FTC-Music-Player Documentation

Contents

Namespace `FTC-Music-Player`

Sub-modules

- FTC-Music-Player.api_client
- FTC-Music-Player.config
- FTC-Music-Player.data_models
- FTC-Music-Player.lib
- FTC-Music-Player.localfiles
- FTC-Music-Player.main
- FTC-Music-Player.models
- FTC-Music-Player.player_handlers
- FTC-Music-Player.search_models
- FTC-Music-Player.ui
- FTC-Music-Player.ui_builder

Namespace `FTC-Music-Player.api_client`

Sub-modules

- FTC-Music-Player.api_client.Youtube
- FTC-Music-Player.api_client.api_client

Namespace `FTC-Music-Player.api_client.Youtube`

Sub-modules

- FTC-Music-Player.api_client.Youtube.api_models
- FTC-Music-Player.api_client.Youtube.youtube

Module `FTC-Music-Player.api_client.Youtube.api_models`

Classes

Class `GetAlbumSongsResponse`

```
class GetAlbumSongsResponse(
    has_error: bool,
    error: str,
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

Class variables

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Variable `songs` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]`

Class `GetArtistAlbumsResponse`

```
class GetArtistAlbumsResponse(
    has_error: bool,
    error: str,
    albums: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum]
)
```

Class variables

Variable `albums` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum]`

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Class `GetArtistSongsResponse`

```
class GetArtistSongsResponse(
    has_error: bool,
    error: str,
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

Class variables

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Variable `songs` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]`

Class `GetSongUrlResponse`

```
class GetSongUrlResponse(
    has_error: bool,
    error: str,
    url: str
)
```

Class variables

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Variable `url` Type: `str`

Class `GetSuggestionsRequest`

```
class GetSuggestionsRequest(
    artist_count,
    album_count,
    song_count
)
```

Class variables

Variable `album_count` Type: `int`

Variable `artist_count` Type: `int`

Variable `song_count` Type: `int`

Class `GetSuggestionsResponse`

```
class GetSuggestionsResponse(
    has_error: bool,
    error: str,
    artists: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineArtist],
    albums: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum],
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

Class variables

Variable `albums` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum]`

Variable `artists` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineArtist]`

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Variable `songs` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]`

Class `OnlineAlbum`

```
class OnlineAlbum(
    id: str,
    artist_id: str,
    name: str,
    cover_art: str,
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

Immutable collection of Songs made by the same Artist and grouped togther.

- name – Name of the Album.
- arist – Name of the Artist that made the Album.

- songs – List of Songs in the Album.

- _path – URL of the playlist on YT.

Ancestors (in MRO)

- data_models.Album
- data_models.Content

Class variables

Variable `artist_id` Type: `str`

Variable `cover_art` Type: `str`

Variable `id` Type: `str`

Class `OnlineArtist`

```
class OnlineArtist(
    id: str,
    name: str,
    cover_art: str,
    albums: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum],
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

An Artist class to model the data for an artist (in our case, a YouTube channel).

- name – Name of the artist.

- albums – Albums (playlists) made by the artist.
- songs – Songs (uploads).
- id – Unique identifier.

Ancestors (in MRO)

- data_models.Artist

Class variables

Variable `cover_art` Type: `str`

Variable `id` Type: `str`

Class `OnlineSong`

```
class OnlineSong(
    id: str,
    artist_id: str,
    name: str,
    url: str,
    cover_art: str,
    duration: datetime.timedelta
)
```

Models a single Song that can be a part of zero or more Albums or Playlists, as well as followed by zero or more Users.

- name – Song's name.
- artist – Creator of the song (channel that uploaded it).
- _path – Either a local path to the song file or a URL generated from the C# back-end.

Ancestors (in MRO)

- data_models.Song
- data_models.Content

Class variables

Variable `artist_id` Type: `str`

Variable `id` Type: `str`

Class `SearchRequest`

```
class SearchRequest(
    query,
    artist_count,
    album_count,
    song_count
)
```

Class variables

Variable `album_count` Type: `int`

Variable `artist_count` Type: `int`

Variable `query` Type: `str`

Variable `song_count` Type: `int`

Class `SearchResponse`

```
class SearchResponse(
    has_error: bool,
    error: str,
    artists: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineArtist],
    albums: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum],
    songs: list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]
)
```

Class variables

Variable `albums` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineAlbum]`

Variable `artists` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineArtist]`

Variable `error` Type: `str`

Variable `has_error` Type: `bool`

Variable `songs` Type: `list[FTC-Music-Player.api_client.Youtube.api_models.OnlineSong]`

Module `FTC-Music-Player.api_client.Youtube.youtube`

This file builds the requests to the API server via the api_client and parses the responses into the models defined in models.py.

Functions

Function `getAlbumSongs`

```
def getAlbumSongs(
    album_id: str
) -> api_client.Youtube.api_models.GetAlbumSongsResponse
```

Invokes a GetAlbumSongs Request via api_client.

Takes in a album_id as a string.

Returns a GetAlbumSongsResponse containing the songs of the album.

Function `getArtistAlbums`

```
def getArtistAlbums(
    artist_id: str
) -> api_client.Youtube.api_models.GetArtistAlbumsResponse
```

Invokes a GetArtistAlbums Request via api_client.

Takes in a artist_id as a string.

Returns a GetArtistAlbumsResponse containing the albums of the artist.

Function `getArtistSongs`

```
def getArtistSongs(
    artist_id: str
) -> api_client.Youtube.api_models.GetArtistSongsResponse
```

Invokes a GetArtistSongs Request via api_client.

Takes in a artist_id as a string.

Returns a GetArtistSongsResponse containing the songs of the artist.

Function `getSongUrl`

```
def getSongUrl(
    song_id: str
) -> api_client.Youtube.api_models.GetSongUrlResponse
```

Invokes a GetSongUrl Request via api_client.

Takes in a song_id as a string.

Returns a GetSongUrlRespose containing the url of the song.

Function `getSuggestions`

```
def getSuggestions(
    request: api_client.Youtube.api_models.GetSuggestionsRequest
) -> api_client.Youtube.api_models.GetSuggestionsResponse
```

Invokes a GetSuggestions Request via api_client.

Takes in a GetSuggestionsRequest object.

Returns a GetSuggestionsResponse containing the suggestions.

Function `search`

```
def search(
    request: api_client.Youtube.api_models.SearchRequest
) -> api_client.Youtube.api_models.SearchResponse
```

Invokes a Search Request via api_client.

Takes in a SearchRequest object.

Returns a SearchResponse object containing the results of the search request.

Module `FTC-Music-Player.api_client.api_client`

This file directly interatcs with the API server via “sendApiRequest” function which takes in the controller, request, and params as strings, sends a request to the server and returns the response (in json format).

It also contains the ApiControllers and ApiRequests enums which are used to build the request url as well as the serverIp variable which is used to build the request url.

Functions

Function `sendApiRequest`

```
def sendApiRequest(
    controller: str,
    request: str,
    params: str
) -> str
```

Classes

Class `ApiControllers`

```
class ApiControllers(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

An enumeration.

Ancestors (in MRO)

- enum.Enum

Class variables

Variable `Youtube`

Class `ApiRequests`

```
class ApiRequests(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

An enumeration.

Ancestors (in MRO)

- enum.Enum

Class variables

Variable `AlbumSongs`

Variable `ArtistAlbums`

Variable `ArtistSongs`

Variable `AudioUrl`

Variable `Search`

Variable `Suggestions`

Module `FTC-Music-Player.config`

Various variables needed by several other files.

Module `FTC-Music-Player.data_models`

Defines several classes modelling data that will be received from the C# backend to be used by the Python frontend:

- User – A class to represent a single User of the application.
- Content – Top level class modelling any kind of content an Artist could make.

- Album : Content – A collection of Songs.
- Playlist : Album – A /mutable/ collection of Songs created by the User.
- Song : Content – A single Song.
- Artist – An entity that has zero or more Content.

Classes

Class `Album`

```
class Album(
    name: str,
    artist,
    songs: list[FTC-Music-Player.data_models.Song],
    path: str
)
```

Immutable collection of Songs made by the same Artist and grouped togther.

- name – Name of the Album.
- arist – Name of the Artist that made the Album.
- songs – List of Songs in the Album.
- _path – URL of the playlist on YT.

Ancestors (in MRO)

- FTC-Music-Player.data_models.Content

Descendants

- FTC-Music-Player.data_models.Playlist

Class variables

Variable `songs` Type: `list[FTC-Music-Player.data_models.Song]`

Class `Artist`

```
class Artist(
    name: str,
    id: str = 'FTC',
    albums: list[FTC-Music-Player.data_models.Album] = [],
    songs: list[FTC-Music-Player.data_models.Song] = []
)
```

An Artist class to model the data for an artist (in our case, a YouTube channel).

- name – Name of the artist.

- albums – Albums (playlists) made by the artist.
- songs – Songs (uploads).
- id – Unique identifier.

Class variables

Variable `albums` Type: `list[FTC-Music-Player.data_models.Album]`

Variable `id` Type: `str`

Variable `name` Type: `str`

Variable `songs` Type: `list[FTC-Music-Player.data_models.Song]`

Class `Content`

```
class Content
```

Class modelling any form of Content made my an Artist, inherited by several other classes.

- artist – Channel that uploaded this content.
- _path – Either a local path to the file or a URL generated from the C# back-end.

Descendants

- FTC-Music-Player.data_models.Album
- FTC-Music-Player.data_models.Song

Class variables

Variable `artist`

Variable `name` Type: `str`

Class `Playlist`

```
class Playlist(
    name: str,
    path: str,
    songs: list[FTC-Music-Player.data_models.Song] = []
)
```

Playlist class to be used in the interface between the back- and front-ends.

- name – Name of the Playlist.
- arist – Name of the Artist that made the Playlist.
- songs – List of Songs in the Playlist.
- _path – Local path to the playlist folder (of the form “./playlist/”).
- add_song() – Adds a Song to the list of Songs.

Ancestors (in MRO)

- FTC-Music-Player.data_models.Album
- FTC-Music-Player.data_models.Content

Methods

Method `add_song`

```
def add_song(
    self,
    song: FTC-Music-Player.data_models.Song
)
```

Class `Song`

```
class Song(
    name: str,
    artist,
    path: str,
    duration: datetime.timedelta,
    cover_art: str = ''
)
```

Models a single Song that can be a part of zero or more Albums or Playlists, as well as followed by zero or more Users.

- name – Song’s name.
- artist – Creator of the song (channel that uploaded it).
- _path – Either a local path to the song file or a URL generated from the C# back-end.

Ancestors (in MRO)

- FTC-Music-Player.data_models.Content

Class variables

Variable `duration` Type: `datetime.timedelta`

Class `User`

```
class User(
    username: str,
    token: str
)
```

A User class to model the data for a user of the application.

- username – Personal identifier
- token – Unique identifier made by concatenating the user’s username with their password and computing their SHA256.
- playlists – List of playlists/albums the user has saved.
- favourites – Special playlist of the user’s favourite songs.
- followed_artists – List of all the artists that the user has followed.
- followed_albums – List of all the albums that the user has followed.

Class variables

Variable `favourites` Type: `FTC-Music-Player.data_models.Playlist`

Variable `followed_albums` Type: `list[FTC-Music-Player.data_models.Album]`

Variable `followed_artists` Type: `list[FTC-Music-Player.data_models.Artist]`

Variable `playlists` Type: `list[FTC-Music-Player.data_models.Playlist]`

Variable `token` Type: `str`

Variable `username` Type: `str`

Methods

Method `change_username`

```
def change_username(
    self,
    new_username: str
)
```

Method `create_playlist`

```
def create_playlist(
    self,
    name: str
)
```

Method `follow_album`

```
def follow_album(
    self,
    album: FTC-Music-Player.data_models.Album
)
```

Method `follow_artist`

```
def follow_artist(
    self,
    artist: FTC-Music-Player.data_models.Artist
)
```

Method `like_song`

```
def like_song(
    self,
    song: FTC-Music-Player.data_models.Song
)
```

Module `FTC-Music-Player.lib`

Definitions of some generally useful functions to use throughout the project.

Functions

Function `TODO`

```
def TODO(
    s: str
)
```

Mark a function or method as unimplemented.

- s – name.

Function `calc_pos`

```
def calc_pos(
    duration: datetime.timedelta,
    time: datetime.timedelta
)
```

Function `err`

```
def err(
    name: str,
    e: str
)
```

General function for logging errors.

- name: Name of the calling module/function.

- s : Displayed log.

Function `logger`

```
def logger(
    name: str,
    s: str
)
```

General logging function for use by the library.

- name: Name of the calling module/function.

- s : Displayed log.

Function `passive`

```
def passive()
```

Sometimes you need to pass a function as a parametre to another, or sometimes you need an expression for the interpreter to stop complaining. This allows you to do so while also not doing anything.

Module `FTC-Music-Player.localfiles`

Defines a Local class that ...

Functions

Function `main`

```
def main(
    page: flet_core.page.Page
)
```

Classes

Class `Local`

```
class Local
```

This class does things.

- flist – …
- pathlist – …

Methods

Method `addtoqu`

```
def addtoqu(
    self
)
```

Method `getfolder`

```
def getfolder(
    self,
    listbox: flet_core.list_view.ListView,
    selected: str
) -> str | None
```

Get a folder selected by the user from a file dialog.

- listbox – Where the results will be displayed.
- selected – Selected folder to be displayed.

Method `getselected`

```
def getselected(
    self,
    _event,
    listbox
)
```

Method `pick_files_result`

```
def pick_files_result(
    self,
    e: flet_core.file_picker.FilePickerResultEvent
)
```

Method `quclear`

```
def quclear(
    self
)
```

Module `FTC-Music-Player.main`

The main file that will initialise the GUI and start the program process. It first initialises a "player", which communicates with an external library to play the actual audio. It then initialises the GUI (graphical user interface) and starts the program.

Module `FTC-Music-Player.models`

Some general classes that are neither data or to do with communicating with the API.

Classes

Class `Player`

```
class Player(
    handlers,
    queue
)
```

Abstract class to represent a music player, i.e., an API to communicate with an external audio playing application from within the program.

- queue – A Queue object representing the queue of songs the player will play when prompted.
- handlers – A dict of handler functions to interact with the UI.
- player – The actual player object that will allow us to communicate with whatever external library we use to play audio.

Descendants

- FTC-Music-Player.models.VlcMediaPlayer

Class variables

Variable `handlers` Type: `dict[player_handlers.HandlerType, typing.Callable[[], NoneType]]`

Variable `player` Type: `Optional[Any]`

Variable `queue` Type: `FTC-Music-Player.models.Queue`

Methods

Method `add_to_queu`

```
def add_to_queu(
    self,
    song: data_models.Song
)
```

Add a song to the current queue.

Method `change_queue`

```
def change_queue(
    self,
    queue: FTC-Music-Player.models.Queue
)
```

Swap the current queue of songs to another one.

Method `next`

```
def next(
    self
)
```

Skip the rest of the current song and move to the next one in the queue.

Method `pause`

```
def pause(
    self
)
```

Pause playing the current song.

Method `play`

```
def play(
    self
)
```

Start playing the current song at the current elapsed time.

Method `prev`

```
def prev(
    self
)
```

Stop playing the current song and return to the previous one in the queue.

Method `seekpos`

```
def seekpos(
    self,
    pos: float
)
```

Jump to a specific position in the current song.

Method `seektime`

```
def seektime(
    self,
    time: datetime.timedelta
)
```

Jump to a specific time stamp in the current song.

Method `stop`

```
def stop(
    self
)
```

Stop playing the current song.

Class `PlayerState`

```
class PlayerState(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

An enumeration representing the current state of the player.

Ancestors (in MRO)

- enum.Enum

Class variables

Variable `finished`

Variable `not_started`

Variable `paused`

Variable `playing`

Class `Queue`

```
class Queue(
    song_list: list[data_models.Song] = [],
    curr_index: int = 0
)
```

Class representing a song queue with methods for interacting with it, for example jumping to the next song, etc.

Class variables

Variable `curr_index` Type: `int`

Variable `current` Type: `data_models.Song`

Variable `duration` Type: `datetime.timedelta`

Variable `elapsed` Type: `datetime.timedelta`

Variable `position` Type: `float`

Variable `song_list` Type: `list[data_models.Song]`

Methods

Method `add_song`

```
def add_song(
    self,
    song: data_models.Song
)
```

Add a song to the queue.

- song – A Song object to add to the end of the queue.

Method `next`

```
def next(
    self
)
```

Go to the next song in the queue. If we're already at the last song, reset the current state but otherwise do nothing.

Method `play_next`

```
def play_next(
    self,
    song: data_models.Song
)
```

Insert a song into the queue to be played directly after the current song.

- song – A Song object to insert to the queue.

Method `prev`

```
def prev(
    self
)
```

Go to the previous song in the queue. If we're already at the first song, reset the current state but otherwise do nothing.

Class `VlcMediaPlayer`

```
class VlcMediaPlayer(
    handlers: dict[player_handlers.HandlerType, typing.Callable[[], NoneType]],
    queue: FTC-Music-Player.models.Queue
)
```

A player that uses VLC to play audio.

Ancestors (in MRO)

- FTC-Music-Player.models.Player

Class variables

Variable `player` Type: `vlc.MediaPlayer`

Module `FTC-Music-Player.player_handlers`

Classes for handling GUI changes from within other code.

Classes

Class `HandlerType`

```
class HandlerType(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

Enumerate all possible UI handlers the Player may need to call.

Ancestors (in MRO)

- enum.Enum

Class variables

Variable `on_source_changed`

Module `FTC-Music-Player.search_models`

Classes for data sent to and from the C# back-end.

Classes

Class `SearchRequest`

```
class SearchRequest(
    query,
    artist_count,
    album_count,
    song_count
)
```

Class modelling data to be sent to the C# back-end.

Class variables

Variable `album_count` Type: `int`

Variable `artist_count` Type: `int`

Variable `query` Type: `str`

Variable `song_count`   Type: `int`

Class `SearchResults`

```
class SearchResults(
    artists: list[data_models.Artist],
    albums: list[data_models.Album],
    songs: list[data_models.Song]
)
```

Class to model data sent back by the C# back-end.

Class variables

Variable `albums`   Type: `list[data_models.Album]`

Variable `artists`   Type: `list[data_models.Artist]`

Variable `songs`   Type: `list[data_models.Song]`

Namespace `FTC-Music-Player.ui`

Sub-modules

- FTC-Music-Player.ui.ui_widgets

Module `FTC-Music-Player.ui.ui_widgets`

Classes

Class `AlbumWidget`

```
class AlbumWidget(
    album: api_client.Youtube.api_models.OnlineAlbum
)
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `ArtistWidget`

```
class ArtistWidget(
    artist: api_client.Youtube.api_models.OnlineArtist
)
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `Home`

```
class Home(
    player: models.Player,
    page: flet_core.page.Page
)
```

Container allows to decorate a control with background color and border and position it with padding, margin and alignment.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Container"

    c1 = ft.Container(
        content=ft.Text("Container with background"),
        bgcolor=ft.colors.AMBER_100,
        padding=5,
    )
    page.add(c1)
```

```
ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/container

Ancestors (in MRO)

- flet_core.container.Container
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Methods

Method `onContentChange`

```
def onContentChange(
    self,
    selectedItem: int
)
```

Method `onKeyboardEvent`

```
def onKeyboardEvent(
    self,
    e: flet_core.page.KeyboardEvent
)
```

Class `HorizontalListView`

```
class HorizontalListView
```

A control that displays its children in a horizontal array.

To cause a child control to expand and fill the available horizontal space, set its expand property.

Example:

```
import flet as ft


def main(page: ft.Page):
    page.title = "Row example"

    page.add(
        ft.Row(
            controls=[
                ft.Container(
                    expand=1,
                    content=ft.Text("Container 1"),
                    bgcolor=ft.colors.GREEN_100,
                ),
                ft.Container(
                    expand=2, content=ft.Text("Container 2"), bgcolor=ft.colors.RED_100
                ),
            ],
        ),
    ),
ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/row

Ancestors (in MRO)

- flet_core.row.Row
- flet_core.constrained_control.ConstrainedControl
- flet_core.scrollable_control.ScrollableControl
- flet_core.control.Control

Class variables

Variable `listView` Type: `flet_core.list_view.ListView`

Methods

Method `append`

```
def append(
    self,
    item: flet_core.control.Control
)
```

Method `scrollLeft`

```
def scrollLeft(
    self,
    e
)
```

Method `scrollRight`

```
def scrollRight(
    self,
    e
)
```

Class `Player_widget`

```
class Player_widget
```

Container allows to decorate a control with background color and border and position it with padding, margin and alignment.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Container"

    c1 = ft.Container(
        content=ft.Text("Container with background"),
        bgcolor=ft.colors.AMBER_100,
        padding=5,
    )
    page.add(c1)

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/container

Ancestors (in MRO)

- flet_core.container.Container
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `SearchResults`

```
class SearchResults(
    results: api_client.Youtube.api_models.SearchResponse,
    player: models.Player
)
```

A scrollable list of controls arranged linearly.

ListView is the most commonly used scrolling control. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.

Example:

```
from time import sleep
import flet as ft

def main(page: ft.Page):
    page.title = "Auto-scrolling ListView"

    lv = ft.ListView(expand=1, spacing=10, padding=20, auto_scroll=True)

    count = 1

    for i in range(0, 60):
        lv.controls.append(ft.Text(f"Line {count}"))
        count += 1

    page.add(lv)

    for i in range(0, 60):
        sleep(1)
        lv.controls.append(ft.Text(f"Line {count}"))
        count += 1
        page.update()

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/listview

Ancestors (in MRO)

- flet_core.list_view.ListView
- flet_core.constrained_control.ConstrainedControl
- flet_core.scrollable_control.ScrollableControl

- flet_core.control.Control

Class `Search_bar_widget`

```
class Search_bar_widget
```

A text field lets the user enter text, either with hardware keyboard or with an onscreen keyboard.

Example:

```
import flet as ft

def main(page: ft.Page):
    def button_clicked(e):
        t.value = f"Textboxes values are:  '{tb1.value}', '{tb2.value}', '{tb3.value}', '{tb4.value}
        page.update()

    t = ft.Text()
    tb1 = ft.TextField(label="Standard")
    tb2 = ft.TextField(label="Disabled", disabled=True, value="First name")
    tb3 = ft.TextField(label="Read-only", read_only=True, value="Last name")
    tb4 = ft.TextField(label="With placeholder", hint_text="Please enter text here")
    tb5 = ft.TextField(label="With an icon", icon=ft.icons.EMOJI_EMOTIONS)
    b = ft.ElevatedButton(text="Submit", on_click=button_clicked)
    page.add(tb1, tb2, tb3, tb4, tb5, b, t)

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textfield

Ancestors (in MRO)

- flet_core.textfield.TextField
- flet_core.form_field_control.FormFieldControl
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `SongWidget`

```
class SongWidget(
    song: data_models.Song,
    songList: list[data_models.Song],
    player: models.Player
)
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )
```

```
ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Methods

Method `onSongClicked`

```
    def onSongClicked(
        self,
        e,
        player: models.Player
    )
```

Class `SquareAlbumWidget`

```
    class SquareAlbumWidget(
        album: api_client.Youtube.api_models.OnlineAlbum
    )
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `SquareArtistWidget`

```
    class SquareArtistWidget(
        artist: api_client.Youtube.api_models.OnlineArtist
    )
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Class `SquareSongWidget`

```
    class SquareSongWidget(
        song: api_client.Youtube.api_models.OnlineSong,
        songList: list[data_models.Song],
        player: models.Player
    )
```

Text buttons are used for the lowest priority actions, especially when presenting multiple options. Text buttons can be placed on a variety of backgrounds. Until the button is interacted with, its container isn't visible.

Example:

```
import flet as ft

def main(page: ft.Page):
    page.title = "Basic text buttons"
    page.add(
        ft.TextButton(text="Text button"),
        ft.TextButton("Disabled button", disabled=True),
    )

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/textbutton

Ancestors (in MRO)

- flet_core.text_button.TextButton
- flet_core.constrained_control.ConstrainedControl
- flet_core.control.Control

Methods

Method `onSongClicked`

```
def onSongClicked(
    self,
    e,
    player: models.Player
)
```

Class `SuggestionsWidget`

```
class SuggestionsWidget(
    results: api_client.Youtube.api_models.GetSuggestionsResponse,
    player: models.Player
)
```

A scrollable list of controls arranged linearly.

ListView is the most commonly used scrolling control. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.

Example:

```
from time import sleep
import flet as ft

def main(page: ft.Page):
    page.title = "Auto-scrolling ListView"

    lv = ft.ListView(expand=1, spacing=10, padding=20, auto_scroll=True)

    count = 1

    for i in range(0, 60):
        lv.controls.append(ft.Text(f"Line {count}"))
        count += 1

    page.add(lv)

    for i in range(0, 60):
        sleep(1)
        lv.controls.append(ft.Text(f"Line {count}"))
        count += 1
        page.update()

ft.app(target=main)
```

Online docs: https://flet.dev/docs/controls/listview

Ancestors (in MRO)

- flet_core.list_view.ListView
- flet_core.constrained_control.ConstrainedControl
- flet_core.scrollable_control.ScrollableControl
- flet_core.control.Control

Class variables

Variable `suggestions` Type: `api_client.Youtube.api_models.GetSuggestionsResponse`

Module `FTC-Music-Player.ui_builder`

Initialise the GUI window.

Classes

Class `UI`

```
class UI(
    player: models.Player
)
```

Methods

Method `open_home`

```
def open_home(
    self,
    page: flet_core.page.Page
)
```

Generated by *pdoc* 0.10.0 (https://pdoc3.github.io).