

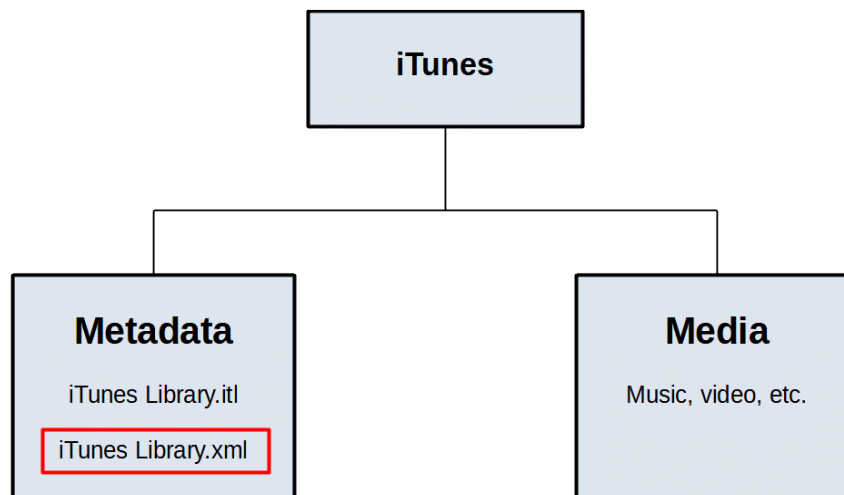
# Unlock iTunes XML with Python

## Introduction

The Extensible Markup Language (XML) is a widely used markup language and text file format for storing and exchanging arbitrary data. A wide variety of applications, including the Apple iTunes application, use the XML format for exchanging or storing library data. In this article, I will show you how you can use Python to read and manipulate data from an iTunes XML library file.

## iTunes library file

iTunes divides its content into two separate categories, media and metadata. Media (music & video) are saved in separate files from metadata. Throughout the rest of this article, I will focus on music information — artist, song title, album title, track number, genre, release year — and refer to it as metadata. The iTunes metadata files have either a .itl or .xml extension. See figure 1.



*Figure 1: iTunes XML metadata file hierarchy*

## Why choose XML?

If you are a music lover like me, there may be times when you want to exchange music metadata between a variety of applications on a variety of digital devices. In my case, I have imported iTunes metadata into database applications to run SQL queries. Some applications use proprietary formats to get optimum performance and efficiency, but limit access with proprietary products. The iTunes .itl format is an example. On the other hand, there are open formats that place more of an emphasis on wide availability and ease of development. XML is in the open format category.

Apple has historically used XML files to export library and playlist metadata to external applications. While the native format for the iTunes library is a binary file with a .itl extension, users can configure iTunes to automatically save an XML copy of the library via Advanced preferences or manually via the File>Export menu. If you prefer to export a subset of the library, the File>Export menu allows you to save playlists to an XML file.

Advantages of the XML format are that it is both human-readable and machine-readable. The benefit of having a human-readable file is to easily inspect and diagnose the data with a simple text editor. Machine readability makes it easy for applications to access the metadata via an application procedural interface (API). Many applications that we use every day — text editors, word processors, spreadsheet editors, presentation tools, and graphics editors — either use XML as the native format or to exchange data with other applications.

## Python XML module

The Python standard library has structured markup tools that include modules for processing XML. In this article, I will describe how metadata in iTunes XML files is organized and how to use the Python `xml.etree.ElementTree` module to navigate and access that data. In order to extract metadata from an iTunes XML playlist or library file, it is useful to understand how the XML file is structured. Songs within a library or playlist file are organized as a list of tracks identified by XML tag `<key>Tracks</key>`. Metadata for each track consists of a dictionary of key-value pairs.

The XML language allows for data types to be defined in a Data Type Definition (DTD) that can be declared either inline inside an XML document, or as an external reference. Apple traditionally uses property list files for their application configuration, thus the term “property list” (plist) in the declaration. See figure 2. Since metadata is structured as elements in an XML tree, the `xml.etree.ElementTree` module, which stores tree elements hierarchically, is an appropriate way to get to those elements. Elements are defined in the external public DTD link specified in the DOCTYPE declaration on the second line of the exported music file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

*Figure 2: DTD in DOCTYPE declaration*

The data types defined in this DTD are: array, data, date, dict, real, integer, string, true, and false. XML elements are also declared in the DTD file. An element is a logical document component that either begins with a start-tag and ends with a matching end-tag. Each metadata element contains a string that is either a key or a value in the music dictionary. Metadata for song tracks is organized as key-value pairs as illustrated in figure 3 where a `<key>` tag with a text string is followed by a `<string>` or `<integer>` tag with a text string.

```
<key>Name</key><string>Justice's Grove</string>
<key>Artist</key><string>Stanley Clarke</string>
<key>Album</key><string>East River Drive</string>
<key>Genre</key><string>Jazz</string>
<key>Track Number</key><integer>1</integer>
<key>Year</key><integer>1993</integer>
```

*Figure 3: XML track metadata example*

Below you will see Python code snippets to show you how to use the following methods and attributes to access metadata in XML elements:

- `parse()`: Sources and parses an external XML file or file object.
- `getroot()`: Returns the root element for the tree.
- `find()`: Returns an instance of the first specified subelement
- `findall()`: Returns a list containing all matching elements in document order.
- `tag`: This attribute specifies the element name as defined in the DTD
- `text`: This attribute specifies the characters between the start and end tag

## Build a tree

The `parse()` method takes input from an XML file object and builds a hierarchical tree as a collection of elements. Each element may have multiple child elements or no child elements. The root element is instantiated by the `getroot()` method. See figure 4.

```
tree = ET.parse(xmlinfile)
root = tree.getroot()
```

*Figure 4: Build tree from XML file*

## Track dictionary

Track metadata is a hybrid of nested lists and dictionaries with key-value pairs. The dictionary containing the metadata is nested two levels below the root. The `find()` method in the two statements shown in figure 5 finds the next level descendant with a `<dict>` tag. The second generation dictionary contains a list of all track dictionaries all track dictionaries in the library or playlist. After locating the track dictionaries, you can work on the child elements that contain the metadata for each track.

```
dict_gen1 = root.find('dict') # 1st generation dictionary
dict_gen2 = dict_gen1.find('dict') # 2nd generation dictionary
```

*Figure 5: First and second generation dictionaries*

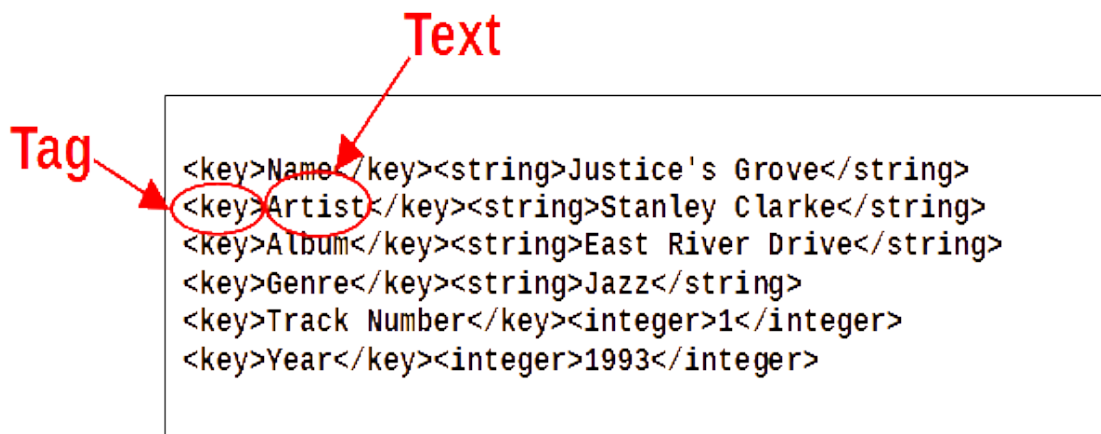
## Element tags and text

To get the track metadata, you need to find the track metadata tags, then retrieve the text from those tags. The following statement (see figure 6) uses the `findall()` method to find all child elements of each track dictionary beneath `dict_gen2` and creates a list object named `tracklist` that contains a list of dictionaries.

```
tracklist = list( dict_gen2.findall('dict') )
```

*Figure 6: Tracklist metadata dictionaries*

Next extract from `tracklist`, nested lists of child elements that have track metadata. Not all `<dict>` tags will contain the metadata that you want, so you can use the `Artist` text string of the `<key>` tag to identify the desired dictionaries. See figures 7 & 8. The element text is accessed with the `element.text` attribute.



```
<key>Name</key><string>Justice's Grove</string>  
<key>Artist</key><string>Stanley Clarke</string>  
<key>Album</key><string>East River Drive</string>  
<key>Genre</key><string>Jazz</string>  
<key>Track Number</key><integer>1</integer>  
<key>Year</key><integer>1993</integer>
```

*Figure 7: Tag and text*

```
itunes_music = [] # Initialize empty list  
for item in tracklist:  
    x = list(item)  
    for i in range( len(x) ):  
        if x[i].text == "Artist":  
            itunes_music.append( list(item) )
```

*Figure 8: Create metadata list for all tracks*

The `itunes_music` list object is a list of all song tracks contained in the input XML file. At this point, you may want to inspect the resulting metadata in the `itunes_music` list. Each `<key>` tag is followed by the corresponding `<string>` or `<integer>` tag. The code in figure 9 assumes that the `tagtrue()` function was previously defined (code not shown for brevity) and tests for `<key>` tag text that matches the desired metadata strings (see figure 3) then prints the key and value pair text strings to your screen.

```
for i in range( len(itunes_music) ):
    for j in range( len(itunes_music[i]) ):
        if tagtrue( itunes_music[i][j].text ):
            print( itunes_music[i][j].text, itunes_music[i][j+1].text )
```

Figure 9: Display metadata

After confirming the metadata, you can then utilize it as you need. Since the list is organized as a two-dimensional array, I will represent it in table form as it would appear in a spreadsheet or database table. Note in figure 10 that the `i` loop or outer loop variable represents rows and the `j` loop or inner loop variable represents columns.

**j = Columns**

	A	B	C	D	E	F
1	Name	Artist	Album	Genre	Track Number	Year
2	Justice's Grove	Stanley Clarke	East River Drive	Jazz	1	1993
3	Fantasy Love	Stanley Clarke	East River Drive	Jazz	2	1993
4	East River Drive	Stanley Clarke	East River Drive	Jazz	4	1993

**i = Rows**

Figure 10: Loop index rows and columns

## Summary

While I don't cover SQL APIs in this article, my ultimate goal in this exercise is to create and update a SQL music database with this metadata. You may choose to use it for other purposes such as analytics, sharing with other applications, or rendering in HTML for web site display. Since XML is a widely used industry standard, there is an abundance of libraries available in almost every programming language. In this case, I've chosen Python. Whatever your goal, it is useful to have a solid understanding of how iTunes music metadata is defined and organized so that you can decide how to get the metadata that you need.

## References

1. Python Software Foundation, The Python Standard Library, version 3.10.4, <https://docs.python.org/3/library/xml.etree.elementtree.html>

2. Apple Inc., iTunes Package Music Specification 5.2, PDF
3. World Wide Web Consortium, XML Technology Schema, <https://www.w3.org/standards/xml/schema>