Code task: OPML to Markdown Converter

The primary purpose of this task is to write a Mac application that allows users to drop one or multiple OPML files, automatically convert them to a suitable Markdown representation and open these Markdown files in Ulysses.

It must include a graphical user interface onto which files can be dropped. The application should indicate visually whether dragged files can be accepted or not accepted. After converting the files to Markdown, the results should be written to a temporary location and auto-imported into Ulysses.

Please know that there is no "perfect" solution here – there are many ways to write such a converter. We will also simplify a few aspects in the task description below. So please read everything in this document carefully before you begin.

We know that there are different opinions on how to write code and how to build user interfaces. Please just try to solve the challenge to the best of your knowledge and experience. There is no template we will compare your solution to, but we will take a look at the code afterward and later discuss aspects of your implementation.

This task should take you around 8 to 10 hours.

Code

The code must be written in Swift and organized as a Xcode project. You can use Swift UI or AppKit as you prefer. The implementation should not be "quick & dirty", but should be a fully implemented and production-ready solution. This includes – among other things – commented code, suitable tests and error handling.

Apart from the system frameworks, no additional libraries or frameworks must be used. Please make sure to separate the OPML parser and the Markdown generator in your software architecture.

There is also no need to write tests with full coverage of all code. It will be fully sufficient if you pick one core aspect of your business logic and test that one thoroughly. When in doubt, try to keep it simple. Inside this ZIP-file there are also some OPML example files you can use for testing.

File formats

In the following, we give a brief description of the OPML and the Markdown file formats. For more information, please refer to the respective specifications.

OPML

OPML is a format to represent simple outlines as XML. The outline is represented as a tree of XML elements carrying the outlined text and further attributes. For this assignment, we are using a subset of the original specification. An OPML file looks similar to the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<opml version="2.0">
  <head>
    <title>Outline title</title>
    <dateCreated>Thu, 30 Jan 2019 09:35:45 GMT</dateCreated>
    <dateModified>Thu, 30 Jan 2019 12:41:16 GMT</dateModified>
    ...
  </head>
  <body>
    <outline text="Simple node" />
    <outline text="Child node">
      <outline text="Nested child node">
        ...
      </outline>
```

```
        ...
      </outline>
      <outline text="Link" type="link" url="https://ulysses.app" />
      <outline text="Comment node" isComment="true">
        <outline text="Child comment node" />
      </outline>
      ...
    </body>
  </opml>
```

On the top level, there is an `opml` element, which must include a version attribute. The `opml` element must have two child elements: `head` and `body`.

The `head` element may contain metadata elements, for instance:

- `title`: The title of the document
- `dateCreated`: The creation date of the document
- `dateModified`: The last modification date of the document

Other metadata elements might be included, but should be ignored for this assignment. If `dateCreated` or `dateModified` has an invalid date format, the element should be ignored.

The `body` element contains one or more `outline` elements.

An `outline` element contains at least a required `text` attribute and may contain additional attributes. For this purpose, the `text` attribute should be treated as plain text, no further escaping is required. An `outline` element may contain `outline` child elements.

The `outline` element may contain an attribute `type`, which specifies how other attributes of the element should be interpreted. If the `type` is `link` and the element contains an attribute `url`, the URL should be associated with the element.

The `outline` element may also contain an attribute `isComment`, whose value might be `true` or `false`. If an element is marked as a comment, all its recursive child elements should be treated as comments as well.

Please note that attribute values such as `link` or `true` should be handled in a case-insensitive fashion. Other attributes (such as `isBreakpoint`, `created` or `type="include"`) should be ignored.

All dates should conform to the following format:

```
Thu, 30 Jan 2019 09:35:45 GMT
```

This means a date string contains the weekday followed by a comma, the date, the time with seconds and the time zone. If any of these components is missing, the date string is treated as invalid. The OPML specification allows some variation of this format, but for our purposes, it's fine to just parse the above format.

Markdown

Markdown is a simple markup language with a minimal set of tags, primarily used for the web. In general, it includes a set of paragraph tags which are applied to whole paragraphs and inline tags, which are applied to a portion of text.

Examples of paragraph tags are headings (heading 1 – `#`, heading 2 – `##`, …) or unordered lists, which are "–" and may be prefixed by tabs for indentation.

Examples of inline tags are links, which contain (among other things) the link name as well as the URL:

```
[Ulysses](https://ulysses.app)
```

Conversion

Please convert OPML to Markdown using the following rules:

- Create one Markdown file for each OPML file.
- The `title` should be converted to a Heading 1.

- The `outline` elements of the `body` element should be converted to heading 2 – heading 6, depending on their nesting level.
- If `outline` elements should be nested further, nested unordered lists should be used.
- URLs should be converted to links.
- `outline` elements which are marked as comments (including their recursive children) should be ignored.
- Set the Markdown file's creation and modification dates to the dates from the OPML's `head` element, if possible. Ulysses will then import the OPML file with this creation and modification date.

Ulysses

To ensure that Ulysses is launched after generating the Markdown file, you have to pass the bundle identifier `com.ulyssesapp.mac` to `NSWorkspace.urlForApplication(withBundleIdentifier:)`[1]

This will provide you the correct application URL for Ulysses. This URL can be then used to enforce that Markdown files are opened with Ulysses: `NSWorkspace.open(:withApplicationAt:configuration:)` [2]

For testing your app, you need to install Ulysses from the Mac App Store. You can use the following link to activate a free one-month usage:

https://sk.ulysses.app/token/792399f2-ff4c-419a-818f-7cc55f94473b

1 https://developer.apple.com/documentation/appkit/nsworkspace/1534053-urlforapplication
2 https://developer.apple.com/documentation/appkit/nsworkspace/3172702-open