any previously selected buttons within that group will be de-selected. Or if you select one or more buttons within a group (using either of those same methods), if the containing group was previously selected, it will be de-selected.

Cloning buttons

When dragging and dropping buttons onto your layouts, you also have the option to pick an existing button on your layout to "clone" with the new button. On Windows, simply hold down the Ctrl key while adding a button to enter "clone mode", then click on the button you'd like to clone. On macOS, when you drop a button on an activity, it will ask whether you wish to add the button, or clone another button. The new button will take on the cloned button's size, image, icon, and text attributes (but will retain the original button's text, dropped location, and other internal button properties).

Copy & paste

If you wish to duplicate any buttons that have been added to a layout, select one or more buttons in the layout, right-click and select "Copy", then right click on your layout background and click "Paste", and the new buttons will be created at the same location as the copied buttons. Paste buttons into the same layout, or into any other layout you wish.

Resizing buttons

After a button has been dropped onto a design layout and a background image has been selected, you may move the button to any location within the layout using the mouse or arrow keyboard keys, and/or resize the button by dragging the sides or corners of the button image. You may also multi-select multiple buttons and then "grab" any of the selected buttons with your mouse and drag the entire group as a set. You can also "grab" the edge or corner of any of the selected buttons, and move the mouse to resize all the selected buttons together.

Fast resizing: (Windows server only) Normally, clicking and dragging the side or corner of a button or hotspot will increase or decrease its size by one pixel at a time. With large iPad layouts, it can take quite some time to resize a large button or gesture pad. To resize buttons more quickly, press and hold the "Alt" key while dragging the side or corner of a button to force the button to resize by 20 pixels at a time. Release the "Alt" key to return to 1-pixel resize, press it again for 20-pixel resize, etc.

Proportional resizing: Normally when resizing a button or hotspot, it will resize only in the direction you are dragging, losing its original proportions. To resize a button and retain its original proportions, hold the "Shift" key while dragging any straight side of a button or hotspot to resize in two dimensions. In the Mac Server, you may also proportionally resize by dragging button corners. In the Windows server, the "Shift" key has no effect when resizing by dragging the corners of a button/hotspot.

Replacing buttons

If you'd like to replace the button associated with an image on a layout after you've already dropped a button onto the layout background, simply drag a new button and drop it on the old one. You will be asked if you'd like to replace the old button with the new one, and if so, just answer "Yes". You may not drop another button onto a Label or a Gesture Pad, however. To replace those types, you must delete them and add a new button in their place.

Designer transparency

When adding buttons to a layout in TouchControl Server for Windows, button images using a transparent background are transparent to the layout background, however they are not transparent to any buttons beneath them. Therefore, if you overlay any button with another button, it will look as if the top button is blocking the view of the button under it. However, this is only an anomaly in the way the server program (Windows) displays the images, and the result on the iOS device will be truly transparent buttons displaying any other buttons beneath them. Note that the Mac server provides true transparency and therefore does not have this limitation.

Rotating buttons

Buttons can be rotated by holding CTRL+SHIFT while clicking on one or more buttons. This will display the button rotation dialog, which will allow you to freely rotate any selected buttons from -180 to +180 degrees using either the slider control, or by entering the rotation degrees directly into the field provided. Click on a blank spot on your layout background to exit rotation mode.

Configuring Buttons

Right-click menu: After a button has been added to the layout, you may right-click on the button to display a pop-up menu of numerous configuration options for the button. Feel free to play around with the options available to make your button look and act as you desire. You may also test and delete buttons from the layout using this pop-up menu.

Enabling/disabling buttons: The first option on the right-click menu will be "Enabled," which by default will be turned on (checked). An enabled button is visible on the activity when viewed on the client device. Disabling a button by un-checking this option will essentially hide the button when the activity is viewed on the client device. This allows you to add buttons to a layout for various uses that do not require interaction by the user. This includes buttons that will be programmatically executed by other buttons via script, buttons used as auto-exec on load/resume/exit, buttons that you want to initially hide and then show to the user later via script, etc.

Button Images: After adding a button to a layout and selecting its image, you may use the Image menu option to select a new image for the button, or to set various other image options:

Pressed Image: Each button may have an alternate image defined for the button's "pressed" state. This image will display when you tap on or hold the button on the device screen. Right-click on a button in a layout and select Image -> Pressed Image -> Select... to pick the alternate pressed-state image (the pressed image must be from the same button pack as the button's primary image). Or, select the "Auto" option which will briefly invert the primary background image when the button is tapped. Note that the "Auto" option will also invert any text or icon that is embedded in the background image, so use carefully. If no pressed image selection is made, the standard momentary shift of the primary image on the screen will be the default. NOTE: The "Auto" option only available on devices with iOS 4.0 or greater.

Icons: You may also add an "icon" image to any button, which is an additional image placed on top of the main button background image. Once a button has been dropped on a layout and the primary button image has been chosen, right-click on the button in the layout and select

"Image - Icon...". This will display a list of icons, grouped by "icon pack", and selecting an image from the list will place that image in the center of the button on the layout. Several common icon images have been provided (e.g. play, pause, stop, ff, rew, back, etc.), in several different colors/textures.

To add your own custom icons, locate the "buttonicons.zip" file within the "images" folder under your "TouchControl" data folder (under "My Documents" unless you moved it during installation), and either add icon images to existing folders within that zip file, or add new folders to the .zip file and populate them with the desired icons. Each folder within the buttonicons.zip file becomes an "icon pack".

Reset size: Select this option to return selected buttons to their original size based on their background images.

Set background color: For hot-spot buttons (those without a background image), you can set the background color of the button to one of several different pre-defined colors as well as round the button corners for an enhanced visual effect. Clicking the "Custom color" option will open the visual color picker, allowing you to select any color in the palette. Optionally you may hover the mouse over the "Custom..." option (Windows server), and an entry field will display, allowing you to enter any HEX color code (e.g. #1A2B3C or #1A3). This option works for multi-selected buttons as well. A button's background color can also be set dynamically via script using the `_setBackgroundColor()` helper function (see the [Advanced Scripting](#) topic). Note that when using the "Liquid Glass" background color, the iOS glass material works best if there is something interesting behind the button for the glass effect to interact with. Using a custom background image for the activity is recommended in this case.

Disable animation: When a button is pressed on the device screen, by default the button image will momentarily shift down and to the right, and then immediately back into place. If you don't want this animation to occur, you can turn off this behavior by selecting "Image - Disable Animation" in the button's right-click menu. You may wish to do this if you are creating your own animation via feedback scripting, for example.

Button text: Use the "Text" menu option to alter the display of text on a button.

Show/hide text: By default, a button's text is visible on the button for buttons originally added with a background image, and not visible for buttons originally added as a hot-spot. Use the Show text and Hide text options to change the button's text visibility.

Text size: By default, a button is added to the layout with the pre-defined "medium" sized text. A button's text size can be altered using this option to various other pre-defined sizes. A button's text can be changed to virtually any size at run time using the `_setTextSize()` script helper function (see the [Advanced Scripting](#) topic).

Text color: By default, a button is added to the layout with black text. The text color can be changed with this option to one of several pre-defined colors, or to virtually any color using the "Custom color" option. Clicking the "Custom color" option will open the visual color picker, allowing you to select any color in the palette. Optionally you may hover the mouse over the "Custom..." option, and an entry field will display, allowing you to enter any HEX color code (e.g. #1A2B3C or #1A3). This option works for multi-selected buttons as well. A button's text can also be changed to any color at run time using the `_setTextColor()` script helper function (see the [Advanced Scripting](#) topic).

Text font and alignment: Special built-in button properties are provided for all button types that display text that allow you to adjust the font, size and alignment of the text displayed for the button. See [Built-In Button Properties](#) for more information.

TouchTips: Buttons may also display a "TouchTip" on the iOS device screen when touching or swiping across the button using this option. TouchTips will enable the equivalent of a "tool tip" or "bubble" above your finger when pressing and holding on a button on your iOS device screen, displaying the button's text within the bubble. Release the button to execute the command, or slide your finger off the button to cancel execution (when the tip disappears). Or, simply slide your finger over multiple TouchTip-enabled buttons to display their tips as you slide, then release when you find the button you want to execute (while its tip is showing). This is only available for buttons that have their text hidden (since the button's text is what displays in the bubble), and is not available for repeating buttons, press & release buttons, gesture pads, sliders, spinners, or web views - so just

standard, single-tap buttons. This is useful if you have several buttons that only display images/icons, and which may be difficult for a user to identify (such as a page of channel favorites). Simply sliding your finger across the buttons displays the relevant text, without taking up the extra room on the layout to permanently display the text.

Size & Location: (Windows server only) For an additional method to move/re-size buttons, this option opens a panel which will allow you to enter/select the left, top, width, and height of the selected button. This is also available for multiple selected buttons. This can be useful when making precise layout changes rather than trying to make many small size/position changes with the mouse.

Delay touch: (Windows server only) Setting button delay allows you to click multiple buttons on the iPhone/iPad screen before the commands are actually sent to the device. This is useful for buttons such as channel numbers, allowing you time to touch all numbers in a given channel, and then have the entire channel signal sent to the device (TV, DVR, etc.) at once, effectively creating a dynamic macro. This should help for those devices that automatically change channels after a short amount of time when sent a channel number, for example.

Auto Exec: A button may be set to automatically execute on load, on appear, on resume, or on exit of an activity. When you right-click on a button in your layout, use the Auto Exec option to set this functionality. Only one button may be selected at a time for on load, on appear, on resume, or on exit, but you may have one of each on a layout, and a single button may be selected for any/all of them. If you'd like to select a button for auto execution that does not display on your layout, simply add the button, disable it, and drag it off into a corner of the layout out of the way.

Be aware of the difference between the on load and on appear button behavior. On load buttons execute only once when an activity first loads. If you link to another activity and then return back to the first activity, an on-load button on that first activity will not re-execute, as that activity still exists and does not re-load when returning. On appear buttons, however, do execute each time an activity appears. This includes immediately after the initial load, and any time you link/navigate away and then link/return back to

the activity. So, you should place any command/script that you only want to execute once in the lifetime of an activity in an on-load button, and any command/script that you want to execute each time an activity is shown in an on appear button. Note that on resume buttons only execute when TouchControl is returned to the foreground after being sent to the background, or the device goes to sleep and then resumes. In those cases, whatever activity is currently being shown will execute its on resume button.

Layout: Several layout options are available for single or multi-selected buttons:

Bring to front/Send to back: Use these options to arrange the layers of overlapping buttons on your layout.

Center horizontally/Center vertically: Use these options to center buttons within their container (either within the activity or within a group).

Align: When multiple buttons are selected together, you will have the option to align those multi-selected buttons to each other. Select the "Tops", "Bottoms", "Lefts" or "Rights" option to align all selected buttons to the desired edge of the button that you right-clicked on to select this menu option.

Stationary: Enabling this option for a button allows it to remain stationary (or "locked") on the device screen while the rest of the activity scrolls beneath/behind it. This is, of course, only useful on activities that scroll, as all buttons are in effect stationary if the activity does not scroll either horizontally or vertically.

Propagate: (Windows server only) You may also right-click on any button in the layout designer and select "Propagate", and you will be asked to select another button (or buttons) to "propagate" the currently selected button's visual properties to. This is similar to the "clone button" option above, but where clone is used to style a single, new button when you drop it on your layout, the "Propagate" option allows you to propagate visual properties from button to button after they have already been added to the layout. So, if you decide you want to change the image on several buttons on your layout to make them all the same, all you need to do is set the image on one button, then propagate the style of that button to all the other buttons. Simply multi-select as many buttons as you wish, then pick one of the selected buttons when prompted.

Properties: You may set custom properties for any button. Please see the [Advanced](#)

[Scripting](#) topic for more information.

Design-Time Features

While designing your remote interface, right-click on a blank spot on your remote background to be presented with a menu which will allow you to:

Snap to grid: This will cause the buttons and hotspots you add to your remote layout to snap to a grid as you drag them around. The grid size is set in server settings as outlined in the [server setup](#) topic. Use this to help you line up buttons more easily. You can still use the arrow keys for more granular movement when this option is enabled, and you can turn this option on and off as needed while designing a remote screen.

Undo all changes: (Windows server only) This will undo all current changes since the last time the layout was saved.

Find: (Mac server only) On the Mac server, the layout designer right-click menu includes a "Find" option that will allow you to quickly locate any auto-exec buttons (on load, appear, resume, exit). If one of these buttons is found on the layout, it will highlight and flash.

Undo button: (Mac server only) On the Mac server, the layout designer includes an "Undo" button in the upper right. Any changes made to button size or location can be un-done using the undo button. If multiple buttons are moved or resized using multi-select, those changes are un-done simultaneously as well. Other button property changes made using the button popup menu are not un-done using the undo button.

Button border color: This will allow you to alter the highlight color of the borders (outlines) around the buttons on your remote layout. This is useful for dark backgrounds or backgrounds with colors that blend with the button borders making it difficult to see the outlines for precise placement of buttons. Please note: These borders are only visible while designing the remote layouts, and not visible on the device display.

NOTE: If you'd like to get a better idea of what your layout will look like without the outlines in the layout designer, just press and hold the "O" key (that's "o" for "outline"), and the outlines will temporarily disappear. Release the "O" key to bring them back.

Background variable: This option allows you to specify a script variable that contains the name of the activity background image or color value that you'd like to use. This allows you to dynamically set or change the activity background at runtime via script by changing the contents of the variable you specify here.

While designing your layouts, if the designer panel is open, each time you select a button in the buttons list, the corresponding button will be highlighted within the layout designer, if it is found there. The button on the layout will be temporarily brought to the top (if it is partially or fully hidden behind other buttons), its border will be outlined with the cyan color, and it will momentarily flash to highlight its location. When you either select a different button in the buttons list, or click any other button in the layout, or click the layout background, the highlighted button will be returned to its original location (depth) and its border will return to the normal layout button border color.

Alternately, when you click on a button that has been added (dragged/dropped) to an activity layout, the corresponding button will be selected in the "Buttons for selected device" list, and the containing device will be automatically selected in the "Devices in this activity" list.

Quick drag (Windows server only)

When moving or re-sizing a button, especially a group with many contained buttons, screen re-draw can degrade the performance of the move/re-size action. To help with this, simply hold the "D" key (that's "d" for "drag") while dragging for "quick drag" mode, and the button/group contents will blank out, leaving just the outline and any background image, showing the new position/size as you drag. Releasing the "d" key while still dragging will re-display the button contents, or releasing the mouse button will re-display the contents when done. "Quick drag" will also help smooth out the process when simply moving several buttons at the same time. So, after you multi-select some buttons (either using CTRL-click, or the new drag-select method), simply hold the "D" key before you start dragging (or while you're dragging if you forget), and the movement of the buttons on the screen will smooth out. This feature is not needed on Mac server.

Designer Hotkeys (Windows server only)

You may use the following hotkeys while working with buttons/layouts in the server designer:

Ctrl-a (select all)

Ctrl-c (copy)

Ctrl-x (cut)
Ctrl-v (paste)
Ctrl-s (save layout)

Saving Layouts

Once you have the layout looking the way you want, you can save the layout to the configuration by clicking the Save Layout button at the top of the layout panel, or pressing Ctrl-s. You may also click the "Hide Layout" button, which will prompt you to save the layout if any un-saved changes have been made.

Interface Manager

Interface Manager is available within TouchControl Server settings to manage the network settings for the remote interfaces (e.g. IP enabled devices, servers such as EventGhost, and devices such as Global Caché iTach & GC-100) that you can control with TouchControl. All remote interfaces that you wish to control must be added via this feature before they will be available to use elsewhere in the app.

To open Interface Manager, select "Settings" from the "Tools" menu, and click the "Interface Manager" button to open the "Interfaces Hosts" configuration panel. This feature can also be accessed from the button configuration panels using the "Manage" button found there.

Once the "Interface Hosts" panel is open, to add a new device or server:

1. Click the "Add New" button. The "New Interface Host Settings" panel will display.
2. Select the type of interface you wish to add.
3. Give the interface a unique name. This will be the name that you will reference this interface by in all other locations in TouchControl Server.
4. Enter the "IP Address" for the interface.
 - a. For the Feedback Listener device type, the IP address field will become a "Multicast Group" field. If your listener needs to join a multicast group, enter the IP address of the group in this field. If not joining a multicast group, simply leave this field blank (for Feedback Listeners).
 - b. You can also dynamically set the IP address of a connection at run-time via script. To do this, set the IP address of the connection to: %myVarName% (where "myVarName" is a local or global variable set using `_local.myVarName='192.168.1.120'`; or `_global.myVarName='192.168.1.120'`; for example). When TouchControl attempts to connect using this interface, it will use the value of the variable as the connection's IP address, allowing you to change the IP address at run-time, if desired. Note that when dynamically changing a connection's IP address, any existing connection to the old device must be terminated so that a new connection can be established to the new device at the new address, so make sure you use execute a button (an auto-exec on load button would come in handy for this) that specifies the Interface Manager entry as its "Host", and returns "[!]" from its pre- or post-script to kill the connection to the previous device. Then the first time a button is tapped to send a command to the device, it will use the new IP address specified by the global variable.

5. Enter or select the "Port" used by this interface. For example, all Global Caché devices use port 4998 for IR control.
6. Enter or select the "WAN Port" used by this interface. This will be the "external" port that you have configured in your network to allow access from the Internet. This allows you to open one port in your router, and forward the requests to a different internal port (the "Port" specified for this device). This would be needed, for example, because all GC devices use port 4998 to listen for requests, so different ports would need to be forwarded from your router to port 4998 at different IP addresses if you have more than one GC device that you wish to access from the internet. If you don't plan to access this interface from the internet, the actual port number entered here is irrelevant.
7. For HTTP Request interfaces, you may specify basic authentication (ID & password) settings within the interface settings. This allows you to specify authentication for all buttons that use a given interface in one place. Authentication settings may also be specified for each individual HTTP Request button by setting the HTTPAuth property separately on each button, however, specifying authentication settings here streamlines the process when all buttons that use a given HTTP Request interface need the same authentication settings. You can also use the HTTPAuth property on any given button to override the authentication setting specified in that button's interface host. When setting authentication within the interface host, you may also specify only a password (leave the ID field blank) for those devices/services that require only a password with no ID.
8. If you have devices which only allow a single connection, such as the Global Caché GC-100 adapters, but you have multiple iOS devices running TouchControl, you can "proxy" all requests for a given host from multiple iOS devices through TouchControl Server, effectively making TC Server that single connection, while providing your devices full two-way communication with the GC-100 (or whatever device you are using). To enable this, simply check the "Proxy internal requests" option when configuring the Interface Manager entry for the device you wish to control. This setting is available for EventTrigger, Global Caché, and iRTrans interface definitions.
9. If your router does not support forwarding one external port number to a different internal port number (as described above), and you have multiple devices that use the same port, you can use TouchControl Server as a "proxy" to receive the device commands and forward them on to the target device, and return any feedback received to TouchControl on your device. This is especially useful if you have multiple Global Caché IR or contact closure devices which all use port 4998 (and cannot be changed). Simply check the "Proxy external requests" option beneath the port assignments when setting the port values to enable this feature. This setting is available for EventTrigger, Global Caché, and iRTrans interface definitions.

10. Select the protocol, TCP or UDP, to use when communicating with the remote host by selecting the appropriate radio button. This option is only available for EventTrigger server/device types. Other server/device types will automatically set the protocol as needed.
11. You may also set a built-in "heartbeat" for any TCP connection. A heartbeat simply sends some small amount of data over the connection at a pre-defined interval. If you are having problems with a particular device dropping the connection after some period of inactivity, or if a device is designed to terminate the connection after some period of inactivity, adding a heartbeat can help keep the connection "alive" and avoid delays or errors when TouchControl attempts to use a connection that has been broken or dropped on the other end. To enable the heartbeat, when creating or updating an entry in Interface Manager, simply check the "Heartbeat" option at the bottom of the panel, provide an interval for the heartbeat to send its data, and provide some small amount of data to be sent at each interval. Normally something as simple as a carriage return (\r) is enough to keep the connection alive, or your device may require more or different data. It would be suggested to use the minimal amount of data necessary, and make sure that the data you send does not trigger unwanted actions in the remote device, or generate unwanted feedback to TouchControl.
12. You may also enable the "Bypass" option to prevent the iOS device from attempting to connect to the remote host when using buttons associated with this interface, and optionally supply static feedback for buttons that require feedback. This is useful for testing purposes when the remote host may not be available. Remember to turn this feature off when you are ready to actually use the remote host, or deploy the configuration to a user/customer.
13. Click "Save" to save this information and return to the server/device list panel.

Click the check box to the right of an interface in the list to set it as the default for that type. When adding new buttons, the default interface for that button type will initially be selected, and can be changed by simply selecting a different host when configuring the button action (if more than one interface for that button type has been added to the configuration).

To edit an existing interface in the list, right-click an entry and click "Edit".

To duplicate an existing interface, right-click an entry and select "Copy".

To delete an existing interface, right-click an entry and select "Delete".

Any time you add an interface on your network that you wish to control using TouchControl (e.g. Global Caché, IP devices, EventGhost, HTTP servers, etc.), simply return to the "Interface Manager" and add the new host before adding buttons to control it.

Template Manager

In TouchControl Server settings, click the "Template Manager" button to open the Template Manager panel, which will display all currently configured group templates. Double click on any template name to display a thumbnail image of the group as it was configured for the template. Highlighting a template name and clicking the "Delete" button will remove the selected template from your configuration (but not any groups created from that template). When opening an activity layout that contains a group linked to a removed template, you will be given the opportunity to un-link the group(s) from the template. You may also re-create a template with the same name as a removed template, and any groups that were linked to the old template will be automatically linked to the new template.

[Server Tools](#)

[PC Remote Control \(Windows server only\)](#)

Using only the free TouchControl server software, you can turn your PC into a universal remote control using the "Remote Control" option on the TouchControl Server menu. This will launch your remote layouts directly onto your PC desktop. [Please see this page for more information.](#)

[Listening \(Windows server only\)](#)

When TouchControl Server starts, it will automatically start listening for connections from the device app. If, for any reason, you wish to disable the server program from listening while it is running, select the Tools > Stop Listening menu option to disable listening. Then click Tools > Start Listening when you want to start listening for request once again.

[Import/Export](#)

You can import and export remote layout settings using the Import Configuration and Export Configuration options from the Tools menu. You can use these functions to copy remote layouts from one computer to another, to share with someone else, or to back up your configuration and restore later.

- To export/backup your entire configuration, choose Tools > Export Configuration > Entire Configuration from the menu. This will automatically save the entire configuration to a file in your TouchControl Server data directory (a popup message will let you know exactly where it was saved, including the file name).
- To export only one activity to a file, choose Tools > Export Configuration > Activities, and you will be presented with a dialog allowing you to select the activity you wish to export. You also have the option of including the activity background image, any referenced button packs, and any referenced external script files in the export zip file.
- To import previously exported data (either an entire configuration or a single activity), choose Tools > Import Configuration..., and you will be prompted to select the import file from your hard drive. Once you've selected the file and provided the requested information on the import panel, the configuration will be added to your setup. You also have the option of importing any background image and/or buttons packs which were included in the export file.

[Reload Configuration](#)

Use the Reload Configuration menu option to force TouchControl Server to reload its config from the XML files on disk. This would be useful if you wanted to swap out the XML configuration files (TouchControlConfig.xml and TouchControlMasterConfig.xml), and then load the new configuration from those files without stopping and restarting the server.

Migrate Activity Buttons (Windows server only)

Have you ever needed to swap out the buttons from one device on and activity for the same buttons from another device? For example, if you want to change the buttons on your "Watch TV" activity from controlling a cable box to controlling a satellite receiver, or vice versa? Or maybe wanted to change the control of a device from USB-UIRT to Global Caché control, but keep the same buttons and layout? That process can now be much easier with the "Migrate Activity Buttons" option on the server Tools menu. All you need to do is create your new device (most likely by making a copy of the existing device), then updating the buttons with the new commands as needed, and then select Tools - Migrate Activity Buttons... from the server menu. This will display a dialog that will allow you to select the activity or activities that you'd like to update, pick the device you are migrating from, and the device you are migrating to, and click "Migrate". You may also specify that you'd like to update all macros in your configuration to replace any references to buttons from the old device with references to the same buttons in the new device. This can be a major time saver, rather than having to go through and update macros manually (think about all of your favorite channel macros!). Give it a try the next time you're making these types of updates to your configuration.

Arrange Locations & Activities (Windows server only)

The default order for both locations and activities in both the server interface and TouchControl on the device is alphabetical. If you would like to arrange the locations/activities in a different order of your choosing, the "Tools -> Arrange Locations & Activities" menu option will present an interface which allows you to arrange the locations, and the activities within each location, into a custom ordering. Simply select a location or activity in the list and use the arrow up/down buttons to move the selected item into place, and then click the "Save" button to save the new arrangement. Then refresh your configuration to your device (using the refresh button in the upper-right nav bar of the main activities screen on the client) to update the arrangement there. Please note that locations are arranged relative to other locations, and activities are arranged relative to other activities within the same location. Therefore, you cannot inter-mingle activities from multiple locations within the resulting arrangement.

Import Global Caché Buttons From iLearn (Windows server only)

Available from the TouchControl Server Tools menu, this feature allows you to use files created by Global Caché's iLearn program to import IR codes into TouchControl Server in bulk. First select the TouchControl Server device that you'd like to import the buttons into from the "Available Devices" list, then select "Tools – Import GC Buttons From iLearn" from the TouchControl Server menu. After you choose the iLearn import file from your computer's file system, you will be presented with a panel that will allow you to select the "Host" GC device to configure for all buttons (from the GC devices available in Interface Manager), and also the module and connector on that GC device that you'd like to configure for each button. Click "Import" and all codes in the file will become buttons in TouchControl Server, immediately

available for you to add to your activity layouts. Note that the names of the newly created buttons are derived from the names provided for each code within the iLearn import file.

Import Global Caché Buttons From Database

Available from the TouchControl Server Tools menu, this feature allows you to access Global Caché's online IR code database ("ControlTower") to import ready-to-use IR codes. When selecting this option, you will be presented with a panel which will allow you to search through the more than 138,000 code sets freely available in the online database, in an easy-to-use interface. First select the TouchControl device that you wish to add the newly created buttons to, then select the brand of the device you wish to control, and then select the device type and then model name to see a list of available functions for the selected device. You can then select all or a subset of those functions to automatically import the related IR codes into TouchControl. Set the "Host", "Module", and "Connector" settings on this panel to the proper settings for the device you wish to control with these codes. You may also select the "Dynamic" option to use script variables to dynamically specify the module and connector at run-time (see the [Global Caché button documentation](#) for more information). The import process will then create a new button in TouchControl for each function you select, automatically populating the buttons with the IR codes from the online database, as well as the host, module, and connector that you specified.

The newly imported buttons will, by default, be named the same as the functions shown for each IR code. Prior to import, click the "Normalize" button on this panel to see a list of optional adjustments that can be made to the button names during the import process. You can strip certain strings from the resulting button names, and/or make certain string replacements. Browse the list of functions shown and see if any of the normalizations look appealing to you. They are completely optional, and any selected normalizations occur in the order shown. Once the import is complete, click the "Done" button and then select the TouchControl device that you chose for the import to see the newly created buttons. Note that if you import buttons into a device that contains existing buttons, any new buttons generated during the import that would result in duplicate button names are skipped, and you are alerted at the end of the import process as to which buttons were skipped (not imported).

Find Unused Buttons (Windows server only)

The "Find Unused Buttons" feature will display a list of all buttons that are not currently in use in your configuration. This feature presents two options for finding unused buttons: "direct" and "recursive". The direct option will find any buttons that are not used directly in an activity, and which are not referenced directly in a composite button (i.e. a macro, gesture pad, slider or spinner). The recursive option finds those same unused buttons, but in addition, it will also report buttons as unused if they are included in a composite button, but that composite button is not included in an activity or in another composite button. No buttons are removed/deleted

using this feature - you are simply presented with a list of unused buttons that you can use as you wish.

Checking for New TouchControl Server Version (Windows server only)

You can use the Check for New Version option on the Help menu to force TouchControl Server to check to see if a new version of the software is available on the web site (requires internet connection). This check is also done by default each time the program is started if you have enabled that option in settings. If a new version is available, you will be prompted to automatically navigate to the download page on this site via your default browser to download the new version. **IMPORTANT:** If you'd rather not automatically perform the check for a new version at startup (for example if you run TouchControl Server in a lights-out environment and do not wish to be prompted at startup in the event a new version is available), you can turn off automatic version checking on the settings page (Tools > Settings). In this case you should periodically perform the version check manually, either by using the Help menu option, or by visiting this site.

TouchControl Server configuration backup/recovery (Windows server only)

Each time a change is made to your configuration (adding/removing elements, configuring activities/buttons, modifying layouts, etc.), a snapshot of your configuration is saved to the "backup" subdirectory under the TouchControl data directory on your computer (typically "\\my documents\TouchControl"). In the event that your TouchControl Server configuration becomes corrupt or unusable, you may recover from the backup by stopping TouchControl Server and copying the two files located in the backup directory into your TouchControl data directory, replacing the files located there with the same names.

[Backgrounds and Button Packs](#)

Custom background and button images are one feature that really set TouchControl apart from other iPhone remote apps. You can supply TouchControl with any images you'd like to use on your remote screens.

Background Images

Background images are stored in the backgrounds.zip file located in your TouchControl images directory (default is My Documents/TouchControl/images). A few sample backgrounds are included with the TouchControl installation. To add your own background image(s) (or images you've downloaded from the [Download](#) page), simply add the image(s) to the backgrounds.zip file using your favorite ZIP archive program (such as WinZip, 7zip, WinRar, Windows compressed folders etc.). After you have added the image(s), restart TouchControl Server, and the new image(s) should be available when you click the Background button above the layout panel.

To add a new background image to TouchControl Server, simply select "Tools -> Import Background Image..." from the TouchControl Server menu and select the image from your computer when prompted. This will add the image to the background collection and make it immediately available for use in your layouts. If you download background images from the Download page, simply uncompress the zip file to your computer and use the import option to import the background(s) into TouchControl Server.

You may also upload images from your device directly into the backgrounds collection. Find the option in Settings in the TouchControl app.

Background images should be at least 320px wide by 420px high to fill the available iPhone display (320px by 480px if using full-screen activities on the iPhone/iPod). Images may be larger, in which case the iPhone display will scroll to allow access to the entire background. The layout design panel, however, will expand to show the entire background to make it easier to layout your remote screen(s). Note that if the background image is too large to fit on your screen at your given resolution, the image will be scaled down to fit within your screen dimensions. This is only a design-time feature, and the background image will be at full resolution on the iPhone screen.

Providing an image that is wider than 320px or taller than 420px is a great way to give you extra space to layout buttons by taking advantage of the iPhone's excellent scrolling motion. Just put less frequently used buttons at the right or bottom, and swipe your finger to scroll them into view when needed.

You may also add high-res background images for use on devices with the high-res retina display. Following Apple's iPhone image naming standard, images that include "@2x" at the end of the filename (i.e. mybackground@2x.png) will be treated as high-res, and will be sized to half the dimensions of the raw image, effectively giving it twice the pixel density of normal images. If you add two images with the same name, but one has the additional "@2x" added to the filename, the iOS devices will automatically use the high-res image on devices with a retina display and the lower-res image on all other devices. This allows you to build one layout and automatically support devices with different screen resolutions.

When adding your own button images to a layout, you'll probably want to keep the background image fairly simple so that it doesn't distract from the buttons themselves. However, using the button hot spot, you can use elements in the background image as buttons themselves (for example, a picture of your device's remote control), which can make for fun and interesting remote control screens. So play around and have fun!

Button Packs

Button images are provided to the TouchControl program in "button packs", which are simply ZIP files with a specific name, location, and file structure. Two sample button packs are included with the TouchControl installation. To create and add your own button packs, follow these simple steps:

- Create your buttons using your favorite image design program. Images are preferred in the .png format and should use background transparency for best results. Create a separate image file for each button. It is suggested that button images be no smaller than 30x30px to be usable on the iPhone screen. 60x60px is a nice size to hit with your finger.
- You can alternately obtain button images from any source available to you. Images can be in the .png or .jpg format, but, again, the .png format is preferred for best results as they support transparency. <http://www.remotecentral.com> contains files with hundreds of great remote control button images. Just please make sure you honor all copyrights and license agreements when obtaining images from any other location.
- If creating your own images, add your images to a directory on your computer named the same as the desired name of your button pack.
- Each button may also have an alternate image defined for the button's "pressed" state, which will display when you tap on or hold the button on your device screen. Pressed-state images must reside in the same button pack as the primary image for the button. See the [Designing Layouts](#) topic for more info on adding pressed-state images.
- You may also add high-res button images for use on devices with the high-res retina display. Following Apple's iPhone image naming standard, images that include "@2x" at the end of the filename (i.e. mybutton@2x.png) will be treated as high-res, and will be

sized to half the dimensions of the raw image, effectively giving it twice the pixel density of normal images. If you add two images with the same name, but one has the additional "@2x" added to the filename, the iOS devices will automatically use the high-res image on devices with a retina display and the lower-res image on all other devices. This allows you to build one layout and automatically support devices with different screen resolutions. Note that the high-res ("@2x") images must reside in the same button pack as the lower-res versions.

- Create a ZIP file containing the directory which in turn contains your button images. That is, the ZIP file should contain one directory which in turn contains multiple images. The ZIP file must be named the same as the directory it contains, but with the .zip extension. Take a look at the sample button pack files in the images directory (see below) as an example.
- Place the ZIP file in the TouchControl images directory (default is My Documents/TouchControl/images).
- If downloading the button packs from this site, they will already be in the correct format, so simply download them directly to your TouchControl images directory.
- Open the settings page by selecting Tools > Settings. Click the Add button next to the Button Packs list, select the button packs you'd like to use, and click OK. Then click Save at the bottom of the settings screen.
- Stop and restart the TouchControl Server program, and your images should be available the next time you add a button to a remote layout screen.

Download!

Check the [Download](#) page for newly available backgrounds and button packs. New images will be posted periodically.

Create!

Find out where you can get more buttons on the [FAQ](#) page...

Share!

If you have a background and/or button pack that you are particularly proud of and you'd like to share it, feel free to [send it to me](#) and I'll post it on the [Download](#) page on this site so that others can take advantage of it. In the future I hope to allow TouchControl users to upload their own content to the site, including complete activity configurations with remote IR codes, layout, images, etc., so watch for that!

[Scripting](#)

Scripting is used within TouchControl to process feedback from your devices, as well as augment and/or alter the normal/defined processing performed by the buttons on your remote interfaces. TouchControl scripting uses standard JavaScript, and is supplied by you during the design process within TouchControl Server. Scripting is not required to use TouchControl, but can make your control interfaces infinitely smarter and more powerful. The primary type of script available within TouchControl is feedback script and is available for any button that supports feedback: Command, AutoHotKey, EventTrigger, Global Caché, and HTTP Request. Other advanced scripting features are discussed [here](#).

[Editing Script](#)

Button script can be edited directly within the button configuration panels, and also within a larger editing window that is available when pressing the "Edit" button above any script field. In addition, when an external script editor [has been specified within Script Manager](#), clicking the "Edit" button above any script field will launch the external editor, populating it with any script found in the script field. When the script is then saved within the external editor, the script within the button configuration panel will be automatically updated with the saved script. The script can be re-edited and re-saved as long as both the button configuration panel and the external editor remain open. If the button config panel is closed, any script visible in the external editor can be discarded, as it is no longer linked to the button script. Re-clicking the "Edit" button above the script field will once-again load the button script into the external editor and re-establish the link.

[Feedback Script](#)

Feedback script takes the data returned from the device you are controlling and does something useful with it, such as updating the remote interface with text and/or images, executing another button/command, switching to another remote control screen, updating a variable for later use, etc. The majority of the JavaScript required for feedback script is simple string parsing - taking the data returned from the device, parsing and analyzing it to determine the status of the device that returned it, and then building a new string (referred to here as the "return string") which tells TouchControl what to do next.

The data returned from the device you are controlling is handed to your feedback script in a variable named `_feedback`, which your JavaScript can read just like any other JS variable. This feedback will contain all raw data received, including carriage return and/or linefeed characters.

NOTE: The "Show script errors" option under "Script Settings" in TouchControl settings on your iOS device is turned on by default. This will display a prompt message for each

script error found, with varying levels of information depending on the type of error encountered. Without this option enabled, script errors will still occur, but you will not be notified of them.

The Return String

- **Note:** The script return string feature was introduced with the first iteration of scripting in the TouchControl app. It continues to work and can be an easy and effective solution for basic needs. However, button script variables and the `_get...` and `_set...` script helper functions that were introduced as more advanced scripting capabilities were added to the app can be a more flexible and powerful scripting technique. See [Button Script Variables](#) and [Helper Functions](#) in the [Advanced Scripting](#) topic for more information.

The TouchControl script “return string” is a defined, formatted string that you return from your script which TouchControl can analyze to determine what actions to take next. The return string can contain several different pieces of information and must be created in a specific format for TouchControl to use it. That format consists of the following elements:

Button text/image/icon:

When updating a button's text, image, and/or icon, this must be the first element supplied in the return string*. This element has the following format:

buttonName^buttonText[@]buttonImage~buttonIcon

Each of the segments within this element may be supplied individually or together:

Update button text only:

```
return 'buttonName^buttonText';
```

Update button image only:

```
return 'buttonName[@]buttonImage';
```

Update button icon only:

```
return 'buttonName[@]~buttonIcon';
```

Update button text and image:

```
return 'buttonName^buttonText[@]buttonImage';
```

Update button image and icon:

```
return 'buttonName[@]buttonImage~buttonIcon';
```

You may update multiple buttons with the same return string using the "|" (pipe) character as follows:

return

'buttonName^buttonText[@]buttonImage | buttonName[@]~buttonIcon | etc...';

When used within Apple Watch activities, the return string may also include an element that generates an alert notification on the watch as follows:

[alert:My Alert Title^My alert text]

This element may be located anywhere within the return string, and the return string may contain only one alert element.

Execute a button

When executing another button, this element must either follow any text/image/icon element (above), or be the first element supplied in the return string* (if no text/image/icon element exists). This element has the following format:

[#]buttonName or [#]deviceName^buttonName

When the button name is unique within the activity layout, you need only supply its name. If you have buttons from multiple devices with the same name found in the activity layout, you should also supply the button's device name, as buttons must be unique within a device.

You may also execute multiple buttons by returning multiple execution elements together, as follows:

[#]buttonName1[#]buttonName2[#]buttonName3[#]etc...

Some button types can be updated using the execution return string element. The content of a web view, the value of a sliders, and the selected element of a spinner can be set by returning:

[#]mywebview==http://someNewUrl.com

or

[#]mywebview==<body>some new html </body>

[#]myslider==100 (i.e. any valid value for the slider)

[#]myspinner==A spinner entry (i.e. one of the spinner's entries)

Referencing spinner, slider and web view buttons in this manner will also result in the button's (the spinner/slider/web view) script being executed – but does not execute the button configured to be executed by a slider or spinner.

Additional timer button flags

The above method of executing a button, when used with a timer button, will start the timer button if it's stopped, or stop the timer button if it's running, effectively acting as a toggle. However, if you simply want to ensure that a timer is running (even if it's already running), or ensure that a timer is stopped (even if it's already stopped), or extend a currently running timer with a new interval, you can specify additional timer action flags in the script return string to perform these functions as follows.

- Note that any flags below that contain “!” will always result in the button being immediately executed, whereas without the “!”, the button may or may not be executed depending on the current state of the timer.

return '[#]{>}myTimerButton';

- *This will start a timer (and immediately execute the button) if it is not started, or leave it running at its current interval if it's already running (not immediately executing the button).*

return '[#]{>>}myTimerButton';

- *This will start a timer (and immediately execute the button) if it is not started, or extend it with a new, fresh interval if it's already running (not immediately executing the button).*

return '[#]{<}myTimerButton';

- *This will stop a timer if it's running, or leave it stopped if it's not running.*

return '[#]!myTimerButton';

- *This will immediately execute a timer button, and leave a timer running at its current interval if it's already running, or leave it stopped if it's not running - basically executing the button as a normal button, bypassing any timer processing.*

return '[#]{>!}myTimerButton';

- *This will immediately execute a timer button, and start a timer if it's not started, or leave it running at its current interval if it's already running.*

return '#{>>!}myTimerButton';

- *This will immediately execute a timer button, and start a timer if it's not started, or extend it with a new, fresh interval if it's already running.*

return '#{<!}myTimerButton';

- *This will immediately execute a timer button, and stop a timer if it's running, or leave it stopped if it's not running.*

Set a local variable

To set local variable for use in a later script, this element must either follow any text/image/icon element and any button execution element, or be the first element supplied in the return string* (if no text/image/icon or button execution elements exist). These variables are local to the currently active (foreground) activity only, and may be accessed from buttons on that activity. This element has the following format:

[&&]varName^varValue

or, to set multiple variables...

[&&]varName1^varValue1|varName2^varValue2|varName3^varValue3|etc...

The name/value pair is added to the local variable collection (the "_local" object) and may be later used within script as follows:

_local.varName

example:

```
if _local.varName == '0' {
    return 'myButton^Off';
} else {
    return 'myButton^On';
}
```

You may also use local variables within any button command (not just within a button's feedback script). To use the value from any local variable in a command, insert **%varName%** (where varName is any local variable name) in the button's command at the position you'd like to substitute the given global variable's value. This allows the feedback from one button to play a direct role in the command sent by a subsequent button.

Use **%0x:varName%** if you wish to automatically convert the value of **varName** to a Hex value. Note that the value of **varName** must be convertible to a decimal (number) value.

Note that if there is a `_global` variable (see below) with the same name as a `_local` variable, when using that variable for `%varname%` substitution, the `_local` variable will take priority. This allows you to override a `_global` variable with a `_local` one in an individual activity if you wish.

Set a global variable

To set global variable for use in a later script, this element must either follow any text/image/icon element and any button execution element and any local variable element, or be the first element supplied in the return string* (if no text/image/icon or button execution elements exist). These variables are global to the entire TouchControl app, and may be accessed from buttons on any activity. This element has the following format:

[&]varName^varValue

or, to set multiple variables...

[&]varName1^varValue1 | varName2^varValue2 | varName3^varValue3 | etc...

The name/value pair is added to the global variable collection (the "`_global`" variable) and may be later used within script as follows:

`_global.varName`

example:

```
if _global.varName == '0' {
    return 'myButton^Off';
} else {
    return 'myButton^On';
}
```

You may also use global variables within any button command (not just within a button's feedback script). To use the value from any global variable in a command, insert **%varName%** (where `varName` is any global variable name) in the button's command at the position you'd like to substitute the given global variable's value. This allows the feedback from one button to play a direct role in

the command sent by a subsequent button.

Use **%0x:varName%** if you wish to automatically convert the value of **varName** to a Hex value. Note that the value of **varName** must be convertible to a decimal (number) value.

Note that if there is a `_local` variable (see above) with the same name as a `_global` variable, when using that variable for `%varname%` substitution, the `_local` variable will take priority. This allows you to override a `_global` variable with a `_local` one in an individual activity if you wish.

Cancellation elements

Various actions/processes in TouchControl can be cancelled via a script's return value. These elements may reside anywhere within the return string, regardless of the existence of any of the previously discussed elements. Therefore, although the above elements indicate they must be the first ones found in the return string, these cancellation elements may precede any of them and still maintain a valid return string.

- [!] - this element will terminate the network connection with the device being controlled by the button as soon as the script exits. Subsequent buttons controlling that device will need to establish a new connection.
- [*] - this element will cancel any further execution of a button when returned from the button's pre-script. Any defined command, feedback script, or post script for that button will be ignored.
- [**] - this element will cancel a timer button when returned from the timer button's own feedback script. If the executing button is not a timer button, this element does nothing.
- [***] - this element will cancel the execution of a macro when returned from the feedback script of a button contained within the macro. Any buttons found after the button returning this element within the macro will not execute. If the executing button is not part of a macro, this element does nothing.

Examples

Here are a few examples of valid return strings.

```
return 'myLabel^myText[@]buttonpack/buttonfile.png[#]deviceName^buttonName';
```

This return string updates the text on the "myLabel" button to the string "myText", updates the background image of that same button to the image

"buttonpack/buttonfile.png", and executes the "deviceName^buttonName" button found on your layout.

```
return 'myButton[@]buttonpack/buttonfile.png~buttonicons/myicons/iconfile.png';
```

This return string updates the background image of the "myButton" button to the image "buttonpack/buttonfile.png", and updates that same button's icon to "buttonicons/myicons/iconfile.png".

```
return '[#]buttonName';
```

This return string only executes the button "buttonName" found on your layout.

```
return '[&]myVar^myValue[**]';
```

This return string sets the global variable `_global.myVar` to the value 'myValue', and cancels the timer button (assuming the button executing this script is a timer button).

```
return '[***][#]buttonName';
```

This return string cancels the currently running macro and executes the button "buttonName" found on your layout.

Full Script Examples

This simple script is used with a "Power On" button for a Denon receiver. The button sends a "PW?" command to check the power-on status of the receiver, which returns the actual power status of the device as feedback. If the power is off ("PWSTANDBY"), it executes the "pwon" button (another TouchControl button which is configured to send the actual power on command), otherwise it does nothing. The "pwon" button must reside within the activity layout, and may be added to the activity with a small, empty hotspot so that it is not visible on the screen, and disabled if desired. This script also tells TouchControl to release the socket when it has completed.

```
if (_feedback == 'PWSTANDBY') {  
    return '[#]pwon[!];'  
}
```

This script is used on the Vol+, Vol-, Mute On, and Mute Off buttons. The commands for each of those buttons returns feedback including the volume/mute status of the device. A label is updated in each case showing either the mute status, or the current master volume of the device, and updates the label image based on the mute status. A global variable named "denonMute" is also created when updating the mute status for use later or by other activities as needed. This script also uses various standard JavaScript string parsing methods (substr, indexOf, length, etc.).

```

if (_feedback.substr(0, 4) == 'MUON') {
    return 'VolumeLabel^Mute On[@]plastic/redcircle_40x40.png[&]denonMute^On';
} else {
    if (_feedback.substr(0, 5) == 'MUOFF') {
        return 'VolumeLabel^Mute Off[@]plastic/greencircle_40x40.png[&]denonMute^Off';
    } else {
        var fbString = _feedback.substr(2, _feedback.indexOf('MVMAX') - 2);
        if (fbString.length <= 2) {
            return 'VolumeLabel^' + fbString;
        } else {
            return 'VolumeLabel^' + fbString.substr(0, 2) + '.' + fbString.substr(2);
        }
    }
}
}

```

Feedback Flags

Using feedback scripting, you may also specify a duration of time to wait for feedback before executing your feedback script, a specific terminator to look for in your feedback, or a minimum length of data that must be returned before the iOS device stops waiting for feedback and continues. Note that these elements are not considered part of your actual feedback script, and are processed and removed from the script before it is handed to the JavaScript engine for further processing.

To instruct TouchControl to wait for a given amount of time for feedback before executing your script, simply add "[wait]*duration*;" to the beginning of your feedback script. NOTE that the format of this is important. It must begin with "[wait]" (without the quotes), and must end with a semicolon, and it may reside immediately before or after either of the following two elements ([term] or [len]), if they exist, but otherwise must be at the beginning of your supplied feedback script. The value for duration must be any valid integer or float value indicating the number of seconds to wait for feedback before proceeding (e.g. 5, 10, 3.5, 2.25, etc).

- Note that TC will wait for the specified duration *before* starting any feedback processing, so the countdown of the *Feedback Timeout* value that you have set in TC's Network Settings won't start until the [wait] period has passed.

To set a specific termination character to look for within the feedback data, simply add "[term]*terminator*;" to the beginning of your feedback script. NOTE that the format of this is important. It must begin with "[term]" (without the quotes), and must end with a semicolon, and it must be at the beginning of your supplied feedback script (or immediately following the [wait] element if it exists). The value you supply as the terminator can be either a literal string value, or can be an escape sequence (i.e. \n for new line, or \r for carriage return), or can be

any HEX value formatted as: \xNN (where NN is any two-character HEX value, such as \x1A for the EOF character). If the specified termination character/string is not found after all feedback data has been read, TouchControl will timeout and display an error.

Alternately, to set a minimum required feedback data length that you expect to receive, simply add "[len]/length;" to the beginning of your feedback script. NOTE that the format of this is important. It must begin with "[len]" (without the quotes), it must end with a semicolon, it must be at the beginning of your supplied feedback script (or immediately following the [wait] element if it exists), and the value you enter as the length must be a valid integer value. Once the minimum length of feedback data has been reached, TouchControl will continue. If the minimum length is not reached, TouchControl will eventually timeout (using the TC settings *Feedback Timeout* value) and display an error.

TouchControl receives the feedback from your devices in "chunks" of data. Therefore, when using the [term] or [len] features, those elements only need to be found anywhere within the "chunk" of data being processed at the moment. That is, TouchControl will not truncate any data found after the termination character or length specified within the current chunk, so your script will receive all data received up to and including all data within that chunk, including the termination character/string (if using [term]).

Adding "[notimeout]" to your feedback script will tell TouchControl to ignore any feedback timeout errors. Use this to suppress unwanted error popups when you expect feedback from a device, but don't really care if the feedback is received or not (such as in a polling scenario where you'll be executing the command again in the future anyway). Using '[notimeout]' will also suppress timeout messages for HTTP request buttons.

Adding "[quiet]" to your feedback script will tell TouchControl to ignore any socket or feedback errors. This is similar to [notimeout], but in addition will also suppress errors generated by Global Caché devices, messages generated when [len] or [term] (above) are not satisfied, and HTTP errors in addition to those that are timeout related.

Adding "[0x]" to your feedback script will force TouchControl to convert the feedback it receives to a HEX string. Like the other advanced flags, TouchControl will strip the "[0x]" from your feedback script before executing it. If you include "[0x]" in your script, it must follow any of the above advanced flags, but precede your actual script.

Adding "[sync]" to your feedback script will force TouchControl to process all feedback before allowing any other button(s) to execute. Without this flag, other buttons could execute while a button's feedback script is running, especially if that feedback script is performing multiple tasks, such as updating the UI as well as executing other buttons using a return string, for

example.

Feedback Slicing

An additional feedback script flag allows you to control what feedback data generated by your devices is presented to your feedback script. Similar to the [term] and [len] script tags which determine how much feedback is expected from your device, the **[slice]** tag allows you to parse the feedback and only return a desired “slice” of the feedback to your script. To use this feature, include the following at the beginning of your feedback script:

```
[slice]prefix[:]suffix[:]
```

This tag tells TouchControl to search through the feedback data and return only the data found **after** the provided *prefix* string, and **up to but not including** the provided *suffix* string. For example, if your feedback data consisted of this:

```
POWER1\r\nVOLUME20\r\nSURROUNDS5\r\nINPUT3\r\n
```

Using this feedback slicing:

```
[slice]VOLUME[:]\r\n[:]
```

...would return the string **20** to your feedback script. This allows you to reduce the feedback that your script receives to just what you are specifically interested in, so your script can be simpler and not required to do extra parsing just to find the desired value. This is especially useful for devices that send large amounts of unrelated feedback, in addition to the desired feedback, when a specific command is sent (as if, in the above example, a command was sent to retrieve the current volume, but the power status, surround mode and current input were also returned as extraneous feedback).

You may also omit the *prefix* string and supply only the *suffix* string, which will return all data from the start of the feedback up to (but not including) the *suffix* string. Or you may omit the *suffix* string and only supply the *prefix* string, which will return all data found after (not including) the *prefix* string, through the end of the feedback. When omitting the *prefix* or *suffix* strings, you must still include the delimiters, as shown in these examples:

```
[slice]VOLUME[:];;
```

```
[slice][:]\r\n[:]
```

If TouchControl searches the feedback data and finds the provided *prefix* string, but does not find the provided *suffix* string, it will return all data found after *prefix*, to the end of the returned feedback (just as if the *suffix* string was omitted). If TouchControl does not find the *prefix* string, the feedback processing will wait for the selected feedback timeout duration (set in TouchControl settings) and will then time out (just as it does using the **[term]** or **[len]** flags if the specified terminator is not found or feedback length is not reached).

In addition, if your activity includes both feedback-enabled buttons that use slicing (as shown above), and also a Feedback Client button using the same remote host, any feedback that the feedback-enabled button's slicing *does not process* (i.e. any feedback found before the *prefix* or after the *suffix*) will be passed on to your Feedback Client button's script to process. So, in the above example, the feedback would be processed in the following order:

1. The Feedback Client button would receive the feedback value of **POWER1\r\n**
2. The feedback-enabled slicing button would receive the feedback value of **20**
3. The Feedback Client button would receive the feedback value of **SURROUND5\r\nINPUT3\r\n**

This allows you to still process other non-sliced feedback (in the order received) if desired. Including a Feedback Client button is not required, and if one does not exist using the same remote host as the slicing button, the remaining feedback will be discarded. Other feedback-enabled buttons will not receive the remaining feedback from another feedback-enabled slicing button. This only applies to Feedback Client buttons that use the same host. **NOTE:** If the result of a slice is an empty string – that is, your *prefix* and *suffix* are found, but there is no data in the feedback between those strings (such as using **[slice]SURROUND5[:]\r\n[:]** in the above example), TouchControl will present no feedback to your feedback script, and thus your feedback script will have no feedback to process, and any script that uses the **_feedback** variable will not run. You may, however, still use a **return** statement in your feedback script to execute a button or update UI elements, even if no feedback is returned from the slicing.

Updating activity background images via script

You can dynamically change the background image of an activity at run time via script. Simply add the following statement to any type of script of any button:

```
_background='myNewBackground.png';
```

The new background image must exist in the backgrounds.zip file in your images directory, and the filename used in the above script must include the file extension (i.e. '.png' in the above

example). The `_background` variable may be set to a static string, as above, or you may set it to a variable that holds the background image file name. When this script executes, the new background will fade in, replacing the previous background image. By default, the duration of the fade will use the "Animation Duration" setting under "Script Settings" in the TouchControl app on your iOS device. If you'd like to change the fade duration for this execution, simply add the following script as well:

```
_backgroundDuration=2.5;
```

Set the `_backgroundDuration` script var to any float value indicating the number of seconds for the fade duration.

If the new background image is not the same size as the previous background image, the activity will be re-sized to fit the new background image. This would allow you to dynamically alter the size of your activity at run time, potentially turning a single-screen activity into a scrolling, multi-screen activity, as an example. If you would like the activity size to remain constant, regardless of the size of the newly supplied background image, simply add the following script as well:

```
_backgroundAdjust=false;
```

Using this script, if the new background image is not the same size as the previous image, the new image will stretch or shrink to fit the size of the previous image. The default for `_backgroundAdjust` is true (re-size the activity).

Determining device network status via script

The following two script variables will allow you to determine your iOS device's current network status (i.e. WiFi or WWAN) at any time:

`_isWiFi` (true if the device is currently connected to a WiFi network, otherwise false)

`_isWWAN` (true if the device is currently connected to a WAN/cellular network, otherwise false)

Usage:

```
if (_isWiFi) {  
    //do something here  
}
```

Disable full-screen activity "go-back" swiping via script

Normally when using a TouchControl activity in full-screen mode, a swipe either from left-to-right, or from top-to-bottom on a blank spot on the activity background will perform the equivalent of a "go back" link, returning you to the previous activity or home screen. If this

gesture interferes with the use of your activity, such as when using sliders, you can disable the swiping gesture for the given activity with the following script helper function:

```
\_enableActivitySwipe\(false\);
```

You can re-enable swiping by calling the helper function and passing true as the parameter as well. Note that this only affects the activity during its life-cycle, so if you close and re-open the activity, you'll need to re-execute this helper function, likely in an auto-exec on load button.

[Advanced Scripting](#)

Advanced scripting features improve and enhance the feedback and scripting functionality of TouchControl, providing advanced automation capabilities.

[Button pre-script and post-script](#)

Most button types now include the ability to execute script before and/or after the “command-feedback cycle” (sending the button’s command and receiving and processing any generated feedback), giving you both greater control over the button’s actions, as well as the potential to consolidate multiple actions into a single button.

Button pre-script runs immediately when you tap on a button on your remote control screen, and before the button’s defined action executes. Pre-script has access to the executing button’s properties (such as the button’s name, type, text, image, icon, command, slider/spinner values, etc.) via script variables available at runtime. See “Button Script Variables” later in this document for information regarding those script variables. Pre-script allows you to, among other things, interrogate and/or update a button’s command immediately before it executes, potentially changing a button’s behavior at runtime based on global variable values, other button states, etc. Pre-script also provides a mechanism for cancelling a button’s command before it executes, potentially allow a button to determine its own fate, rather than requiring an additional button and its feedback to determine whether or not it should be executed. These capabilities have the potential to consolidate the processing that previously may have required multiple buttons to perform. Button pre-script is comprised of standard JavaScript, as is all other script contained within TouchControl.

If a button supports pre-script, you will find a “Pre-Script” input field on the button’s configuration panel within TouchControl server. Simply enter your custom JavaScript in that field, refresh the configuration to your device, and the script will begin executing as soon as you tap on the button on your remote control screen.

Button post-script runs immediately after a button’s defined action executes (including receiving any feedback and any feedback script has completed). Post-script has access to the executing button’s properties (such as the button’s name, type, text, image, icon, slider/spinner values, etc.) via script variables available at runtime. See “Button Script Variables” later in this document for information regarding those script variables. These capabilities have the potential to consolidate the processing that previously may have required multiple buttons to perform. Button post-script is comprised of standard JavaScript, as is all other script contained within TouchControl.

If a button supports post-script, you will find a “Post-Script” input field on the button’s

configuration panel within TouchControl server. Simply enter your custom JavaScript in that field, refresh the configuration to your device, and the script will execute after your button has completed its defined action. Note that unlike pre-script, post-script does not have access to the button's command, as the command has already executed prior to post-script running. Also, the script defined for both spinner and slider buttons will always run as post-script, that is after the spinner or slider control has finished with its selection event.

Custom Button Properties

Any button may have its own custom "properties" defined, which are simply name/value pairs that you assign to individual buttons which are then available within script at runtime. Your script (pre/post/feedback) may read and/or write to those property variables at button execution time, giving you enhanced abilities for conditional command execution, finding and executing buttons from another button's script, etc. Properties may be added to buttons from either the buttons list, and/or after a button has been added to an activity layout within the layout designer. Adding properties from the buttons list allows you to create "default" properties that carry forward to any activity the button may be added to. Alternately, adding properties to buttons after they have been added to an activity layout allows a button to be used in multiple activities, but with different, unique properties in each activity.

A button may also display any of its property values as its text on the iOS device screen. This allows you to have the same buttons displayed on a remote control screen multiple times, but with different text.

To add properties to buttons:

1. First you must define the properties that will be available to all buttons. Choose Tools – Settings from the TouchControl Server menu, then click the "Custom Properties" button to open the Property Manager panel.
2. Click the "+" button to add a new property, enter a unique property name, then click "Save" to add the property name to the list. Each property name can then be given a unique value for each button on your layouts.
3. Click the "OK" button to close the Property Manager panel, then click "Save" to exit Settings.
4. Create a button
5. To add a "default" property to the button, right-click on the button in the buttons list, and select "Properties...". Or to add an activity-specific property to the button, drag the button to a layout, then right-click on the button and select "Properties...". Either of these methods will open the Button Properties panel for that button.

6. Choose the property you would like to add to this button from the "Name" list at the top (the list created in steps 1-4).
7. Enter a value for that property for this button.
8. Click the "+" button to add the property name to the button with the specified value. You may only add an individual property once to a button.
9. To edit a property value, select it in the list and change the value above, then click the "+" button again to set the new value for the property.
10. Select a property in the list and click the "-" button at the bottom to remove a property from the button.
11. Click the "Save" button when finished adding/updating the button's properties.
12. If a button which has default properties assigned is added to a layout, accessing the button's properties from the layout designer will display the default properties for that button. You may delete or update the value for any of the default properties, and those changes will be local to the current activity (i.e. will not alter the default properties for this button on any other activity).

Now at runtime, within pre-, post-, or feedback script, you may access the currently executing button's properties via JavaScript object notation using the `_property` object:

`_property.myProperty`

You may also update or create new properties in the same manner:

```
_property.myProperty = 'new value';  
_property.newProperty = 'a value';
```

Where this becomes valuable and powerful is when combined with the current method of updating and/or executing another button from the current button's script. Currently you can update a button's text, image, and/or icon by returning a string similar to the following from a button's script:

```
return 'Mute^Mute On[@]mybuttons/muteon.png';  
(where "Mute" is the name of the button, "Mute On" is the new text displayed on the button, and "mybuttons/muteon.png" is the button's new background image)
```

With the addition of button properties, you can update a button or buttons using the following script return string:

```
return 'myPropertyName~myPropertyValue[@]mybuttons/blue.png';
```

(where myPropertyName is a custom property, and myPropertyValue is its value)

This return method will match all buttons with the given property/value pair, and update the text/image/icon of all of them (only the image is updated in the above example).

In the same manner, you can currently trigger the execution of a button by returning a string similar to the following from a button's script:

```
return ['#]Play';
```

(where "Play" is the name of a button on your layout)

Now with the addition of button properties, you can execute a button or buttons using the following script return string:

```
return ['#]myPropertyName~myPropertyValue';
```

(where myPropertyName is a custom property, and myPropertyValue is its value)

This return method will match all buttons with the given property/value pair, and execute each one (which in turn could use the same method to update/execute other buttons as needed). So, with the combined ability to update button properties at runtime, and then use those properties to find, update and/or execute buttons dynamically through script, this opens up the capability to create extremely dynamic and interactive custom control interfaces.

(Mac only) In addition to the above methods of setting button properties, if you right-click on a device in the "Available Devices" list, and then select the "Property..." option, a dialog will appear that will allow you to add a property to, or remove a property from, all buttons in that device. This is the same as using the Buttons list right-click method of setting a property on a button, but allows you to set the property on all buttons in the device simultaneously.

Built-In Button Properties

Some button types have "built-in" button properties that are pre-existing and known to TouchControl. These properties can be used and referenced exactly like any other custom property (above), but also provide certain internal type-specific functionality for their associated button types. The following is the list of built-in properties and their provided functionality:

For buttons that display text

Property name	Value	Purpose	Default
TextAlign	left/center/right	Align text within the button	center
TextFont	fontname (e.g. Georgia-Bold) fontname^size (e.g. GillSans-Italic^24)	Set the button's font to any iOS font name and optional size	Bold system font at size selected in interface designer

For Slider buttons

Property name	Value	Purpose	Default
sliderBarLeft	See Custom Slider Images	Provide custom slider bar left image	Native iOS slider bar
sliderBarRight	See Custom Slider Images	Provide custom slider bar right image	Native iOS slider bar
sliderThumb	See Custom Slider Images	Provide custom slider thumb image	Native iOS slider thumb
useGlass	yes or true Remove the property to disable the glass effect. See Liquid Glass	Display the slider using glass material	None/not set

For HTTP Request buttons

Property name	Value	Purpose	Default
HTTPAuth	userid:password	Provide credentials for web requests	None
HTTPHeaders	headername1=headervalue1^ headername2=headervalue2^etc...	Provide any required HTTP headers for web requests	Standard headers
UserAgent	See Alter a Web View's Identity	Provide custom user agent for Web View	Standard Mobile Safari user agent

For all button types that render as a button on the screen

Property name	Values	Purpose	Default
CornerRadius	Any integer	Round the corners of the button	0 (square corners)
stretch	true/false	Setting to false prevents the button from stretching or compressing in an activity set to "Scale to Fit".	true

For Gesture Pad buttons

Property name	Values	Purpose	Default
MouseServer (Windows server only)	See Redirect Mouse & Keyboard Control	Control the mouse and keyboard on computers other than the primary TouchControl server system	None

For Group buttons

Property name	Values	Purpose	Default
TouchMotionXExtents	xLeft,xRight (e.g. 10,300) See TouchMotion	Constrains the movement of a group when using TouchMotion to the extents provided (i.e. left edge cannot move left of xLeft, and right edge cannot move right of xRight).	None
TouchMotionYExtents	yTop,yBottom (e.g. 5,450) See TouchMotion	Constrains the movement of a group when using TouchMotion to the extents provided (i.e. top edge cannot move above yTop, and bottom edge cannot move below yBottom).	None
GlassSpacing	Any integer greater than 0. Higher values increase the effect. 0 disables the effect. See Liquid Glass	Dictates the distance that Liquid Glass buttons added to the group need to be spaced in relation to each other to trigger Liquid Glass interactions and morphing effects.	None

For _deviceMotion buttons

Property name	Values	Purpose	Default
MotionInterval	See Device Motion Sensing	Sets the device motion sensing update interval	Per device

For _mouseMoveButton buttons

Property name	Values	Purpose	Default
---------------	--------	---------	---------

MouseThrottle	Integer (e.g. 5) See Gesture Pad Mousepad	The number of pixels required to swipe before moving the mouse pointer – provides mouse speed control	3
---------------	--	---	---

For `_macroMessage` buttons

Property name	Values	Purpose	Default
DisplayFrame	See MacroMessage	Provide size and position parameters for macro messages	Size and position of <code>_macroMessage</code> button on layout

For all buttons on a watch activity

Property name	Values	Purpose	Default
WatchIndex	See Watch Button Configuration	Provide ordering/positioning for buttons on a watch layout	None

For Multi-Peer buttons

Property name	Values	Purpose	Default
MPAutoAccept	See Multi-Peer Button Properties	Sender auto-accepts all requests from receivers	False
MPConnectScript	See Multi-Peer Button Properties	Script that runs when a peer connects	None
MPDisconnectScript	See Multi-Peer Button Properties	Script that runs when a peer disconnects	None
MPSendDataMode	See Multi-Peer Button Properties	Allows ensuring all messages are sent (Reliable)	Unreliable

Custom script libraries

With the increase in scripting capabilities for buttons, TouchControl also adds the ability to create custom script libraries that can contain variables and function to use from within your button scripts, eliminating the need to duplicate common code in multiple buttons/locations. Script libraries can be added in two forms, either raw script entered directly into TouchControl server, or URLs that point to JavaScript source files location on other Web servers.

To create or reference a script library:

1. Choose Tools – Settings from the TouchControl Server menu, then click the “Script Manager” button to open the Script Libraries panel.

2. Select the type of library you would like to add from the tabs at the top. The “URLs” option reference URL links to libraries on other servers. The “Script” option allows you to enter your own raw JavaScript which is stored within TouchControl Server. The “Files” option allows you to specify external script (.js) files in your PC's file system or on a network drive, which will be downloaded to your iOS device during config refresh. The “Internal” option presents a list of “built-in” libraries provided by the developer or other 3rd parties that are there for you to use. There may or may not be libraries available on this tab. NOTE: Currently external script files are not included in the configuration export/import process, so when exporting and sharing configurations between PC's or between users, those script files, if any, must be transferred along with the configuration .zip file, and the references to those files in Script Manager must be updated with the locations of the script files on the destination system.
3. When “URLs”, “Script” or “Files” is selected, click the “+” button to open an interface that allows you to enter the required information for the library – a unique name for the library (for your reference), and either a URL to a file on a remote server, a filename, or the raw script itself.
4. You may also add an external script file to the "Files" list by dragging and dropping the script (.js) file onto the "Files" tab/list. This will open the new script file panel, allowing you to provide a unique name for the script library file.
5. To modify a script library entry, simply double-click it in the list.
6. To remove a script library entry, select it in the list and click the “-“ button.
7. To include a built-in “Internal” script library, switch to the “Internal” tab and simply select the library entry you desire, or de-select it to no longer load it in the device app.
8. Once you’ve included all libraries of each type that you desire, click the “OK” button to exit the Script Manager. Then click “Save” to exit server Settings.

After adding external script files to the "Files" list, you may also edit those .js files directly from the Script Manager interface. To enable this, you must first set the default script editor executable by right-clicking the "Files" list and selecting "Set file editor...", and entering the path and filename of the executable of your desired script editor. Alternately, you may drag and drop the executable (.exe) file for your desired script editor onto the "Files" list to automatically set it as your default script editor. Once your script editor is set, you may right-click on any entry in the "Files" list to launch that file in the specified script editor program. Note that the editor specified here is also used to edit script within the server button designer panels, when the "Use external script editor in designer" server setting/preference is enabled.

Once your script libraries are defined, the next time you refresh the configuration within TouchControl on your device, the app will pull down the script libraries from the specified

locations (either from TouchControl server or from a remote Web server). Then JavaScript within your buttons may reference the functions and variables within the script libraries using standard JavaScript syntax:

```
if (myScriptLibraryVar == true) { ... };  
var x = myScriptLibraryFunction();
```

If a function from a script library needs to return data (such as a formatted return string) to TouchControl, when calling that function from your button script, you must "chain" the returned data from the called function on to TouchControl. For example:

Your script library function:

```
function myScriptLibraryFunction() {  
...  
...  
return 'myButton^newText';  
}
```

Your button script:

```
return myScriptLibraryFunction();
```

Without this "chaining", your script library function will simply return its string to your button script, but not on to TouchControl itself.

Please note that referencing script libraries located on other servers (the "URL" type) has the potential to slow down the initial load or configuration refresh in the app on your device, if those script libraries are very large, or are for some reason slow to load or unavailable. Under normal circumstances with reasonably sized libraries located on servers on the local LAN network, you should not have any issues. ("Reasonably sized" is an intentionally vague term, and is based on what your interpretation of "slow to load" is.)

Button Script Variables

TouchControl can read, and in some cases write, script variables containing information about the currently executing button. The following script variables are available:

- `_name` – the name of the currently executing button (read-only)
- `_type` – the type of the currently executing button (read-only)
- `_text` – the text displayed on the currently executing button (read/write)

- `_image` – the image displayed on the currently executing button (read/write)
- `_icon` – the icon displayed on the currently executing button (read/write)
- `_top` - the pixel location of the y coordinate of the button on the screen (read/write)
- `_left` - the pixel location of the x coordinate of the button on the screen (read/write)
- `_width` - the width of the button in pixels (read/write)
- `_height` - the height of the button in pixels (read/write)
- `_rotation` - the buttons rotation, in degrees (read/write)
- `_alpha` - the button's opacity - 0.0=fully transparent, 1.0=fully opaque (read/write)
- `_textSize` - one of the designer values: "small", "medium", "large", "x-large"; or a numeric font size value (read/write)
- `_textColor` - one of the designer values: "black", "white", "blue", "red", "green", "yellow", "orange", "purple", "brown", "cyan", "magenta", "gray", "lightgray", "darkgray"; or a hex color value (read/write)
- `_buttonBackgroundColor` - one of the designer values: "black", "white", "blue", "red", "green", "yellow", "orange", "purple", "brown", "cyan", "magenta", "gray", "lightgray", "darkgray", "glass" (requires iOS 26+), "transdark", "translight", "transextralight"; or a hex color value (read/write)
- `_enabled` - the "Enabled" property of the button - also available from the button's popup (right-click) menu (read/write)
- `_command` – the command executed by the currently executing button (read/write)
- `_statusCode` - the HTTP status code from an HTTP request, when accessed from the feedback or post script of an HTTP Request button (read-only)
- `_interval` – the repeat interval of the currently executing button (read/write)
- `_sliderAction` – indicates whether the current button was executed by a slider (read-only)
- `_spinnerAction` – indicates whether the current button was executed by a spinner (read-only)
- `_slideSpinValue` – the current value of the slider/spinner which executed the button (read-only)
- `_selectedItem` – the selected item in a spinner/table (read/write)
- `_selectedIndex` – the selected index in a spinner/table (read/write)

- `_execByName` – the name of the button which executed the button via script (if any – read-only)
- `_execByType` – the type of the button which executed the button via script (if any – read-only)
- `_property` – object containing the button's custom properties (read/write – see "Custom Button Properties" above)
- `_items` – array containing a spinner button's entries (read/write)
- `_pressed` – the path to the "pressed" image defined for the currently executing button, if any (read/write)
- `_device` – the name of the device that contains the currently executing button (read-only)

These variables may be accessed from script using:

```
if (_name == 'myButton') { ... };
_text = 'My New Button Text';
if (_property.myPropertyName == 'myPropertyValue') { ... };
```

Although a button's text, image and icon may still be updated using the formatted script return string (as shown in the "Custom Button Properties" section above), you may also update these properties on a button directly from script without the requirement for the return string. Accessing these properties from your script can give you greatly enhanced visibility into and control over the buttons on your layouts, allowing a much more dynamic and interactive experience.

Helper functions

The above variable may be accessed directly to alter the properties of the currently executing button (the button that "owns" the script that references these variables. The read/write properties of other buttons on the layout may also be accessed and/or updated from any other button's script as well, using built-in "helper functions." Each of the above variables has corresponding "`_get`" and "`_set`" helper functions which allow you to read or write the property on the designated button. Below are just a few examples, but all of the above read/write vars have a corresponding "`_get`" and "`_set`" helper function.

- `_getText('myButton');`
- `_setText('myButton','new text');`
- `_appendText('myButton','text to append');`
- `_appendLine('myButton','text to append');` // adds a line feed after the supplied text

- `_setTextFont('myButton','Papyrus');` // any [supported font name](#)
- `_getImage('myButton');`
- `_setImage('myButton','myButtons/theButton.png');`
- `_getPressed('myButton');`
- `_setPressed('myButton','myButtons/theButton.png');`
- `_getIcon('myButton');`
- `_setIcon('myButton', 'buttonicons/White 20x20/play.png');`
- `_getTop('myButton');`
- `_setTop('myButton',100);`
- `_setTop('myButton',_getTop('myButton')+20);`
- `_getLeft('myButton');`
- `_setLeft('myButton',200);`
- `_setLocation('myButton',newLeft,newTop);`
- `_setAlpha('myButton',.5);`
- `_setRotation('myButton',_getRotation('myButton')+10);`
- `_setTextSize('myButton','medium');`
- `_setTextSize('button',20);`
- `_setTextColor('myButton','white');`
- `_setTextColor('button','#35FB4C');`
- `_setBackgroundColor('button','white');` // see all available colors above with the `_buttonBackgroundColor` variable)
- `_setBackgroundColor('button','#35FB4C');`
- `_setEnabled('myButton',false);`
- `_getCommand('button');`
- `_setCommand('myButton','some new command\n');` // include any required command terminator
- `_setItems('mySpinner', arrayOfItems);` // set new spinner items
- `_getSelectedItem('mySpinner');` // get the selected item value (string) - valid for spinners
- `_setSelectedItem('mySpinner', 'item value');` // select an item - valid for spinners

- `_getSelectedIndex('myButton');` // get the selected index (integer) - valid for sliders or spinners
- `_setSelectedIndex('myButton', 10);` // select an index - valid for sliders or spinners
- `_setFrameAndAlpha('myButton',newleft,newtop,newwidth,newheight,newalpha);`
- `_setFrameAndRotation('myButton',newleft,newtop,newwidth,newheight,newrotation);`
- `_setFrameAndAlphaAndRotation('myButton',newleft,newtop,newwidth,newheight,newalpha,newrotation);`
- `_animateTop('myButton',newtop);`
- `_animateLeft('myButton',newleft);`
- `_animateWidth('myButton',newwidth);`
- `_animateHeight('myButton',newheight);`
- `_animateAlpha('myButton',newalpha);`
- `_animateRotation('myButton',newangle);`
- `_animateLocation('myButton',newleft,newtop);`
- `_animateSize('myButton',newwidth,newheight);`
- `_animateFrame('myButton',newleft,newtop,newwidth,newheight);`
- `_animateFrameAndAlpha('myButton',newleft,newtop,newwidth,newheight,newalpha);`
- `_animateFrameAndRotation('myButton',newleft,newtop,newwidth,newheight,newrotation);`
- `_animateFrameAndAlphaAndRotation('myButton',newleft,newtop,newwidth,newheight,newalpha,newrotation);`

The following helper function are for use with spinner buttons using the "Grid" layout:

- `_scrollTo('MyGrid','first');` //scroll to the first button in the grid named "MyGrid"
- `_scrollTo('MyGrid','last');` //scroll to the last button in MyGrid
- `_scrollTo('MyGrid','top');` //scroll to the top of MyGrid (same as "first")
- `_scrollTo('MyGrid','bottom');` //scroll to the bottom of MyGrid (same as "last")
- `_scrollTo('MyGrid',12);` //scroll to the button at position 12 in MyGrid (any integer)

Other available non-button-specific script variables and helper functions

- `_deviceName` - the unstructured name of your device, found in iOS Settings - General - About
- `_ipAddress` - the current local WLAN IP address of the iOS device
- `_activity` - the name of the currently activity
- `_location` - the name of the current activity's location
- `_title` - the string displayed in navigation bar on the main activity screen - can be the current TouchControl Server's name, a custom string that you supply, or blank
- `_fullScreen` - returns true if the current activity is in full-screen mode, otherwise false.
- `_sleep` - the current device sleep mode ("enabled" or "disabled")
- `_brightness` - the current screen brightness between 0.0 and 1.0
- `_orientation` - the current device orientation ("portrait" or "landscape")
- `_screenWidth` - the width of the device screen
- `_screenHeight` - the height of the device screen
- `_activityWidth` - the width of the current activity layout
- `_activityHeight` - the height of the current activity layout
- `_tcClearHistory()` - remove all activities from the history stack except the current activity
- `_tcClearHistory('locationName^activityName')` - remove a single activity from history
- `_tcClearHistory('locationName^*')` - remove all activities for a given location from history
- `_tcClearHistory('*^activityName')` - remove all activities with a given name in any location from history
- `_tcClearHistory('!locationName^activityName')` - remove all activities from history except the one specified
- `_keyboardShowing` - indicates if the iOS keyboard is currently showing (true or false)
- `_sleep(ms)` - tell TouchControl to sleep for the specified number of milliseconds after the current script has completed.
- `_setActivityTitle('New Title')` - Change the title shown in the navigation bar at the top of the activity screen for the lifetime of the activity (i.e. must be re-set upon exiting/re-entering activity)

- `_dynamicServer` - dynamically change the server that is being used by a Gesture Pad mousepad, or by any AutoHotKey, Command, or IR (USB-UIRT) type buttons on the currently displayed layout
- `_appVersion` - returns the version of TouchControl currently running (read-only)
- `_isInBackground` – returns true if the app has been sent to the background, false if the app is currently in the foreground
- `_ping()` - see [this page](#)
- `_connectable()` - see [this page](#)
- `_speak()` and `_voices()` - see [this page](#)
- `_hideActivityButtons()` - hide all buttons on an activity layout (tap anywhere on the activity background to re-show all activity buttons)
- `_showActivityButtons()` - un-hide all buttons on an activity layout
- `_showGPKeyboard('padName')` - Show the keyboard for a specified gesture pad mousepad
- `_hideGPKeyboard()` - Hide the gesture pad mousepad keyboard
- `_toggleGPKeyboard('padName')` - Show/hide the keyboard for a specified gesture pad mousepad
- `_batteryMonitor` - see [this page](#)
- `_setFullScreen(true/false)`
- `_screenFullBright()`
- `_screenFullDim()`
- `_screenRestoreBright()`
- `_screenSetBright(absoluteValue)`
- `_screenIncreaseBright(relativeValue)`
- `_screenDecreaseBright(relativeValue)`
- `_getBright()`
- `_scroll(x, y)`
- `_scroll('right'/'left'/'bottom'/'top'/'page right'/'page left'/'page up'/'page down'/'page left+up'/'page left+down'/'page right+up'/'page right+down')`
- `_enableSleep()`

- `_disableSleep()`
- `_toggleSleep()`
- `_getSleep()`
- `_bringToFront('ButtonName')`
- `_sendToBack('ButtonName')`
- `_showKeyboard('TextFieldName')`
- `_hideKeyboard()`

Local/Global Variables, and State Variables via iCloud

In addition to the above variables, TouchControl also supports both local and global variables using script objects.

Local Variables

Local variables may hold any type of data, and are scoped to individual activities (i.e. they are only available within the activity in which they were created). Users can read and write local variables using JavaScript object notation as in the following examples:

```
_local.myLocalVar = true;  
var x = _local.myLocalVar;
```

A new `_local` object is created each time an activity is opened, and deleted when the activity is closed. Navigating from one activity to another (via link buttons) maintains the `_local` object in the calling activity (in the background), while creating a new `_local` object for use in the linked-to (foreground) activity. When returning to the calling activity, its `_local` object is in-tact, with all previously defined variables and values. `_local` objects in different activities can contain variables with the same name, but containing different data/values, without overwriting the data/values from `_local` objects in other living activities. TouchControl takes care of making sure the `_local` object for the currently visible (foreground) activity is in scope at the proper time.

Global Variables

You may also read and write global variables using JavaScript object notation as follows:

```
_global.myGlobalVar = true;  
var x = _global.myGlobalVar;
```

Global variables may hold any type of data, and are scoped to the application, so all `_global` variables are available from within any activity. The `_global` object is re-created (i.e. any variables created on the object are removed) each time the app is killed & restarted, or when the configuration is refreshed from TouchControl Server.

State Variables and iCloud

You may also store global variables in iCloud, and share them between devices running TouchControl. This is useful for storing state of devices that you control, or of the TouchControl app itself. For this purpose, the `_state` script object is available to store these state variables as follows:

```
_state.myStateVar1 = "On";  
_state.myStateVar2 = "red";
```

Any variables created or updated on the `_state` object are automatically sent to iCloud, where they are automatically synchronized to any other iOS device running TouchControl (and which uses the same Apple ID, as iCloud sharing is based on authenticated Apple ID on each device). `_state` variables may be used just like any other `_global` variable, i.e. accessed across activities, used in `%varname%` command substitutions, etc.

Normally, when you have this feature enabled, data from iCloud is synched to your device on startup, and each time a state variable is updated from another device. However, if the device you are currently using was the last one to update a variable's value in iCloud, then iCloud will assume you already have the most recent value, and will not sync the data to your device on subsequent sessions. So if you wish to make sure you are retrieving the most up-to-date value for a state variable from iCloud, you can use the `_getState('varname')` helper function in your script, which will return the current value for that `varname` stored in iCloud.

To enable iCloud and the `_state` script object, access settings in TouchControl on your iOS device, scroll down to "Script Settings", and select the "Use iCloud State" option to enable it. If this option is not enabled, the `_state` script object will not exist, and will present the "undefined is not an object" script error when attempting to use it (if the "Show script errors" option is also enabled). iCloud variables will also not sync in either direction if this settings option is not enabled. Within the iCloud state settings, you can also view any existing state variables that are currently stored in iCloud, and also clear all state variables, removing them from your iCloud storage.

Script Handlers

Handle socket disconnects/re-connects with script

You can include script to run automatically when TouchControl detects a broken socket with a

remote device, and also when TouchControl successfully re-connects with the remote device. Simply include the following JavaScript in any button in your layout that gets executed prior to using a remote connection (such as an auto-exec on load button):

```
_local.tcDisconnect=myDisconnectFunction; // executes when socket connection is broken  
_local.tcReconnect=myReconnectFunction; // executes when socket connection is re-  
established
```

The above functions are defined on the `_local` script object, allowing you to define different connect/disconnect scripts for each of your activities if you wish.

Then include the specified function(s) in any script library, as follows:

```
function myDisconnectFunction(button, host, port) {  
    /* your code here */  
    /* executes when socket breaks */  
    /* button = name of button executed */  
    /* host = IP address of remote device/host */  
    /* port = port number of remote device/host */  
}
```

```
function myReconnectFunction(button, host, port) {  
    /* your code here */  
    /* executes when socket re-connects */  
    /* button = name of button executed */  
    /* host = IP address of remote device/host */  
    /* port = port number of remote device/host */  
}
```

Handle iPad rotation with script

You can include script to run automatically when TouchControl detects that an iPad is rotating or has rotated to a new orientation. You may specify different script to run on rotation for each individual activity in your config. Simply include the following JavaScript in any button in your layout that gets executed prior to rotating the device (such as an Auto Exec on load button):

```
_local.tcOrientationPrep=myOrientationPrepFunction; // executes just before rotation  
_local.tcOrientationChange=myOrientationChangeFunction; // executes just after rotation
```

The above functions are defined on the `_local` script object, allowing you to define different orientation scripts for each of your activities if you wish.

Then include the specified function(s) in any script library, as follows:

```

function myOrientationPrepFunction(newOrientation) {
    /* your code here */
    /* newOrientation = "portrait" or "landscape" */
}

function myOrientationChangeFunction(newOrientation) {
    /* your code here */
    /* newOrientation = "portrait" or "landscape" */
}

```

Combine this feature with the ability to move and re-size buttons via script to dynamically create different portrait and landscape layouts, and automatically switch between them when the iPad is rotated. This is an iPad-only feature.

Global Watchers

“Global Watchers” allow buttons and labels to automatically update their displayed text based on the current value of `_global` variables set by any of your scripts. To use this feature, simply set a button’s alternate Text value or a label button’s value (in the button config panel) to include the dynamic replacement tag for the desired `_global` variable.

For example, to have a button automatically update its displayed text based on the value of the variable `_global.currentVolume`, set the button’s Text field to `%currentVolume%`. Any time any of your scripts update that `_global` variable, any buttons or labels using that dynamic replacement tag will automatically update to the new value, so you don’t have to worry about updating them yourself in your script. The button or label text can contain only the dynamic replacement tag, or can include other strings along with the dynamic replacement, for example: **Volume: %currentVolume%**.

An “Image” field is also available in the alternate text area of the button config panel that allows you to use the global watcher feature to dynamically update button images. Just enter a dynamic var replacement string (such as `%toggleImage%`) in this field to dynamically update a button’s image based on the value of a global variable (e.g. `_global.toggleImage` for the example shown here). The contents of the `_global` var just needs to contain an image path/name formatted the same as when using `_setImage()` (something like `MyButtonPack/myToggleImage.png`, for example). Same rules apply as the text watchers – the image will be updated to match the global var on activity load, when the var changes, and when returning to the activity after linking off to another activity.

“Watcher” fields are also available on the slider and spinner config panels. Again, enter a dynamic variable replacement string and you can dynamically update the selected value for sliders and spinners based on the value of a `_global` variable.

If a dynamic replacement tag is used, but the referenced `_global` variable does not exist, the button or label will display the dynamic replacement tag in your activity. Therefore, you should ensure that your referenced `_global` variables exist, most likely by initializing them in a script library or an auto-exec on load button.

Global watchers will automatically update their text based on the `_global` variable's value when the activity first loads, when the `_global` variable is updated, and when returning to an activity after linking to another activity. This is useful, for example, if you go to another activity to change the channel or adjust the volume, and want to display the newly selected values on the original activity without having to kick off script to manually update the UI. A subsequent `_getText()` of a watcher will return the updated text based on the `_global` replacement.

NOTE: `_local`, and `_state` variables are also included in this processing, just as they are when using the other current dynamic variable replacement features in the app.

NOTE: You can also use `%0x:varName%` in global watchers if you wish to automatically convert the value of `varName` to a Hex value. Note that the value of `varName` must be convertible to a decimal (number) value.

[Apple Watch](#)

Activity configuration

To create an activity for Apple Watch, use the activity config panel (arrow next to activity name drop down list) and change the activity type selection to "Watch". After selecting the watch type, the "Style" list will be available to select the layout style for the activity. The available styles are:

- List (1, 2, or 3)
This generates a vertically scrolling list of buttons on the watch face. The number associated with the list style indicates the maximum number of buttons that can be placed on each row of the list. Individual rows in styles 2 and 3 can contain fewer buttons, but not more than the style number indicates. If buttons are omitted from rows in style 2 or 3, the adjacent buttons will grow horizontally to fill in the available space, allowing for varying numbers of buttons per list row.
- Grid
This style allows for up to nine buttons placed on the watch face laid out in a 3 X 3 grid. If buttons are omitted, the adjacent buttons will grow to fill in the available space. If all buttons in a horizontal row are omitted, the buttons above and below will gravitate toward the center of the watch face to fill in the available space.
- Stock layout styles
The remaining styles are stock layouts provided by TouchControl with pre-determined layouts and included button images.

See "Apple Watch interface styles" below for images of the various layout styles. The numbers at each button location are for reference only and not included on the actual watch activities.

Once you have completed designing your activity, simply refresh the config in TouchControl on your iPhone to automatically transfer your watch activity to your Apple Watch. You may alternately use the refresh menu option in TouchControl on the watch to force your iPhone to refresh its config from TouchControl server, also automatically transferring all watch activities to the watch.

Activity background

You do not need to select a background for the activity in the designer, as the background of the activity on the watch will, by default be plain black. You can change this to use either the default blue gradient background used in the TouchControl iOS app, or you can use your own background image. To alter the activity background, right-click on the activity background in the

designer and select "Watch background". This will open a text box allowing you to enter the background you would like to use. Enter "default" in this field to use the default blue gradient from the iOS app, or enter an image name (including button pack button file name - i.e. myButtonPack/myImageName.png).

Note that you can add a background to the activity in the designer if you wish simply for design purposes (to increase the size of the activity design panel, for example), but it is not required or used by the watch.

Default button background

By default, all buttons on an activity will use a light gradient background, stretched to fill the size of the button. You can change this default for an activity by right-clicking on the activity background in the designer and selecting "Watch button background". Other selections include "Dark" (a dark gradient), "Blue" (a blue gradient), and "Transparent" (for a completely transparent button).

Watch Button configuration

Currently TouchControl watch activities support the following types of buttons:

- IR
- Command
- HTTP Request
- EventTrigger
- AutoHotKey
- Global Caché
- IRTrans
- Script
- Macro
- Label

Due to the networking and scripting limitations of the Apple Watch, all button commands and script are executed via TouchControl running on the watch's paired iPhone. This requires your watch to be able to communicate with your iPhone via [Bluetooth and/or WiFi](#). TouchControl does not need to be active on the paired iPhone to use TouchControl on the watch, as the watch communication framework will automatically launch TouchControl and execute the button commands in the background. Note that the first button command issued may incur a delay as TouchControl is started on the iPhone, but subsequent commands should execute

immediately.

To add buttons to the watch layout, simply drag and drop them on the activity designer panel as you would with any other TouchControl activity. The position of the buttons on the designer panel is not important, as the location of each button on the watch face will be determined by the watch layout style chosen, and the index specified for each button.

Each layout style has a specific order for the buttons added to the activity (see the watch layout images below). The placement of buttons at each specified position is controlled by the "WatchIndex" property assigned to each button. To set the WatchIndex property, right click on each button on your layout and select "WatchIndex". This will open a dialog allowing you to specify the desired WatchIndex for the given button, or remove the WatchIndex property. The WatchIndex option on the popup menu will also display the current index value assigned to that button - or blank if no index has been assigned. WatchIndex values start at 1 and increment up to the maximum number of buttons allowed for the given watch layout. Note that list layouts can accommodate any number of buttons.

Adjusting layouts

You can adjust any of the base layouts by omitting buttons at various indexes, or by adding "placeholder" buttons that retain their assigned space but do not render on the watch face. Omitting buttons at any given index will allow the buttons on either side to grow to fill the space available from the missing button. This allows you to create rows with varying numbers of buttons with varying sizes. "Placeholder" buttons are added by including a button with a given WatchIndex, but then disabling the button in the designer (right-click on the button and de-select the Enabled setting). Adding "placeholder" buttons will retain the assigned space for the given index on the given layout, but will not render the button on the watch face. This also allows you to create rows with varying numbers of buttons with "custom" spacing. Note that Watch OS does not allow for positioning of buttons at specific pixel locations on the watch screen. All button placement is based on the horizontal and vertical flow of buttons on the screen. TouchControl allows for the horizontal flow of buttons on any given row, and the vertical flow of rows based on the presence or absence of buttons on any adjacent row.

Hidden and placeholder buttons

Hidden buttons can be added to the watch layout by adding the button to the activity designer panel, not setting a WatchIndex for the button, and disabling the button by un-selecting its Enabled setting (on the right-click popup menu). Hidden buttons can be used for auto exec (see below) or scripting purposes (such as [#] execution of buttons via script). Hidden buttons do not render in any fashion on the watch face.

Placeholder buttons can be added to the watch layout by adding the button to the activity designer panel, setting a WatchIndex for the button, and disabling the button by un-selecting

its Enabled setting (on the right-click popup menu). Placeholder buttons will not render on the watch face, but will reserve the space allocated for the specified WatchIndex on the given layout. This allows you to have greater control over the location of buttons and the overall layout of the activity.

Button text

Watch buttons can show or hide their text just as with an iPhone or iPad activity, by right-clicking on the button in the designer and selecting Text - Show Text/Hide Text. You can also set the color of the button text in this same location. Note that text size cannot be set for watch buttons. The text on watch buttons will be rendered at a default size, and adjusted dynamically as needed to fit the button on the watch screen. Watch buttons can be provided custom text on the button config panel, but cannot render custom HTML. Button repeat and timer settings are also ignored on the watch.

Button images and icons

Watch buttons can be assigned background images and icons in the designer, just as with an iPhone or iPad activity. By default, any image you assign to a button will be placed on top of the default button background (described above). To use only your selected image for a given button, right-click on that button in the designer and select "Watch button background". This will allow you to override the default button background for just that button, and selecting "Transparent" will render the button with only your image. Button icons will be placed on top of your selected button image, if one exists, or directly on the default button background if not. If a button contains both an icon and text, the icon will be placed to the left, and the text to the right for all layout styles except the Grid style, for which the icon will be placed above and the text below. If a button contains an icon but no text, the icon will be centered on the button.

Please note that when adding both activity background images and button images/icons, you should use images that are relatively small and sized appropriately for the watch. The images used may be resized/stretched on the watch to fit the button that they are assigned to in the watch interface layout, so keep that in mind when creating and choosing a button image. Each of these images will be transferred to the watch and stored there for continued use. So be aware that using overly large images will require more space on the watch, and could impact both app performance and watch battery life.

Auto Exec

Watch buttons can be configured for Auto Exec on load, on resume, and on exit. Auto exec buttons can be either visible or hidden on the watch.

Button feedback

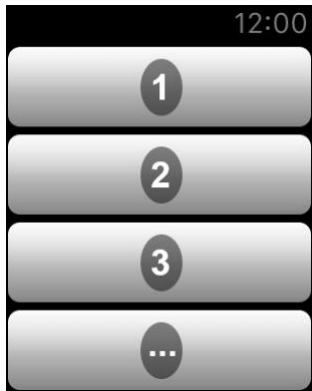
Watch buttons can be configured for feedback, allowing for normal feedback scripting executed on the paired iPhone, as well as basic feedback functionality on the watch. On-watch feedback functionality is limited to basic button text, image and icon processing as documented [here](#).

Watch Haptics

Watch haptics provides feedback in the form of vibrations when certain events occur within the app. To enable haptics in TouchControl, open settings within TouchControl on the paired iPhone, scroll down and select Tools - Apple Watch, and select the type of haptic feedback you would like to receive from TouchControl on the watch.

Apple Watch interface styles for TouchControl

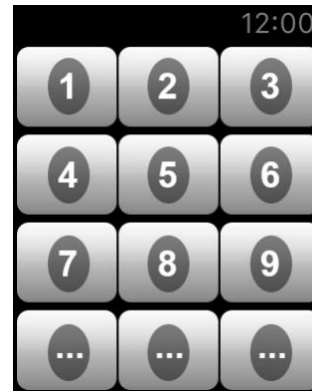
List 1



List 2



List 3



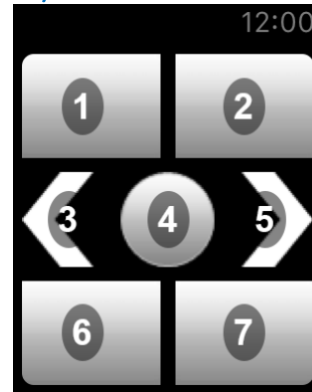
Grid



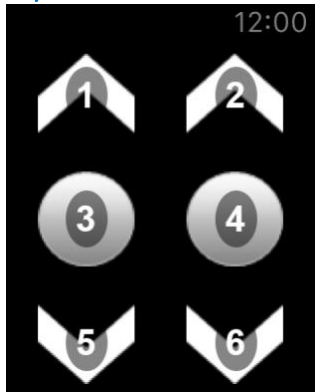
Style 1



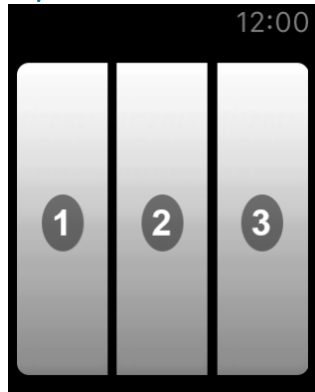
Style 2



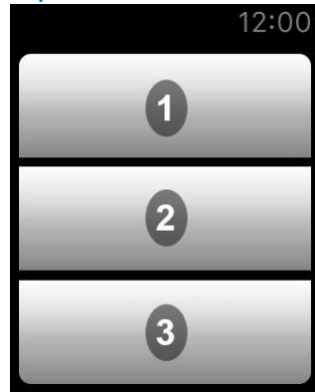
Style 3



Style 4



Style 5



[Miscellaneous Topics](#)

The following topics dive deeper into several key TouchControl features. These topics are in no particular order. See the table of contents for a quick reference of what's here.

Sizes

In TouchControl on your iOS device, open Settings (the gear icon in the upper left on the navigation bar), then scroll down and select the “Sizes” option under the “Tools” section. This will provide an alert message that shows various sizes for UI elements on the current device, such as screen size, top bar height, @1x/@2x/@3x background image sizes (for both standard and full-screen activities), and safe areas (for full screen devices – i.e. the iPhone X and later, etc.). Use these sizes to create your activity background images and position your buttons, if desired. When executed on an iPad, this will report the sizes based on the current orientation of the device (portrait or landscape).

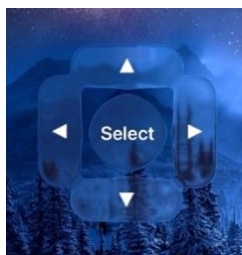
Liquid Glass

TouchControl v13+ and TouchControl Server for Mac v5+ and for Windows v13+ introduced Apple's Liquid Glass material, available on devices running iOS and macOS 26+. TouchControl now includes a new Liquid Glass theme, available in the Theme settings in TouchControl on your iOS devices, displaying location and activity buttons using glass material, as well as other user interface elements like navigation bar buttons, etc.

Your own buttons can also have Liquid Glass backgrounds on your layouts by adding hot-spot buttons (buttons with no background image), and then selecting the Liquid Glass button background color using the **Image -> Set background color** option on the button popup menu on your layouts. Slider buttons do not have background colors, so to use glass material on slider buttons, set their built-in **"useGlass"** property to **"yes"** or **"true"**, and make sure the **"sliderBarLeft"**, **"sliderBarRight"**, and **"sliderThumb"** built-in properties are not set. TextField buttons do not support this type of background. To give a TextField button a Liquid Glass background, simply create a group button, give the group a Liquid Glass background, add the TextField button with a clear background to the group, and size the group to the size of the TextField button.

Note that when using the Liquid Glass theme or Liquid Glass backgrounds on your buttons, the iOS glass material works best if there is something interesting behind the button for the glass effect to interact with. Therefore using a custom home screen background or custom background image for your activity is recommended when utilizing Liquid Glass.

Group buttons can be configured as glass containers by setting the built-in **GlassSpacing** property on the group. With this property set to a number greater than zero, when buttons with the Liquid Glass background are added to the group, the buttons will interact with and react to each other and morph into a single object when they are within the spacing distance from each other. Below is a screen shot showing buttons morphing into one object inside a glass container group.



IMPORTANT: Adding non-Liquid Glass buttons to a group using GlassSpacing can produce unexpected results. It is recommended to use the GlassSpacing property on groups that **ONLY** contain buttons with Liquid Glass backgrounds!

Background Slideshow

If you use TouchControl in an always-on setting, for example if you use a client or listener button to receive and process network requests/messages, use TouchControl's built-in web server, use TouchControl as an always-on keypad/controller, etc., you can configure an activity to display a rotating slideshow of background images as a method of adding some visual interest, and helping to protect your device screen from burn-in.

Slideshow images are accessed via the photo library on the device TouchControl is running on, and must be stored within a photo album in the Photos app. All images located in the desired photo album will be used for the slideshow and continuously rotated through in an indefinite loop – as long as the activity is showing on your screen, or until you stop it. TouchControl can access albums/images that are created/stored on the device, synched from iTunes, or available in iCloud. When images from the selected album are loaded into the activity background, they are scaled to fill the activity/device screen. Slideshow images transition using a fade/cross-dissolve animation.

To enable the background slideshow for an activity, use the following script:

```
_backgroundSlideshow = true;  
_slideshowInterval = 60; // number of seconds to show each image – default is 300 (5 mins)  
_slideshowAlbum = "My Photos"; // the name of a user-created album in your Photos library
```

Note that the **_slideshowAlbum** name must be entered exactly as shown in the Photos app, including proper case and any punctuation.

To start a slideshow immediately when an activity loads, add this script to an Autoexec on Load button. When starting the slideshow on activity load, the original background image for the activity (set in the designer) may show briefly when the activity first loads and before the slideshow starts, for the duration of the slideshow transition animation. If this is an issue, you could set the background in the designer to a solid black (or other desired color) image of the proper size for the activity.

To stop a running slideshow, use the script:

```
_backgroundSlideshow = false;
```

Or, to toggle the slideshow on/off, use:

```
_backgroundSlideshow = !_backgroundSlideshow;
```

When the slideshow is stopped, the last shown slideshow image will remain as the activity background until the slideshow is re-started, or the activity is closed and re-opened. To change the photo album being used for the slideshow, execute the script:

```
_slideshowAlbum = "New Album";
```

To change slideshow interval, execute script that updates the **_slideshowInterval** variable.

The slideshow will pause if TouchControl is sent to the background and should resume when brought back to the foreground. Exiting the activity will terminate the slideshow, and re-launching the activity will restart the slideshow from the beginning. Slideshow images are shown in the order of image creation date as determined by the iOS photo library.

A tool is available to show you the names of the available photo albums in your photo library on the current device. Open Settings (the gear icon in the upper left on the navigation bar), then scroll down and select the “**Script Settings**” option under the “**Tools**” section, and tap on the “**Show Photo Albums**” option. This will display an alert on your screen showing the user-created albums found on your device (“smart” albums are not included in this list and are not available for use in a slideshow). Remember to enter the album names exactly as shown in this alert, including proper case and any punctuation.

You can also get an array of the available album names in your own script using the script variable:

```
_photoAlbums
```

You could use the album names, for example, to populate a spinner or table for album selection prior to starting the background slideshow.

To use the same photo album on multiple devices, make sure the album is synched to all of your devices via iCloud.

Optional: If desired, you can hide all of the buttons on an activity using the **_hideActivityButtons()** script helper function. Execute this from a button/hotspot somewhere on your activity to hide your layout and give full attention to the slideshow background images. Simply tap anywhere on the activity background (slideshow image in this case) to re-show all of your activity buttons.

Activity & Device Sharing

TouchControl Server for macOS and Windows includes a feature that allows you to share your configuration information with other TouchControl users via the TouchControl web site. You are able to share individual activities or devices that other users can download and import into their configurations. [See the Sharing page](#) on the web site for more information.

Network PING

You can "ping" IP addresses or hostnames directly from TouchControl button script, allowing you to determine if a network-attached device or service is available (powered on, etc.) before you send a command. This can be very useful, as an example, for devices that use a toggle power command, so you can determine ahead of time whether to send an on/off command, and avoid possibly getting your device out of sync with other devices in your environment. Use the following command in your script to execute a ping:

```
result = _ping('address', timeout)
```

- **address** (string - required): The destination IP address or hostname (a specific port should not be included).
- **timeout** (integer - optional): The number of whole seconds the ping process will wait for a response from the device/service. If **timeout** is omitted, the default timeout will be **2** seconds. Partial seconds are not supported.
- **result** (Boolean): True or false indicating whether or not the device/service returned a valid response to the ping within the timeout period.

Example 1:

```
var pingResult = _ping('192.168.1.1', 1)  
if (pingResult) {  
    // device is on  
    // your code here...  
} else {  
    // device is off  
    // your code here...  
}
```

Example 2:

```
if (!_ping('192.168.1.100')) {  
    // device is off  
    // your code here...  
}
```

Connectable

Use the `_connectable()` script function to test the availability of a specific port on a device on your network. This can be useful, for example, if a device will respond to a PING even though it is technically "off", but a specific port will be unavailable until the device is turned "on". Use the function as follows:

```
var result = _connectable('192.168.1.100', 5000);
if (result) {
    // port is available
    // your code here...
} else {
    // port is unavailable
    // your code here...
}
```

`_connectable()` will return true if connection was successful, or false if it was not. No timeout is applied by default. To set a timeout, use:

```
_connectable('192.168.1.100', 5000, 1);
_connectable('192.168.1.100', 5000, 3);
...etc.
```

The timeout value is always in full seconds. With a timeout set, if the device responds immediately, the function will return immediately and will not wait for the timeout.

LocationManager

Use the **LocationManager** script object (**_locationManager**) to retrieve your device's current location via script. LocationManager will retrieve the current location latitude and longitude values via the **getCurrentLocation()** function, and return the values via a script callback function as follows:

```
_locationManager.getCurrentLocation(  
    function(lat, long) {  
        _global.locationLat = lat;  
        _global.locationLong = long;  
    }  
);
```

The callback function is required because retrieving the current location is an asynchronous process which can take up to a few seconds to return. The **getCurrentLocation()** function will return immediately, and the supplied **function(lat, long){...}** will be called in the background by TouchControl when the values are available. This example stores the values to **_global** variables for later use. You may also return feedback from this callback function, such as **return ['#]MyButton'**; for additional control and functionality.

Device Battery Monitor

The `_batteryMonitor` script object allows you to get the current device battery charging state and level, and optionally turn on battery monitoring to be notified when the charging state and/or level changes. You can either access the state and level info directly or supply a callback function to run whenever the state and/or level changes.

The `_batteryMonitor.state` property returns an integer as follows:

- 0 = unknown
- 1 = unplugged
- 2 = charging
- 3 = full

The `_batteryMonitor.level` property returns a float value indicating the current charging level.

See the following script examples using Emoji symbols to display battery state on the screen:

```
var batteryStateIndicators = ['🔋', '✖', '🔌', '🔋']; //unknown, unplugged, charging, full
```

To access the battery info directly:

```
_setText('BatteryStateLabel', batteryStateIndicators[_batteryMonitor.state]);  
_setText('BatteryLevelLabel', (_batteryMonitor.level * 100).toFixed(0) + '%');
```

To execute a callback function when the battery state and/or level changes, start the monitor and pass it a function (TouchControl will dynamically call this function and pass the current state and level as parameters):

```
_batteryMonitor.start(  
  function(state, level) {  
    _setText('BatteryStateLabel', batteryStateIndicators[state]);  
    _setText('BatteryLevelLabel', (level * 100).toFixed(0) + '%');  
  }  
);
```

To stop the battery monitor:

```
_batteryMonitor.stop();
```

Speech Synthesis

Use the `_speak()` script function to generate text to speech from within your own activities, and optionally specify a specific voice for the speech synthesis. Use the function as follows:

```
_speak(text to speak, voice ID/code)
```

Examples:

```
_speak('Hello world'); // use the TouchControl default voice  
_speak('Hello world', 'en-US'); // use the default voice for language/country code  
_speak('Hello world', 'com.apple.ttsbundle.Samantha-compact'); // use a specific voice ID
```

An additional function provides a list of all available iOS speech synthesis voices. You can use this function to provide a selection list of voices, or more likely, simply use it once to determine which voice you wish to use, and then provide the desired voice ID or code in your use of the `_speak()` function. Note that using a different voice will not *translate* the given text to the given language. This simply provides a different *persona* related to the given language/country. To get the list of voices use:

```
var voices = _voices();
```

This function provides an array of **Voice** objects containing the information needed to provide to the `_speak()` function (along with some other helpful info). The properties of each Voice object are as follows:

id: a unique voice identifier
code: a country-specific language code
language: the native language for the voice
country: the name of the country for the voice
name: the name of the persona used for the voice

You will find there are multiple ID's/personas for several of the available languages/countries. Passing a voice **ID** in the `_speak()` function will use the specific voice/persona assigned to the ID. Passing a voice **code** will use the default voice/persona for the given country code. Not passing an ID or code in the `_speak()` function will use the default US English voice as follows:

```
voice.id = 'com.apple.ttsbundle.Samantha-compact';  
voice.code = 'en-US';  
voice.language = 'English';  
voice.country = 'United States';  
voice.name = 'Samantha';
```

NOTE: Audible speech synthesis when the device is in silent mode requires iOS 13 or above.

A sample activity is provided on the [Download](#) page that demonstrates using `_voices()` to get the list of available voices, and using `_speak()` to synthesize dynamic text using a selected voice. You may wish to use this activity to determine which voice you wish to use, and then just pass the desired voice ID or voice code in your use of the `_speak()` function.

Siri Integration

TouchControl is integrated with Siri on your iOS device, allowing you to create shortcuts for your favorite buttons and execute them with your voice. TouchControl buttons available for Siri Shortcuts are also exposed through Search results, allowing you to search for buttons by name outside of TouchControl and easily execute them.

Siri/Search integration is available on devices running iOS 12 or later, and requires TouchControl 10 or later, as well as TouchControl Server version 10 for Windows, or version 2 for macOS. All primary button types, as well as macros and script buttons can be enabled for Siri and Search.

To enable a button for Siri, add the button to an activity layout in the TouchControl Server designer, then right-click the button and choose the **Shortcut** menu option. This will display three configuration options:

- **Enable** - this enables/disables Siri for the button.
- **Require confirmation** - when enabled, this requires that you respond to a confirmation prompt before the button is executed.
- **Alert on success** - when enabled, this triggers the display of a success message on the screen that you must manually dismiss.

Once you have configured the desired buttons for Siri in the server designer, simply refresh the configuration to TouchControl on your device and the buttons are immediately available from Siri/Search. To disable a button from being available for Siri, disable the Shortcut option for the button in the server designer and refresh your config to your device.

Performing these actions makes your buttons available as actions for Siri Shortcuts. To create a Siri Shortcut using a TouchControl button, you'll need to use the Shortcuts app on your device to create the shortcut, set the TouchControl button action, and provide a phrase to trigger the shortcut when talking to Siri. You'll also use the Shortcuts app to delete any shortcuts that you have created that you no longer need.

A **Shortcuts** option also exists in the **Tools** section of TouchControl settings on your device that will display all buttons currently enabled for Siri. You may disable buttons from being available as Siri actions by removing them from this location. This is useful if the activity that you enabled a button in no longer exists and you want to remove the button as a Siri action. Be aware that if you remove a button in this location but it is still enabled for Siri in an existing activity, it will be re-added the next time you refresh your config.

When a button is triggered via Siri or Search results, TouchControl will be activated on your device to execute the button. The activity that you enabled the button in will be loaded into memory in the background (not visible) and the button will be executed within the context of that activity. Make

sure that any other buttons required for execution of the desired button exist on that layout as well - just as if you were executing the button normally from a remote screen in TouchControl.

Any activity can contain buttons enabled for Siri, but you may wish to create a separate activity dedicated to all of your Siri-enabled buttons. This could be more efficient, as loading a large activity in the background to execute a single button via Siri could require more background memory resources than potentially required for that single button. Again, just make sure that any other buttons/elements required to execute the button exist on that activity as well. A dedicated shortcuts activity can also be hidden in your config if you wish.

[IFTTT Webhooks Integration](#)

Easily generate **IFTTT Webhooks** service requests in TC via HTTP Request buttons using either the Webhooks 3-value method or using a JSON payload. [See the IFTTT Webhooks documentation](#) for information about those two Webhooks service features. To enable this feature in TouchControl, access TC Server Settings - Preferences, and check the "Enable IFTTT Webhooks in HTTP Request buttons" option, and enter/paste your personal Webhooks key (found in IFTTT) in the provided "Key" field. Then, when configuring HTTP Request buttons in TC Server, check the "IFTTT" option in the upper right to display the Webhooks trigger field, and select the Values or JSON payload type to display the appropriate fields. Note that when using the IFTTT Webhooks option, the button's selected host IP address (configured in TC Server Interface Manager) must contain "maker.ifttt.com", and the port must be 443 (Webhooks requests use the HTTPS protocol). Again, see the IFTTT web site for full documentation on the Webhooks service.

External Mousepad

Gesture Pad Mousepads can be configured to act as either "internal" or "external" mousepads. "Internal" mousepads (the default) perform as normal mousepads, controlling the mouse on your TouchControl Server PC (Windows server only), using internal commands that are not exposed. "External" mousepads, on the other hand, execute your buttons as your finger(s) move over the mousepad, gaining access to the X and Y coordinates of the mouse movements, as well as two-finger scroll interactions, one- and two-finger taps, and pinch/zoom gestures. This could allow you to control your own devices (such as a new "smart" TV with cursor control) using natural mouse-like movements.

When you configure a Gesture Pad and select the "MousePad" option, the mousepad may be further configured as "Internal" or "External" via radio buttons available on the Gesture Pad config page. As described above, "external" mousepads use buttons on your activity to execute the desired commands. To accomplish this, TouchControl will look for specifically-named buttons on your layout that it will execute when the various mousepad gestures are performed. This is similar to enabling the hard volume buttons on your device by adding buttons named `_vol+` and `_vol-` to your layout, or enabling device motion sensing by adding the `_deviceMotion` button.

Following are the button names that TouchControl will look for and the functions that they provide. The buttons names must match exactly, including the leading underscore and proper case.

`_mouseMoveButton` - this button handles primary mouse movements, and is continuously executed as you drag your finger over the mousepad. This button will have access to two new script variables: `_mouseMoveX` and `_mouseMoveY`. Those variables contain the distance in pixels that the mouse moved since the last time the button was executed. This distance, and thus the number of times this button is executed, will vary depending on how fast you move your finger. A slow swipe will usually result in the button being executed about every 3px of movement, but can jump up to over 100px+ on fast swipes, and even more if you have "Use Swipe Velocity" turned on in the gesture pad. Also, the X value will be positive when moving right and negative when moving left. Likewise, the Y value will be negative when moving up and positive when moving down. You can of course convert all the values to positive using `Math.abs()` in script if you need.

- `_mouseLeftButton` - this button handles single, one-finger taps, which is equivalent to left mouse clicks on an internal mousepad.
- `_mouseRightButton` - this button handles single, two-finger taps, which is equivalent to right mouse clicks on an internal mousepad.

- `_mouseHScrollButton` - this button handles two-finger horizontal swipes, which is equivalent to horizontal scrolling on an internal mousepad. This button has access to a new script variable: `_mouseScrollX`.
- `_mouseVScrollButton` - this button handles two-finger vertical swipes, which is equivalent to vertical scrolling on an internal mousepad. This button has access to a new script variable: `_mouseScrollY`.

For the above scroll buttons, TouchControl determines if you swiped farther horizontally or vertically and executes the appropriate scroll button (but only one or the other, never both at the same time), and populates the associated variable accordingly with the number of pixels traveled since the last execution (positive or negative).

- `_mouseZoomInButton` - this button handles the pinch-out gesture, which is equivalent to zoom in (make things bigger) on an internal mousepad.
- `_mouseZoomOutButton` - this button handles the pinch-in gesture, which is equivalent to zoom out (make things smaller) on an internal mousepad.

The above zoom buttons are executed multiple times as your fingers perform the pinch gesture on the mousepad. There are no variables associated with these buttons. Simply perform whatever function you wish from these buttons as the pinch is occurring.

The buttons with the above names can be any type of “normal” button (i.e. not sliders, spinners, gesture pads, listeners, labels, text fields, etc.). Basically, you can use any type of button that executes a command, script, or link when you tap on it (including a macro). So, for example, they could be HTTP Request buttons that send commands directly to your TV, or they might be Script buttons that first “massage” the raw numbers and then execute other buttons using `return '[#]some button'`;

A sample activity can be found on the download page that demonstrates all of these buttons, logging the events to a text field to display the x/y values in real time.

[Simple Service Discovery Protocol \(SSDP\)](#)

Simple Service Discovery Protocol (SSDP) is a network protocol that allows for the discovery, configuration, and control of devices on a network. Using SSDP allows TouchControl to automatically find and control devices (receivers, televisions, set-top boxes, etc.) that support SSDP without the need to specifically configure IP addresses, ports, etc. Devices that support SSDP for discovery also typically allow control via HTTP requests. This makes it very easy to find and control devices with TouchControl.

The SSDP process starts with an M-SEARCH request being broadcast to a specific IP address and port on a given network. All devices that support SSDP should be listening for search request messages on that IP/port, and will respond directly to the device that sent the search request with information about the discovered device. Many times, the discovered devices will generate multiple responses containing information about the various services that are available to interact with on the device. All of the responses should contain a "Location" parameter that specifies a URL that can be used to get more information about the specific service. The information available from the Location URL is specific to the device/service, so please see your device documentation for more information.

SSDP requires the use of specific protocols, network addresses and ports as follows:

- The M-SEARCH request must be sent via the UDP protocol to the multicast IP address 239.255.255.250, on port 1900. In TouchControl you would use an EventTrigger button to generate this request, with an interface host that specifies that IP address and port, and the UDP protocol.
- Responses to the search request are sent back to the specific IP address and port that sent the request. This requires TouchControl to also be listening for the responses on the same port used to send the request. In TouchControl you would use a Feedback Listener button to listen for these responses. The interface used for the Feedback Listener should specify no multicast address (which results in your device's local IP address), and use port 0 as the LAN port. By specifying port 0, TouchControl will automatically ensure that it listens on the same local port used to send the search requests. Note this is NOT port 1900. Search requests are sent TO port 1900, but are sent FROM an internally specified port that TouchControl will manage.
- Devices that support SSDP also periodically send messages out to the network advertising that they are available. These "notify" messages are also broadcast to the multicast IP address 239.255.255.250 on port 1900. To receive these messages in TouchControl, create a Feedback Listener button using a Feedback Listener interface host with that IP address and port. Note that an M-SEARCH request DOES NOT need to be sent in order to receive these notifications. This would be useful in TouchControl, for

example, if you have already discovered a device, but want to be notified at a later time if the device's IP address has changed.

A [sample activity](#) is available on the download page that demonstrates using SSDP to discover devices on your network. Please see the READ-ME file located in that activity export for more information on how to use it.

Another [sample activity](#) is also available that uses SSDP to discover and control DirecTV receivers on your network, to give you a complete example of both discovering and controlling devices. Please see the READ-ME file located in that activity export for more information on how to use it.

[Integrated Global Caché IR Database](#)

Global Caché's new "ControlTower" online IR code database has been integrated into TouchControl! You can now search for and import ready-to-use IR codes directly into TouchControl Server while designing your buttons and layouts. Individual codes can be added to existing GC buttons via the "Find" feature when [configuring GC buttons](#), or partial or entire code sets [can be imported, automatically generating buttons on the fly](#).

Device Motion Sensing

You can configure TouchControl to execute buttons as you tilt, roll, and rotate your device in your hand. This feature uses the gyroscope and accelerometer built into your device to detect device movements, and provides the raw data to your buttons via script variables that you can use to perform whatever functions you choose via scripting.

Data is passed via the following three script variables:

`_motionPitch` - a floating point value indicating the "pitch" of the device if you rotate it end-over-end

`_motionRoll` - a floating point value indicating the extent that the device is "rolled" from side to side in your hand

`_motionYaw` - a floating point value indicating the extent that the device is rotated on a flat plane around in a circle

Motion sensing is triggered by the presence of a button on an activity named "`_deviceMotion`". (This is very similar to how buttons named "`_vol+`" and "`_vol-`" trigger the use of the hard volume buttons on the iOS devices.) This allows you to use any type of button you wish to process the motion events, with the button name being the only requirement. When TC detects a "`_deviceMotion`" button at activity launch, it starts monitoring the device motion events and provides the raw motion data via the above script variables, available to any button on the layout, and then executes the `_deviceMotion` button each time the device motion data is sampled.

The default frequency (interval) for sampling the device motion events is .1 (10 per second). This can be adjusted by setting the "MotionInterval" property on the `_deviceMotion` button to some other value. (Right-click on the `_deviceMotion` button and select "Properties...", then select "MotionInterval" from the list of available property names.) Note that the MotionInterval should be set as high as possible for whatever the specific use case is, as the `_deviceMotion` button gets executed at each interval, and can result in excessive overhead/lag depending on the type or amount of processing triggered by the execution of the `_deviceMotion` button.

Beyond this, it is up to you to determine what is done with the device motion data. You could execute buttons to scroll through a channel list on your TV/set top box, brighten or dim lights, control the mouse on your computer, etc. On the Download page you will find a [sample "AirMouse" activity](#) that will allow you to control the mouse on your PC by waving your device around in the air. Take a look at the script within the `_deviceMotion` button in this

activity. This script contains comments which explain what the button is doing and how it is using the pitch/roll/yaw values. The MouseMove and MouseScroll buttons are Command-type buttons that are executed by `_deviceMotion` to send the mouse commands to the server based on the values calculated within the `_deviceMotion` script. There are several variables used within this script to control the speed & smoothness of the mouse movements. Play with these values to find the best experience for your particular situation and preferences.

There is also a hidden text field on the AirMouse layout named "MotionField" that you can enable, as well as a line of script in the `_deviceMotion` button's script that you can un-comment to display the raw pitch/roll/yaw values in real time. This will give you an idea of what the raw motion values look like. Although, as the comment in the code indicates, doing this can cause significant mouse lag (see the above discussion of frequency interval and `_deviceMotion` processing overhead), so you'll only want to do this briefly for testing/debugging. If you have any questions or problems with this feature or sample activity, please don't hesitate to contact support@touchapptech.com.

[Designer Device & Button Search \(Windows server only\)](#)

Device Search

To easily locate a device in the Available Devices list, simply select any entry in the list and start typing any part of the desired device's name. This will immediately filter the list to only the devices containing the typed string (which can appear at any location in the found devices' names). To filter the list to only devices starting with a given character or string, simply press and hold the first character you wish to search for, and the list will immediately be filtered to display devices whose name starts with the entered characters. Once you've found the device you are looking for, just press "Enter" to open the device and display its buttons, or press Ctrl-Enter to add the device to the currently open activity.

Button Search

To easily locate buttons from any device in your configuration, simply select any entry in the "Available Devices" or the "Buttons for selected device" lists and either right-click and select "Find Button...", or just press Ctrl+F. This will open the Find button panel. Simply begin typing any part of a button name in the "Button name" field, and the result list will automatically filter to any buttons whose name contains the entered string. Highlighting and selecting a button in the results list (use arrow buttons or click with mouse to highlight, then click "OK" or double-click or press enter to select) will automatically select that button's device in the devices list, and automatically display and select the button in the buttons list.

Grid Buttons

A unique button presentation feature is available - referred to as a "grid" - which allows you to lay out multiple buttons in a free-flowing, independently-scrolling grid layout within your activities. The grid is configured as a spinner-type button, using the "Buttons" mode, which allows you to add buttons from your configuration to the spinner (or grid in this case), and then selecting the "Display as grid" option in the spinner configuration will trigger the spinner to display as a grid on your iOS device. By default, the grid will scroll vertically within the bounds that you specify when you size the spinner on your layout at design time. When "Display as grid" is selected, you can also select "Scroll horizontally", which will trigger the grid to scroll horizontally within the spinner button's bounds.

An additional requirement for using a "grid" type spinner button is that the buttons that you add to the grid within the spinner button's config panel must also be physically added to your layout at design time. This allows you to set the image for each button, as well as other design-time properties, such as text size, color, Touch Tips, etc. At run-time, a copy of these buttons will be added to any grid that references them on the layout (you can have multiple grids on a given layout, and you may reference the same button in multiple grids if you choose). Therefore, it is suggested that after you add the desired buttons to your layout, you should disable them (right-click and un-check the "Enabled" option), which will hide them in their initial location on the layout, and any button added to a grid will be automatically enabled before being added to the grid. An easier method of doing this could be to add all of the buttons used in grids to a group button on your layout, and then disable the group button, keeping you from having to manage the visibility of each button individually (buttons added to a disabled group will not be visible, as the group's enabled property prevents it).

The order that the buttons are added to your layout (or to a group button on your layout) does not matter, as the order that the buttons are referenced within the spinner/grid button's configuration will determine the order that the buttons are displayed within the grid. Vertically-scrolling grids will layout buttons left-to-right/top-to-bottom. Horizontally scrolling grids will layout buttons top-to-bottom/left-to-right. If a button is referenced within a grid but cannot be found on the layout, a static label will be added to the grid in the location of the missing button, displaying the missing button's name. Simply drag the button to your layout (and configure as desired) to fix this issue.

All grids have a transparent background. If you would like your grid to have a background color or image, you can either place a label button behind the grid and set a background color or image on the label, or add the grid to a group button, and set a background color or image on the group button. Sample activities for both iPad and iPhone demonstrating grid buttons can be found on the [download page](#), and these samples use a group button to provide a background image for the grid.

A new `_scrollTo()` script helper function is available to facilitate programmatic scrolling of grids to specified buttons.

Examples:

```
_scrollTo('MyGrid','first'); //scroll to the first button in the grid named "MyGrid"  
_scrollTo('MyGrid','last'); //scroll to the last button in MyGrid  
_scrollTo('MyGrid','top'); //scroll to the top of MyGrid (same as "first")  
_scrollTo('MyGrid','bottom'); //scroll to the bottom of MyGrid (same as "last")  
_scrollTo('MyGrid',12); //scroll to the button at position 12 in MyGrid (any integer)
```

[Multi-Peer \(Nearby Networking\)](#)

Multi-peer, also referred to as "nearby networking", allows devices to find each other using various networking methods (WiFi or Bluetooth), and share data over those connections. TouchControl uses this functionality to allow you to create activities that share data directly with other devices running TouchControl (peer-to-peer, without a server in between), for whatever data-sharing needs you may have, all managed and controlled behind the scenes by iOS and TouchControl.

The multi-peer feature in TouchControl is exposed through Interface Manager, where you add a multi-peer interface with the required information. To add a multi-peer interface:

- Select "Tools - Settings - Interface Manager", then click "Add New".
- Select the "Multi-Peer" option.
- Give the interface a unique name. This name is only used to distinguish this interface within Interface Manager.
- Specify whether this interface will be a "Sender" or "Receiver". Note that an activity may be both a sender and a receiver, using two different Interface Manager hosts.
- Specify the "Session ID". This is a unique ID used to identify the connection to both the sender and receiver, and is suggested to be short, simple, yet unique. For each "Sender" interface using this unique session ID, there should also be a "Receiver" using this same session ID.
- Click "Save" to add the interface to TouchControl Server.

The multi-peer interface created above is used within your activity by an EventTrigger button. Create an EventTrigger button and select the multi-peer "Sender" interface as the "Host". Any data sent by this EventTrigger button will be received directly by the "Receiver" interface on another device.

To add a receiver to an activity, create an EventTrigger button and select the "Receiver" interface as the "Host". Any data received from the "sender" will be received in the `_feedback` variable accessible by the button's feedback script. You do not need to explicitly "execute" the receiver button each time the sender sends data. The receiver's feedback script will automatically be executed whenever it receives data from another multi-peer sender.

[The Multi-Peer Session](#)

The connection between a multi-peer sender and receiver is based on a session. The sender creates a session and then browses the network for receivers. This is done automatically when the sender EventTrigger button is executed, therefore setting a sender as an auto-exec on load button, or executing the sender from an auto-exec on load button is a good way to start the

process when an activity is loaded. Likewise, a receiver starts advertising its availability on the network when the receiver EventTrigger button is executed, so again, executing the receiver EventTrigger button when the activity is loaded will make this happen automatically.

Once the sender detects a receiver on the network, the sender sends an invitation to the receiver to join the session. In TouchControl, the receiver will be prompted with a request to accept or decline the invitation. If the invitation is declined, the receiver will ignore any additional invitations from that sender for the life of the activity (until you exit and reload the activity). If the invitation is accepted, the sender is then prompted to either accept or decline the receiver's request to join the session. Once both the receiver and sender have accepted their respective requests, the connection is established, and the sender can begin sending data to the receiver.

A single session can include a sender and any number of receivers. The sender EventTrigger button must be executed whenever data needs to be sent to the receivers who are included in the sender's session. The data found in the sender EventTrigger button's command field will be sent to all receivers upon execution of the sender button. Use the dynamic %varname% replacement feature of EventTrigger commands to dynamically update the data sent to all receivers each time the sender button is executed.

The receiver EventTrigger buttons do not need to be manually executed each time a sender sends data. TouchControl will handle running the receiver button's feedback script each time data is received via the multi-peer session. The receiver EventTrigger button's command field may be left empty, and the receiver does not send data back to the sender. Only the feedback script is used on this button.

If you need data to flow both ways (sender to receiver, and receiver back to sender), simply add both a sender and receiver button to an activity, so both clients can act as both sender and receiver.

Multi-Peer Button Properties

EventTrigger buttons have built-in properties that control various aspects of multi-peer communications.

Sender-only EventTrigger Properties:

MPSendDataMode - Set to "Reliable" to force reliable mode (iOS ensures the message is sent). Set to an empty string to set to unreliable mode (the default, no guarantee the message will be sent). Use unreliable when it is not imperative that a message be received by the receiver, such as when additional updates will be sent later anyway, as unreliable mode requires less overhead.

MPAutoAccept - Set to "true" to force the sender to automatically accept all requests to connect from receivers. Set to "false" (default) to force the sender to manually accept the incoming requests (via a popup alert). Receivers must always manually accept an invitation from a sender.

Sender and Receiver EventTrigger Properties:

MPConnectScript - Set to a string containing script that will run when a peer connects to your session (suggestion: add a function to a script library, and set this property to that function call).

MPDisconnectScript - Set to a string containing script that will run when a peer disconnects from your session (suggestion: add a function to a script library, and set this property to that function call).

As a reference, the Scoreboard activity available with the [default configuration](#), and also available on the [download page](#), makes extensive use of the multi-peer feature, including the above EventTrigger button properties.

If you try out this feature and have any problems at all, please contact support@touchapptech.com. I'd be glad to help you get it up and running!

[Activity Locking](#)

If you'd like to either restrict the activity or activities within your configuration that users can navigate to (or from), or if you'd like to present a lock screen requiring a password to unlock (such as after a timeout, similar to the iOS device lock feature), TouchControl's "Activity Lock" feature can provide this functionality.

To lock an activity with a password, use the following special device command available with the [Command button](#) type:

{screen lock:on:password}

Replace "password" in the above command with the password you would like to use. When locking is turned on using the above command, any time the user attempts to exit the activity (i.e. go back to the previous activity or activities home screen) using any provided mechanism (i.e. the "back" button in the navigation bar, or any [back] link button that you provide), they will be prompted to enter the configured password. If the correct password is supplied, the activity will exit normally. If an incorrect password is supplied, the user will be left on the activity. The password you provide is case sensitive. Note that the user can use links on the activity to navigate forward to other activities, so you can provide a set of activities that the user can access. It is up to you to determine what other activities are available via supplied link buttons. The user can then navigate back from those linked activities to the original locked activity, and when attempting to exit the locked activity, will encounter the password prompt. Once the activity has been successfully exited, the activity is gone, so the next time that activity is loaded, the lock must be re-enabled, likely using an auto-exec on load button. To lock an activity without a password, use the following device command using a Command button:

{screen lock:on}

When locking is turned on using the above command, all activity exit functionality will be disabled. Any attempt to exit the activity using any supplied mechanism will be ignored. In this case, the activity must be unlocked programmatically before the user is allowed to navigate away from the activity. This allows you to create your own mechanism for prompting the user for some criteria to unlock the screen. This can include presenting your own custom keypad to enter an unlock code, presenting the user with questions to answer, requiring some preset sequence of button presses, etc. Use your imagination. When your unlock requirement has been met, you can then unlock the activity programmatically.

To unlock a locked activity, use the following special device command using a Command button:

```
{screen lock:off}
```

This will disable activity locking for the currently active activity, once again enabling all normal activity exit mechanisms. After the activity is unlocked, you may either programmatically execute a [back] link command, or wait for the user to press a button to exit.

Note since the unlock command requires a Command button, to execute this via script, simply use:

```
return '[#]unlockButton';
```

(where "unlockButton" is the name of the Command button using the above unlock command). Two sample activities (one for iPhone and one for iPad) are available on the [download page](#), which demonstrate using activity locking to display a "lock screen", requiring the user to enter a numeric PIN to unlock and navigate back to the previous activity. In this example, the activity is locked without a password, as described above, and the unlock PIN is defined within the script included with the activity buttons. An external script file is also included, which contains some common script used by several buttons included in the layout. See the "OnLoad" button's script to find/change the configured PIN. This example also demonstrates a few other TouchControl features, such as pressed button images, group buttons, button alpha, animations, vibrations, Unicode characters on buttons, etc.

Interactive Web Views

With TouchControl Server 7.1 or later, you can configure a Web View button as "Interactive" (when creating the button or editing the button's definition), which allows the links, buttons, and/or script within the web page loaded in the web view to interact directly with the TouchControl activity that is hosting the web view. This allows you to design at least a portion of your activity with HTML/CSS, yet still retain use of TouchControl features not supported in HTML, such as custom TouchControl scripting, global and state variables (shared with other TouchControl activities), executing macros, link buttons, etc., along with executing all other primary TouchControl button types.

The easiest way to use this feature is to create TouchControl buttons that perform the tasks you wish to perform, drag them onto your layout and configure them with hot-spots (no images), disable them and size them small-ish and drag them out of the way. Then create a Web View button that will load the desired HTML page, which includes HTML links and buttons that directly execute the hidden "native" buttons on your layout. This could be especially useful if you have data collected on a server that you would like to display in TouchControl, using XSLT and/or CSS styling to generate a web page from the data, for example, and execute "native" TouchControl commands from the links and buttons on that web page.

What you need

1. A web page with some links, buttons, and/or script
2. A web server to server up the web page (optional)
3. TouchControl & TouchControl Server

The web page

The page containing the links/buttons/script can be as simple or complex as you like. The only requirement is that each element that triggers a TC button needs to execute one of two different "built-in" JavaScript functions which handle the communication with the "native" TouchControl activity. The first function is used to execute a native TC button found in the activity. This function has the following signature:

`_tcExecButton(buttonName);`

Here is an example of an HTML button and the associated script to execute a button:

```
<button onclick="_tcExecButton('MyButton');">My Button</button>
```

The built-in `_tcExecButton()` function "connects" the web page to the native TouchControl activity and its associated buttons. In fact, executing this function from a web page ultimately results in the same action as executing one TC button from another using the script `return ['#]MyButton'`; Any HTML element that can execute JavaScript (or a block of script itself, of

course) can execute the `_tcExecButton()` function.

One additional built-in function is included, which will allow you to update a native button's text, image, and/or icon. This function has the following signature:

```
_tcUpdateButton(buttonName, buttonText, buttonImage, buttonIcon);
```

Here is an example of an HTML link and the associated script to update a button:

```
<a  
href="javascript:_tcUpdateButton('MyButton','newText','myButtonPack/myImage.png','butt  
onicons/iconPack/iconImage.png');">My Button</a>
```

If you wish to only update the text and/or the image and/or the icon, but not all three, simply pass null for the attribute(s) you do not wish to update.

One last built-in function is necessary if you wish to interact with the native activity automatically when the web page first loads. The signature for this function is:

```
function _tcOnLoad() {}
```

To use this feature, create your own JavaScript function with this name within your web page and execute the desired commands from within this function. For example:

```
function _tcOnLoad() {  
    _tcExecButton('myOnLoadButton');  
}
```

This function will get executed automatically by TouchControl once the hosting activity has completely loaded. The reason this is required is because the web page may load before an auto-exec on load button completes in the native activity, and attempting to execute a native button from a web page prior to an auto-exec button completing can result in unexpected behavior. So, make sure that any calls to the above built-in functions are either included within the `_tcOnLoad()` function, or not attempted until after `_tcOnLoad()` has completed (i.e. triggered from links, buttons, etc.).

[The web server](#)

If you wish to host the web page on an external Web server, you may load the page in a Web View button by entering the page's URL on your web server in the Web View button configuration, as always.

However, this feature also includes the ability to send stand-alone HTML pages to your device (during config refresh) to store and load locally into Web Views, removing the need to host the pages on a remote web server, if desired. To do this, simply copy your stand-alone HTML page into the "HTML" folder located under your TouchControl Server data directory (where your TC config files are located - the path can be found on the Settings page in TouchControl Server). Any files found in this folder will be automatically sent to your iOS device the next time you tap the refresh button in TouchControl.

Then in the Web View button config, rather than entering an external URL to a web page, enter the following URL:

<http://localhost/html/myWebPage.htm>

Using the "localhost" hostname, along with the "/html/" path, will trigger TouchControl to look in the app's internal file system to locate the web page to load into the Web View. When using this feature, you need to make sure that any external resources referenced in your web page (images, script files, etc.) are absolute references vs. relative ones (unless, of course, you are referencing other images from your device).

[TouchControl & TouchControl Server](#)

Of course, you need to create an activity with a Web View button that will display the web page, as described above. Also on that activity, you'll need to add all of the buttons that will be triggered by the built-in script function. Just add the buttons as hot-spots (no image), disable them, and size them small-ish and drag them out of the way.

A complete, working sample activity is [available for download](#) that includes all of the above features. This sample activity and associated web pages also demonstrate controlling TouchControl from an external source using HTTP requests, [discussed here](#). Note that the sample activity is formatted for full-screen on the iPhone.

[USB-UIRT Broadcast \(Windows server only\)](#)

If you are a user of the USB-UIRT device, you can configure TouchControl Server (7.1 or later) to listen for signals received by the USB-UIRT, and re-broadcast the raw IR codes onto your network, where iOS devices running TouchControl can listen for and process those codes to perform client-side commands (update the activity interface, execute buttons, run script, etc.). To enable this feature, select Tools - Settings from the TC Server menu, then enable (check) the "USB-UIRT Broadcast" option at the bottom of the settings panel, then save settings. Any IR codes received by the USB-UIRT will be re-broadcast onto your network using the UDP protocol on port 8851. To receive these codes in TouchControl, create a Feedback Listener-type button and an associated Feedback Listener-type interface in Interface Manager in TouchControl Server settings. In the interface, leave the "Multicast Group" blank and set the port to 8851. Then in the Feedback Listener button, set this interface as the "Host", and add any script needed to process the various IR codes you expect to receive, drop it on your layout, and refresh to your device.

If you've been away from your network, or your iOS device has been asleep, or TouchControl has been in the background on your device, and you may have missed one or more automatic IR code broadcasts, you can trigger TouchControl Server to re-broadcast all unique IR codes that it has received since the last time the server was restarted, in the order they were last received by the USB-UIRT. To trigger the re-broadcast, create a [Command](#) button, and in the Command field for that button enter {rebroadcast ir}. When this button is executed from your device, the server will re-broadcast all IR codes in its current memory queue, and your Feedback Listener button (referenced above) will receive and process these broadcasts just as if they were being initially received by the USB-UIRT device.

Note that this only re-broadcasts unique IR codes. This means that if, for example, the USB-UIRT received codes: A, E, B, C, A, C, C, D, B, in that order, when re-broadcast, the server will send the codes: E, A, C, D, B, as those are the most recent instances of those codes, in the order they were last received.

When TouchControl Server is re-started, the IR code queue will start empty and will once again begin queuing IR codes for re-broadcast. Simply re-broadcasting the IR codes will not clear the queue, as you may have additional iOS devices that need to "catch up" with the broadcast IR codes at a later time. If you wish to clear the IR code queue without restarting the server, create a Command type button with the command {rebroadcast clear}. Sending this command will immediately empty the queue.

Note that the automatic broadcast feature is disabled while the USB-UIRT is in learn mode in TouchControl Server (learning a command from a remote control for an IR type button).

Server Configuration Management

If you are an installer, supporting and configuring TouchControl for multiple clients, or if you maintain multiple TouchControl configurations for your own use, such as for multiple TouchControl Servers in your household, or like to maintain "development" and "production" configs, you can easily switch between multiple configurations using the TouchControl Server "Data Directory" field on the server Settings panel.

The "Data Directory" field in server Settings designates the directory on your server where the active configuration is stored. Using the "Browse" button next to this field, you can easily switch to other directories containing complete configurations, or you may manually enter any directory you wish. Once you select or enter a new directory name, clicking the "Save" button on the Settings panel will automatically load the configuration found in that directory. If there is no configuration found, you will be asked if you'd like to copy the currently active config into that directory. This is an easy way to copy the current directory to start a new config for a new purpose. If you have a full config export .zip file, you may place it in an empty directory, then select/enter that directory name in the "Data Directory" field, and then when you click "Save", you will first be asked to copy the current config to that directly, and answering "No" will then prompt you to un-zip the full export into that directory, creating a new active configuration from that export .zip file. This is an easy way to share configurations between servers, or between users.

The "Data Directory" field is also a drop down list, which will automatically collect any directories that you have used in the past. This will allow you to easily switch back and forth between configurations that you have stored in separate directories on your server. If you manually delete a directory from your server's hard drive, it will automatically be removed from this list the next time you enter Settings. Or, if you'd just like to remove one of the entries from the list, but not remove the directory from your server, you may select it, then click the "X" button next to the drop down to remove it from the list.

(Windows only) You can also easily create a new config directory by typing it into the "Data Directory" field, then click the "+" button next to the field, and you will be asked if you'd like to copy the currently active config into that new directory, creating the directory if it doesn't exist.

Also, clicking the "Default" button next to the "Data Directory" field will simply populate the field with the default data directory name used during TouchControl Server install.

Server Configuration Recovery

The TouchControl iOS app includes a feature which allows you to recover your full configuration in the event that you lose the configuration on your server, for whatever reason. With this feature, you can easily email your full TouchControl configuration from your device to yourself (or anyone else). This feature is found at the bottom of the "Tools" section of the Settings screen in TouchControl on your device.

Web Remotes

Use your remote screens from a web browser on non-iOS devices (EXPERIMENTAL)
Consider this feature a "work in progress", but you can now render your remotes in any modern web browser that supports HTML 5. Only basic functionality is provided with this release - primary button types, macros, links, labels, repeating buttons, press & release buttons, timer buttons. Features not (yet) supported - sliders, spinners, gesture pads, web views, feedback clients & listeners, scripting, ...and anything else not mentioned here. Some of these features likely will never be implemented in a browser, but more features will be forthcoming.

To launch the web remotes, navigate a browser to:

<http://192.168.xx.xx:pppp/touchcontrol/webremote>

(where "192.168.xx.xx" is your TouchControl server's IP address, and "pppp" is the "Server Port" shown in TouchControl Server settings).

A couple of notes on using this feature. The WebRemote interface uses the "Location Overview" layout for the home activities screen (as in the iOS app - locations are initially listed, and selecting a location expands to display its contained activities). You can use the browser's back and forward buttons to move through the "stack" of activities as you navigate from screen to screen using your link buttons. If you wish to return directly to the main activities screen, you may long-press (or click & hold if using a mouse) on a blank spot on your activity background image, and the browser will navigate back to the main layout screen, removing any activities from the "stack".

Here's a screen shot of one user's WebRemote with Web Views displaying output from his security cameras.



[TouchControl Server JSON HTTP API \(Windows server only\)](#)

TouchControl Server supports incoming HTTP requests which return JSON-formatted responses, allowing you to retrieve locations, activities, devices, and/or buttons, as well as execute buttons remotely via HTTP requests to the server.

All JSON API requests start with the following host:port/path:

<http://serveripaddress:serverport/touchcontrol/json/>

...where *serveripaddress* is the IP address of your server, and *serverport* is the main server port configured on the Settings page of TouchControl Server.

The following unique requests are supported:

- getlocations
- getactivities
- getactivities?location=location
- getactivities?visible=trueORfalse
- getdevices
- getdevices?location=location&activity=activity
- getbutton?device=device&button=button
- getbuttons
- getbuttons?device=device
- getbuttons?location=location&activity=activity
- executebutton?device=device&button=button
- executebutton?device=device&button=button&feedback=trueORfalse
- executebutton?device=device&button=button&var=value&var=value&...

In all cases above, location, activity, device, and button must exactly match the specified element within your configuration, including upper/lower case, etc.

The `getactivities?visible=trueORfalse` parameter must match the string "true" to retrieve all visible activities, the string "false" to retrieve all invisible activities, or no value (or omit the parameter) to retrieve all activities by default. The "visible" parameter may be used with or without the "location" parameter.

The `executebutton?feedback=trueORfalse` parameter must match the string "true" to return feedback from the request, or the string "false" (or omit the parameter) to not return feedback. The `var=value` pairs available with the `executebutton` request define dynamic substitution variables that are embedded in the button commands (e.g. `%varname%`), which normally match global variables defined within TouchControl using `_global.varname` variables in JavaScript. This includes the dynamic IP address and dynamic Global Caché module and connector features as well. The only restriction is that the `var=` string may not be one of the strings "device=", "button=", or "feedback=". Dynamic substitution variables may be used with or without the "feedback" parameter.

The `executebutton` request may be used to execute buttons of the following types only:

- IR (ir)
- Command (cmd)
- AutoHotKey (ahk)
- HTTP Request (http)
- EventTrigger (et)
- Global Caché (gc)
- iRTrans (trans)
- IRCommand2 (irc2)
- URL (url)
- Macro (mac)

Attempting to execute a button type not in the above list will be ignored by TouchControl server. Therefore, the `getbuttons` request will only return buttons of the above types as well.

Examples of JSON API requests

```
http://192.168.1.100:8822/touchcontrol/json/getactivities?location=Home&visible=true
http://192.168.1.100:8822/touchcontrol/json/getbuttons?location=Theater&activity=Main
http://192.168.1.100:8822/touchcontrol/json/executebutton?device=DVR&button=On
http://192.168.1.100:8822/touchcontrol/json/executebutton?
    device=iTach&button=Back&module=1&connector=3
```

Response Format

The data returned from these requests will be in [JSON](#) format. JSON formatted data can be turned into JavaScript objects using either [the eval\(\) function](#), or [a built in JSON parser](#), if

available.

The following are the JSON response formats for the above requests:

getlocations:

```
[
  {
    "location": "locationName1",
    "activityUrl": "url_to_retrieve_activities_for_locationName1"
  },
  {
    "location": "locationName2",
    "activityUrl": "url_to_retrieve_activities_for_locationName2"
  },
  { ... }
]
```

getactivities:

```
[
  {
    "activity": "activityName1",
    "deviceUrl": "url_to_retrieve_devices_for_activityName1"
    "buttonUrl": "url_to_retrieve_buttons_for_activityName1"
  },
  {
    "activity": "activityName2",
    "deviceUrl": "url_to_retrieve_devices_for_activityName2"
    "buttonUrl": "url_to_retrieve_buttons_for_activityName2"
  },
  { ... }
]
```

getdevices:

```
[
  {
    "device": "deviceName1",
    "buttonUrl": "url_to_retrieve_buttons_for_deviceName1"
  },
  {
    "device": "deviceName2",
    "buttonUrl": "url_to_retrieve_buttons_for_deviceName2"
  }
]
```

```
},  
{ ... }  
]
```

getbutton:

```
{  
  "name": "buttonName",  
  "type": "buttonType",  
  "device": "buttonDevice",  
  "execUrl": "url_to_execute_buttonName"  
}
```

getbuttons:

```
[  
  {  
    "name": "buttonName1",  
    "type": "buttonType1",  
    "device": "buttonDevice1",  
    "execUrl": "url_to_execute_buttonName1"  
  },  
  {  
    "name": "buttonName2",  
    "type": "buttonType2",  
    "device": "buttonDevice2",  
    "execUrl": "url_to_execute_buttonName2"  
  },  
  { ... }  
]
```

executebutton (feedback only):

```
{  
  "feedback": "button_feedback_result"  
}
```

In all cases above, the URLs returned in the JSON response can be used as-is in a subsequent HTTP request, allowing you to "drill down" through your configuration to find the desired element(s), if desired.

Sample Code

[serverjsonapi.zip](#) contains the following files which provide examples of using the Server HTTP JSON API.

serverjsonapi.htm is a web page that demonstrates navigating through your config to provide a list of links which execute the buttons found. It also allows entering a device and button name to execute a given button. This file must be edited to update the network settings with your server's IP address and server port. Extract this file to your Windows desktop, update the network settings in the file, and double click on it to load it in your default browser.

<http://www.touch-ir.com/json/serverjsonapi.js>

serverjsonapi.js is a Windows JScript file that demonstrates executing a button given a device and button name. This file must be edited to update the network settings with your server's IP address and server port. Extract this file to your Windows desktop, update the network settings in the file, and double click on it to execute it.

Zero Config

ZeroConfig networking is now available which allows TouchControl on your device to automatically detect and configure the network settings (IP address and port number) of your TouchControl Server(s). During initial setup of the client app, if any TouchControl Servers are found on your network, TouchControl will update the configuration and automatically download the configuration from your server with no intervention required from the user. If multiple TouchControl Servers are found, the client configuration will be updated with the network settings of each server, and you may select which server to use for the initial configuration download.

To enable ZeroConfig networking, you must have Apple's "Bonjour for Windows" installed and active on your TouchControl Server. If it is not found, TouchControl Server will alert you and allow you to either quit TouchControl Server and install Bonjour, or disable ZeroConfig networking via server settings. Bonjour for Windows may be downloaded from the Apple web site here: <https://support.apple.com/kb/DL999>

TouchControl Server will now prompt you to supply a name for the server during the initial run of the server program. This should be a "friendly" name that you would like to use to distinguish the server, and will be used on the client to configure the server in the app settings. Once the client app is initially configured, any changes to your TouchControl Server topology (server name change, new server added, etc.) will be automatically detected by the client app and the settings will be updated accordingly (and you will be notified via an alert in the app on your device). Updates to the server settings are detected each time TouchControl is started, and each time you refresh your configuration by pressing the "Refresh" button in the upper-right corner of the TouchControl main activities screen.

Note: If your TouchControl Server has multiple network adapters (i.e. Ethernet and wireless), ZeroConfig will find both networks and configure them as multiple servers (with the same name) in the client configuration. This is not an error, and will allow you to contact your server when it is connected to your network in either wired or wireless modes.

If you prefer not to use ZeroConfig networking, once you disable the feature in TouchControl Server, you will not be prompted again (until you re-enable the feature), and all networking updates must be made manually in TouchControl on your device.

Screen Grabber

IMPORTANT: if you have trouble getting the grabber output to properly display, there is [workaround for this problem in the "Grabber not working?" topic on the Troubleshooting page](#) for more information.

NOTE: In older iOS releases there have been issues with viewing Motion JPEG (MJPEG) streams in Mobile Safari (including the Web View feature in TouchControl) which can prevent viewing TouchControl Grabber output on a device. If you have problems viewing grabber output, TouchControl Server (for Windows) has an option under the "Help – Troubleshooting" menu titled "Use Frame Grabber". When enabled, this will force the Grabber to serve up individual images (frames), rather than a MJPEG stream. To use the frame grabber, you'll also need to alter the configuration of your grabber Web View button to turn ON the "Autorefresh" option, and set the refresh rate to 0 (zero) seconds. This will force the Web View to continually request new frames from the server. Unfortunately, this will not provide the same frame rate experienced with the MJPEG grabber (due to latency introduced with the individual HTTP requests), but at least it will display the grabber output, where the previous grabber settings may not.

The Screen Grabber is a unique feature of TouchControl Server that allows you to "grab" or "capture" a region of your Windows desktop and view it within a Web View in TouchControl on your iOS device. The Screen Grabber consists of a "grab frame", which is a transparent window that you can drag, resize, and place anywhere on your desktop, and whatever is displayed within that region - whether it's any other window, or the desktop itself), is "grabbed" and sent to your iOS device and displayed as an image within a Web View using [Motion JPEG over HTTP](#). This allows you to watch, or monitor, activity on your computer's desktop from within a TouchControl remote activity. Uses of this can include capturing output from games or simulators and displaying in real time on your iPhone or iPad, monitoring program output/progress remotely, or viewing any activity that occurs on your computer. Using the screen grabber is incredibly easy. When in TouchControl Server simply select Tools - Screen Grabber from the menu, and the grab frame will open and begin "grabbing" the region of your desktop located within its bounds. Simply move the window to the desired location by dragging the grab frame title bar, and resize the frame by dragging the frame border - just like any other window on your desktop. When moving or resizing the frame, a panel will appear in the upper right corner of the grab frame which shows the current grab frame size and location. For more precise control, simply double-click on the frame's title bar to display an input panel within the grab frame which allows you to enter precise size and location parameters in pixel coordinates.

This input panel will also allow you to select the quality of the images that will be served to your

device. "Normal" quality provides the best blend of quality and performance. "High" quality will improve the quality/resolution of the images, while degrading performance to some degree.

You will also have the option to capture the mouse pointer within the grab window. When paired with a *gesture pad* mousepad (placing a transparent mousepad on top of the grabber Web View in your layout), this option will allow you to control your PC (with the mousepad) while viewing your desktop from your iOS device (via the grabber). While this option is enabled and the grabber is running, the mouse movements generated from the TouchControl mousepad will not allow the mouse pointer to exit the grabber frame on your computer's desktop. This ensures that you can always see the mouse pointer in the grabber output while controlling the cursor. (This does not affect mouse movement generated from the actual mouse connected to your PC, however.)

On this same input panel there is also an option to "Set & Hide" the grab frame. This option will set the size and location, and then hide the grab frame from view. To re-display the grab frame, select Tools - Show Screen Grabber from the TouchControl Server menu.

When done with the grabber, simply click the 'X' in title bar, just like any other window, or the grabber will also automatically shut down when the TouchControl Server application is shut down. Each time the grabber is started, it will automatically return to the same size and location from the last time the grabber was used.

Viewing Grabber Output

To view the output from the grabber, simply drop a Web View button on any activity within the TouchControl Server designer, and when configuring the Web View (using "Set Data"), simply enter the following:

```
%touchcontrolserver/grabber%
```

When the Web View is rendered within TouchControl on your iOS device, the above string will be dynamically converted to the required URL (including TouchControl Server IP address and port number) to retrieve the screen grabber output.

You can also enter raw HTML into a Web View to have that HTML rendered within TouchControl. This allows you to use HTML like the following to gain more control of the grabber output:

```
<body style="margin:0px"></body>
```

This is an extremely simple HTML example, but you can also include JavaScript and any other

style attributes/tags to interact with and manipulate the grab images in virtually any way desired.

The grab frame does not need to be on top of any other windows to display the contents of those windows. It simply provides an easy way to specify the region of the screen that will be captured by the screen grabber. If the grab frame is hidden by other windows, you can easily find it by selecting Tools - Screen Grabber again from the TouchControl Server menu, and the grab frame will pop to the top and flash on the screen. If the grab frame is somehow moved off screen to an inaccessible location, you can easily return it to its default location by selecting Help - Troubleshooting - Reset Grabber Coordinates. This will stop the grabber, and the next time it is started, it will be located in the upper-left of your primary display.

Grabber Protection (Windows server only)

You may "protect" your screen grabber output - that is, restrict it's viewing to only desired devices - by specifying the device IDs which are allowed to view Screen Grabber output. To specify the approved device IDs, double-click the grabber frame title bar to display the grabber config panel, select the "Protect" option, then click the "Enter Device IDs" button, and add any desired device IDs. Use the "+" button to add a new ID, use the "-" button to remove a selected ID, or double-click an existing device ID in the list to modify it. To find the ID for your device, access Settings within TouchControl, then tap the "i" icon in the upper right, displaying the app's "About" screen, and then tap "Device ID" in the lower left corner of the screen. Devices running operating systems prior to iOS 7 will display the device's MAC address, which should be entered as displayed, and devices running iOS 7 or later will display a unique "Vendor ID", which is a lengthy, multi-part hex string separated by dashes. This entire string must be used in the grabber protect feature, and can be entered either with or without the dashes.

Within the web view displaying the grabber output, simply use the %touchcontrol/grabber% substitution tag to generate the proper request including the device ID. See above for more information.

You may also use the "Help - Troubleshooting - Local Command Echo" feature of TouchControl Server to view incoming requests for grabber output. Any request made by TouchControl on an iOS device will display the device's ID in the local command echo window (denoted with a "[grab:id]" prefix). You may copy & paste the ID from there as an easy way to include your devices in the protection list.

Troubleshooting the Grabber

Some Windows applications use full-screen mode to display content (including games, simulators, etc.). In some cases, it will be necessary to disable Windows Aero in Windows 7 or Vista in order to grab the output from those applications when in full-screen mode. If Windows Aero is enabled, and you find that the grabber only captures the underlying windows desktop when an application is in full-screen mode, try using the Help - Troubleshooting - Disable Aero

While Grabbing option within TouchControl Server to temporarily disable Aero in order to capture the desired output. If Aero was enabled when using this option, it will be automatically re-enabled when you either stop the grabber, or shut down the TouchControl Server application. Please note that some Windows applications use certain DirectX hardware acceleration features that may keep you from capturing their output even when Aero is disabled. In those cases, you may need to also disable DirectX hardware acceleration in order to grab the output from those applications. Please consult your video card documentation or Microsoft DirectX documentation for more information on DirectX hardware acceleration enabling/disabling.

Please see the [Solutions page](#) for an overview of some uses of the TouchControl Screen Grabber!

[EventGhost \(Windows server only\)](#)

To control EventGhost, you must also install the TouchControl EventGhost plugin that is included with the TouchControl Server installation. The string you enter in the button configuration will then become the event trigger within the EventGhost system. The event prefix is configured within in the TouchControl EG plugin in the EventGhost interface, and defaults to "TouchControl", so an EventTrigger button configured with the string "MuteOn", for example, would show up in EventGhost as "TouchControl.MuteOn". Please consult the EventGhost documentation for instructions on setting up macros and other events that can respond to the TouchControl triggers.

EventGhost Feedback

The EventGhost plugin also includes an action that allows you to send feedback from your EventGhost macros directly back to TouchControl on your iPhone/iPad/iPod. When you create an EventTrigger button in TouchControl Server, you are given the option to check the "Feedback" box, which tells the device app that after it sends the command to EventGhost, it should wait to receive some text back from EventGhost before continuing. That text can then be used to replace the caption on any of your activity buttons or labels, and/or replace the images associated with any button or label on your layout. To configure and use EventGhost feedback:

- Create an EventTrigger button in TouchControl Server and select the Feedback option
- Install the EventGhost plugin using the files located in the "EventGhost Plugin" directory created in the install location (under Program Files) during the TouchControl Server installation.
- Restart EventGhost
- Create a macro in EventGhost that triggers on any TouchControl event
- Add the TouchControl "Feedback" action to your macro
- When prompted to configure the action, enter the name of the button or label that should receive the feedback string into the "Button/Label Name" field (an example is pre-populated into this field - change it to your button/label name), and enter the text that you'd like to send back to TouchControl into the "String Value" field.
Note that the string value can either be static text, or it can be a dynamically generated string. The default example that is populated in the "String Value" field is an example of a dynamically generated string: {eg.Utils.time.strftime('%Y-%m-%d @ %H:%M:%S')} This generates a string with the current date and time. Please consult the EventGhost documentation, documentation from any other plugins that you may use to generate feedback text, and/or a Python programming references for information on how to generate dynamic string values.

For more control over the data returned from EventGhost, use the "Custom Value" field on the TouchControl EventGhost plugin feedback action settings. Any value entered into the "Custom Value" field will override any existing values in the "Button/Label Name" and "String Value" fields.

- `buttonName^buttonText`

Where "buttonName" is the name of the label or button that will receive the feedback value, and "buttonText" is the string value that will be placed on the button or label. Either of those values may be static text, or may be dynamically generated from EventGhost script. Please consult the EventGhost documentation for information on generating dynamic string values.

- You may also update multiple buttons/labels by including multiple update strings as follows:

`buttonName1^buttonText1|buttonName2^buttonText2|etc...`

- If you wish to update the image associated with any button or label, enter a string in the Custom Value field in the following format:

`buttonName[@]buttonpack/buttonfile.png`

Where 'buttonpack/buttonfile.png' is the name of an image from a button pack that has been included with your layout.

To update multiple button/label images, use the same format as above:

`buttonName1[@]buttonpack/buttonfile1.png|buttonName2[@]buttonpack/buttonfile2.png|etc...`

Please note that when you designate a button as an EventTrigger feedback button, when the button is pressed TouchControl will wait up to 10 seconds to receive the feedback. If no feedback is received within that time, TouchControl will not populate your feedback label or button caption, and will simply continue normally.

If you wish to process the feedback from this button using custom script, select the "Feedback Script" option and enter/paste your JavaScript in the provided text box. See the [Scripting](#) topic for more information regarding feedback and feedback scripting.

If, when attempting to send feedback, the EventGhost plugin displays the message "Bad file descriptor" in the EventGhost log window, this means that TouchControl is not connected to EventGhost to receive the feedback. Please check to make sure your TouchControl button has been designated to receive feedback in the button definition, or if using a Feedback Client button, make sure the button exists on your activity layout, and the activity is open on your iOS device.

Note: If using EventGhost, your EventGhost server may be running on any PC/server on your network. EventGhost does NOT have to be running on the same system as TouchControl Server.

Wake-on-LAN

Broadcast a wakeup call to your network devices to bring them to life - right from a button on your TouchControl screen! Wake up any device that will respond to a "magic packet" message. This feature is implemented via the "EventTrigger" interface (see #5).

To send wake-on-LAN (WOL) to any device on your network:

- Use "Interface Manager" (in TouchControl Server Settings) to add a broadcast IP address for your network. When adding the broadcast device:

Select "EventTrigger" as the server type

Give the server a unique name, such as "WOL Broadcast"

Enter a broadcast IP address for your network:

This can either be 255.255.255.255, or can be specific to your network address scheme, such as 192.168.1.255 (if your router/gateway address is 192.168.1.1, for example)

This interface can now be used to send wake-on-LAN messages to any device on your network (just make sure the final octet of the broadcast IP address is "255").

- Save the new interface settings.
- Add an "EventTrigger" button to one of your devices.
- In the "EventTrigger Command" field, enter the following:

WOL[device MAC address]

The device MAC address should be all hex characters, with no dashes, colons, or other special characters.

Example: WOL[001C67617D2D]

- Please consult your device's documentation or network settings to find its MAC address.
- The "Command Terminator" is not used for this button.
- Set the "Host" selection to the broadcast server added previously (above)
- The "Magic Packet" will be broadcast out to your entire network, but only the device whose MAC address matches the hex string entered in the command will respond.

To wake up multiple devices at the same time, add a button for each and create a macro containing all of the buttons.

[Access From the Internet](#)

Within Settings in the iOS app on your device, you can supply an additional, internet-facing IP address for your TouchControl Server(s), and TouchControl will automatically detect the network state and use the best connection available (i.e. WiFi if it is available, and WWAN if it is not).

Within Settings, select your TouchControl Server (under "Network Settings"), then tap the "Edit" button on the nav bar, and enter your Internet-facing IP address in the "WAN Server" field. If you use a dynamic domain name service (such as DynDNS or Windows Home Server DNS service), you may enter your DNS hostname instead of an IP address in this field. If there is an IP address or host name entered in this field, TouchControl will attempt to use it whenever the device does not have a WiFi connection available. On the main settings screen, you can also turn on the "Force WAN" option, which will force TouchControl to use the "WAN Server" address or host name, even when on a WiFi connection. This is useful when you want to use a public WiFi hotspot such as at a coffee shop or hotel. Note that in this situation it is not truly forcing WAN connectivity, it is simply forcing TouchControl to use the "WAN Server" address to connect to the server. An additional option is also available to "Force LAN", which always uses the LAN server and interface host addresses, regardless of the type of network you are connected to. This is useful when you are connected to a cellular network, but are using a VPN to access your local network.

Also, see the [Interface Manager](#) section to find instructions for setting up WAN-specific ports for your interface hosts.

Custom Slider Images

One feature that this enables is the ability to provide custom image for slider buttons. This includes the left and right slider bars, as well as the "thumb" image (the image that you slide back and forth with your finger).

These custom images are specified for a slider button using the "button properties" feature while designing a layout. After a button has been dropped onto a layout, access custom properties by right-clicking on the slider button within the layout and selecting "Properties..." from the popup menu. All slider buttons will have three built-in properties named "sliderBarLeft", "sliderBarRight", and "sliderThumb". Select any of these properties from the "Name" list, and enter the location of the slider image in the following format:
myleftbarimage.png

Once you've entered the desired image file name in the property value field, click the + button to add the property to the button's properties list, then click Save to exit the properties dialog after adding all three slider image properties to the list.

Custom slider images are located in the "sliderimages.zip" button pack within the images folder under your TouchControl data directory on your server. A few sample slider images are included with the server installation. The filenames of these sample images are:

sampleleftbar.png
samplerrightbar.png
samplethumb.png
sampleleftbar2.png
samplerrightbar2.png
samplethumb2.png

Give these images a try to see what is possible.

To add your own images, simply include them in this .zip file, just as you would any other button image that you supply. The slider images must be found in this file, and be located within the "sliderimages" folder within the sliderimages.zip file to be rendered within TouchControl activities on your device.

Also note that the slider images will not display on the layout designer panel on the server, but will display when the slider is rendered within the activity in TouchControl on your device.

Global Caché "Smooth Continuous IR Commands"

"Smooth continuous IR commands" is a term that Global Caché (GC) uses to describe a method of sending IR commands via their adapters that results in continuous repeating IR signals sent to your devices with no interruptions, or "choppy actions". To fully understand this concept, you should first read the documentation provided by Global Caché on their web site at this address: <http://www.globalcache.com/files/docs/API-iTach.pdf> (see page 13 or this document). After you have read that, please return here to continue. As outlined in the GC document, "smooth continuous IR commands" are very useful for things like volume control, so that's where this document will focus as well.

To perform smooth continuous IR commands within TouchControl, we need a way to send repeating commands to the GC adapter, and then abruptly stop those commands when we want the repeating to stop. Luckily TouchControl includes everything required to perform this action:

- Ability to send repeating commands - the repeating button type, of course
- Ability to send individual IR commands with an embedded repeat count greater than 1 - Global Caché buttons allow this in the GC command settings when learning or editing GC IR commands
- Ability to stop an IR command from repeating at any moment - using Global Caché "stopir" command
- Ability to start repeating when you press a button, and stop repeating when you release a button - the TouchControl "Press & Release" macro type

In short, when we press a button (Volume Up, for example), we want TouchControl to begin sending volume up IR commands, and when we release the button, we want it to stop. Simple, right? Well, when we press the button, we don't want to just repeat normal, single IR commands. For truly continuous, un-interrupted IR control, we'd like each repeating command that we send to have its own repeat count that is internal to the GC adapter. So, for each repeated command from TouchControl at, say, 1 second intervals, the GC adapter will actually repeat the command to your device 10 times (again as an example). This basically takes the network and any associated lag time out of the equation for those 10 repeated signals, since they are all generated within the GC adapter and sent straight to your device with no communication back to TouchControl needed. Then as you continue to hold down the button in TouchControl, TouchControl will continue to send IR commands which will in turn be repeated on their own within the GC adapter.

A nice feature of the GC adapters that really makes this possible (and is discussed in their documentation linked above) is that while the adapter is sending a repeating command, if it

receives another identical repeating command, it will simply start the repeat count over at zero, rather than adding the new repeating count to the old, currently repeating command. Therefore, if you set the GC IR repeat count to 10, then you will be guaranteed to not get more than 10 repeated commands after you send any individual signal from TouchControl.

So now what about terminating the repeating commands. If we've just sent a command to GC that will repeat 10 times on its own, we still want to be able to terminate that repeating command when we release the button on the TouchControl screen. That's where the "stopir" command comes into play. If the GC adapter is currently repeating an individual IR command, and it receives a "stopir" command, it will immediately stop repeating and discard any further commands in its current repeat cycle.

So now we have the basic elements needed - the ability to continuously repeat IR commands to your device without network lag, and the ability to stop those commands on demand. To put those together, we'll use a "Press & Release" TouchControl macro. The following is a step-by-step guide to creating a smooth continuous press & release button.

1. Within TouchControl Server, select any device that you'd like to add the button to and click "Add Button".
2. Create a new Global Caché repeating button (in any device desired) with a repeat interval of .25 sec. We'll name this button "VOL++".
3. Click "Add" to add this button to your list.
4. Click "Set Data" for this button and select the "IR" type in the upper left, then either click "Learn", "Edit", or "Import" depending on how you will be acquiring your IR code. If you already have an existing GC IR button with an existing code that you are using instead, click "Edit" to modify the existing button settings.
5. Select the iTach or GC-100 device you will be using in the "Device Name" drop-down at the bottom.
6. On the left side of the dialog box, set the "Module" and "Connector" values based on the module/connector you will be targeting on the GC adapter.
7. Set the "Repeat" count for this command to 20.
8. Set the "Separation" value for this command to 20 milliseconds. The separation value will indicate the length of time to wait between repeats by the GC adapter. Given all the math, this should be sufficient to allow TouchControl to send another command before the current repeating command has completed, ensuring there is no break in the continuous IR signal to your device.
9. Save this button.

10. Create a new Global Caché non-repeating button. We'll name this button "StopIR".
11. Click "Set Data" for this button and select the "IR" type in the upper left, then click the "Edit" button to modify the button config.
12. Select the iTach or GC-100 device you will be using in the "Device Name" drop-down at the bottom (the same device used for the "VOL++" button above).
13. On the left side of the dialog box, set the "Module" and "Connector" values to match the module/connector used for the "VOL++" button above.
14. Set the "Repeat" value to 1.
15. The "Separation" value doesn't matter in this case since the command will not be repeating, so just leave it at its default. The "Frequency" also is not used since this is not a "true" IR hex command.
16. In the empty text box to the right, simply enter "stopir" (all lower case, without the quotes).
17. Save this button.
18. Create a new "Macro" button. We'll name this one "Volume Up".
19. Click "Set Data" to modify the button's config.
20. Select the "Press & Release" option at the top.
21. Locate the "VOL++" button created above and add it as the "Press" action.
22. Locate the "StopIR" button created above and add it as the "Release" action.
23. Save this macro.

Add the new "Volume Up" macro button to a layout and refresh to your device.