# SoCo (Sonos Controller) Documentation

*Release 0.1*

Rahim Sonawalla, et al.

June 05, 2014

SoCo (Sonos Controller) is a Python library to control your Sonos speakers.

CHAPTER 1

# Contents

## 1.1 Tutorial

*SoCo* allows you to control your Sonos sound system from a Python program. For a quick start have a look at the example applications that come with the library.

### 1.1.1 Discovery

For discovering the Sonos devices in your network, use the `SonosDiscovery` class.

```
sd = SonosDiscovery()
ips = sd.get_speaker_ips()
```

### 1.1.2 Music

Once one of the available devices is selected, the `SoCo` class can be used to control it. Have a look at the *The soco module* for all available commands.

```
sonos = SoCo(ip)
sonos.partymode()
```

## 1.2 The `soco` module

SoCo (Sonos Controller) is a simple library to control Sonos speakers

**class** `soco.`**`SonosDiscovery`**

A simple class for discovering Sonos speakers.

Public functions: get_speaker_ips – Get a list of IPs of all zoneplayers.

**class** `soco.`**`SoCo`**`(`*speaker_ip*`)`

A simple class for controlling a Sonos speaker.

Public functions: play – Plays the current item. play_uri – Plays a track or a music stream by URI. play_from_queue – Plays an item in the queue. pause – Pause the currently playing track. stop – Stop the currently playing track. seek – Move the currently playing track a given elapsed time. next – Go to the next track. previous – Go back to the previous track. mute – Get or Set Mute (or unmute) the speaker. volume – Get or set the volume of the speaker. bass – Get or set the speaker's bass EQ. set_player_name – set the name of the Sonos

Speaker treble – Set the speaker's treble EQ. set_play_mode – Change repeat and shuffle settings on the queue. set_loudness – Turn on (or off) the speaker's loudness compensation. switch_to_line_in – Switch the speaker's input to line-in. status_light – Turn on (or off) the Sonos status light. get_current_track_info – Get information about the currently playing track. get_speaker_info – Get information about the Sonos speaker. partymode – Put all the speakers in the network in the same group. join – Join this speaker to another "master" speaker. unjoin – Remove this speaker from a group. get_queue – Get information about the queue. get_current_transport_info – get speakers playing state add_to_queue – Add a track to the end of the queue remove_from_queue – Remove a track from the queue clear_queue – Remove all tracks from queue get_favorite_radio_shows – Get favorite radio shows from Sonos' Radio app. get_favorite_radio_stations – Get favorite radio stations.

**add_to_queue**(*uri*)

Adds a given track to the queue.

Returns: If the Sonos speaker successfully added the track, returns the queue position of the track added.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**bass**(*bass=None*)

Get or set the Sonos speaker's bass EQ.

Arguments: bass – A value between -10 and 10.

Returns: If the bass argument was specified: returns true if the Sonos speaker successfully set the bass EQ.

If the bass argument was not specified: returns the current base value.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**clear_queue**()

Removes all tracks from the queue.

Returns: True if the Sonos speaker cleared the queue.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**get_current_track_info**()

Get information about the currently playing track.

Returns: A dictionary containing the following information about the currently playing track: playlist_position, duration, title, artist, album, position and a link to the album art.

If we're unable to return data for a field, we'll return an empty string. This can happen for all kinds of reasons so be sure to check values. For example, a track may not have complete metadata and be missing an album name. In this case track['album'] will be an empty string.

**get_current_transport_info**()

Get the current playback state

Returns: A dictionary containing the following information about the speakers playing state current_transport_state (PLAYING, PAUSED_PLAYBACK, STOPPED), current_trasnport_status (OK, ?), current_speed(1,?)

This allows us to know if speaker is playing or not. Don't know other states of CurrentTransportStatus and CurrentSpeed.

**get_favorite_radio_shows**(*start=0*, *max_items=100*)

Get favorite radio shows from Sonos' Radio app.

Returns: A list containing the total number of favorites, the number of favorites returned, and the actual list of favorite radio shows, represented as a dictionary with *title* and *uri* keys.

Depending on what you're building, you'll want to check to see if the total number of favorites is greater than the amount you requested (*max_items*), if it is, use *start* to page through and get the entire list of favorites.

**`get_favorite_radio_stations`**(*start=0*, *max_items=100*)

Get favorite radio stations from Sonos' Radio app.

Returns: A list containing the total number of favorites, the number of favorites returned, and the actual list of favorite radio stations, represented as a dictionary with *title* and *uri* keys.

Depending on what you're building, you'll want to check to see if the total number of favorites is greater than the amount you requested (*max_items*), if it is, use *start* to page through and get the entire list of favorites.

**`get_queue`**(*start=0*, *max_items=100*)

Get information about the queue.

Returns: A list containing a dictionary for each track in the queue. The track dictionary contains the following information about the track: title, artist, album, album_art, uri

If we're unable to return data for a field, we'll return an empty list. This can happen for all kinds of reasons so be sure to check values.

This method is heavly based on Sam Soffes (aka soffes) ruby implementation

**`get_speaker_info`**(*refresh=False*)

Get information about the Sonos speaker.

Arguments: refresh – Refresh the speaker info cache.

Returns: Information about the Sonos speaker, such as the UID, MAC Address, and Zone Name.

**`get_speakers_ip`**(*refresh=False*)

Get the IP addresses of all the Sonos speakers in the network.

Code contributed by Thomas Bartvig (thomas.bartvig@gmail.com)

Arguments: refresh – Refresh the speakers IP cache.

Returns: IP addresses of the Sonos speakers.

**`join`**(*master_uid*)

Join this speaker to another "master" speaker.

Code contributed by Thomas Bartvig (thomas.bartvig@gmail.com)

Returns: True if this speaker has joined the master speaker

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`mute`**(*mute=None*)

Mute or unmute the Sonos speaker.

Arguments: mute – True to mute. False to unmute.

Returns: True if the Sonos speaker was successfully muted or unmuted.

If the mute argument was not specified: returns the current mute status 0 for unmuted, 1 for muted

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`next`**()

Go to the next track.

Returns: True if the Sonos speaker successfully skipped to the next track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned. Keep in mind that next() can return errors for a variety of reasons. For example, if the Sonos is streaming Pandora and you call next() several times in quick succession an error code will likely be returned (since Pandora has limits on how many songs can be skipped).

**`partymode`**`()`

Put all the speakers in the network in the same group, a.k.a Party Mode.

This blog shows the initial research responsible for this:

http://travelmarx.blogspot.dk/2010/06/exploring-sonos-via-upnp.html

The trick seems to be (only tested on a two-speaker setup) to tell each

speaker which to join. There's probably a bit more to it if multiple groups have been defined.

Code contributed by Thomas Bartvig (thomas.bartvig@gmail.com)

Returns: True if partymode is set

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`pause`**`()`

Pause the currently playing track.

Returns: True if the Sonos speaker successfully paused the track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`play`**`()`

Play the currently selected track.

Returns: True if the Sonos speaker successfully started playing the track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`play_from_queue`**(*queue_index*)

Play an item from the queue. The track number is required as an argument, where the first track is 0.

Returns: True if the Sonos speaker successfully started playing the track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`play_uri`**(*uri='', meta=''*)

Play a given stream. Pauses the queue.

Arguments: uri – URI of a stream to be played. meta — The track metadata to show in the player, DIDL format.

Returns: True if the Sonos speaker successfully started playing the track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`previous`**`()`

Go back to the previously played track.

Returns: True if the Sonos speaker successfully went to the previous track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned. Keep in mind that previous() can return errors for a variety of reasons. For example, previous() will return an error code (error code 701) if the Sonos is streaming Pandora since you can't go back on tracks.

**`remove_from_queue`**(*index*)
Removes a track from the queue.

index: the index of the track to remove; first item in the queue is 1

Returns: True if the Sonos speaker successfully removed the track

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`seek`**(*timestamp*)
Seeks to a given timestamp in the current track, specified in the format of HH:MM:SS.

Returns: True if the Sonos speaker successfully seeked to the timecode.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`set_loudness`**(*loudness*)
Set the Sonos speaker's loudness compensation.

Loudness is a complicated topic. You can find a nice summary about this feature here: http://forums.sonos.com/showthread.php?p=4698#post4698

Arguments: loudness – True to turn on loudness compensation. False to disable it.

Returns: True if the Sonos speaker successfully set the loundess compensation.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`set_play_mode`**(*playmode*)
Sets the play mode for the queue. Case-insensitive options are: NORMAL – Turns off shuffle and repeat. REPEAT_ALL – Turns on repeat and turns off shuffle. SHUFFLE – Turns on shuffle *and* repeat. (It's strange, I know.) SHUFFLE_NOREPEAT – Turns on shuffle and turns off repeat.

Returns: True if the play mode was successfully set.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`set_player_name`**(*playername=False*)
Sets the name of the player

Returns: True if the player name was successfully set.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`status_light`**(*led_on*)
Turn on (or off) the white Sonos status light.

Turns on or off the little white light on the Sonos speaker. (It's between the mute button and the volume up button on the speaker.)

Arguments: led_on – True to turn on the light. False to turn off the light.

Returns: True if the Sonos speaker successfully turned on (or off) the light.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`stop`**()

Stop the currently playing track.

Returns: True if the Sonos speaker successfully stopped the playing track.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`switch_to_line_in`**()

Switch the speaker's input to line-in.

Returns: True if the Sonos speaker successfully switched to line-in.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned. Note, an error will be returned if you try to switch to line-in on a device (like the Play:3) that doesn't have line-in capability.

**`treble`**(*treble=None*)

Get or set the Sonos speaker's treble EQ.

Arguments: treble – A value between -10 and 10.

Returns: If the treble argument was specified: returns true if the Sonos speaker successfully set the treble EQ.

If the treble argument was not specified: returns the current treble value.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

**`unjoin`**()

Remove this speaker from a group.

Seems to work ok even if you remove what was previously the group master from it's own group. If the speaker was not in a group also returns ok.

Returns: True if this speaker has left the group.

**`volume`**(*volume=None*)

Get or set the Sonos speaker volume.

Arguments: volume – A value between 0 and 100.

Returns: If the volume argument was specified: returns true if the Sonos speaker successfully set the volume.

If the volume argument was not specified: returns the current volume of the Sonos speaker.

If an error occurs, we'll attempt to parse the error and return a UPnP error code. If that fails, the raw response sent back from the Sonos speaker will be returned.

CHAPTER 2

# Indices and tables

- *genindex*
- *modindex*
- *search*

# Python Module Index