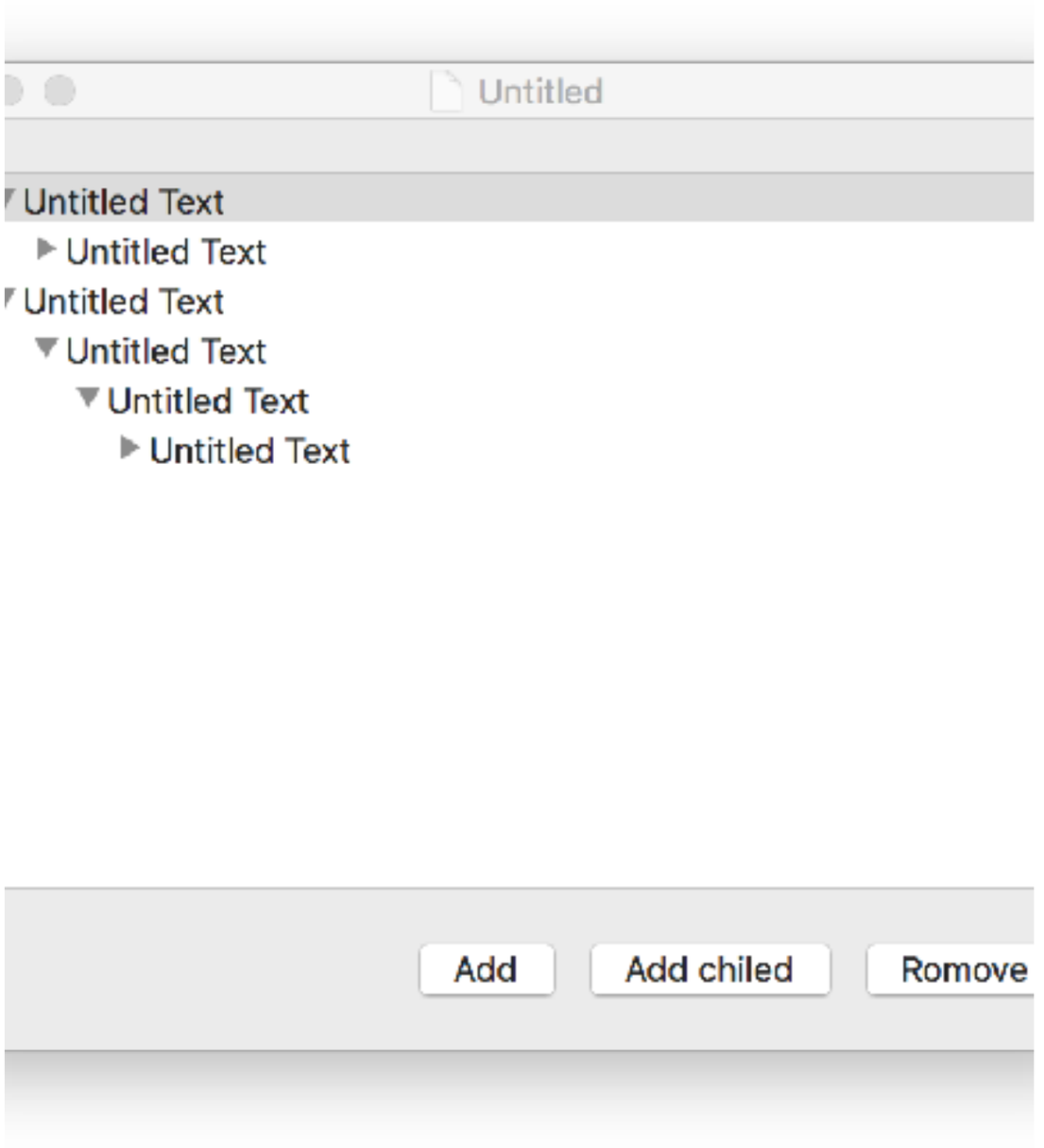
# NSDocumentの複数file formatの実装
# その2

mindtools@mac.com

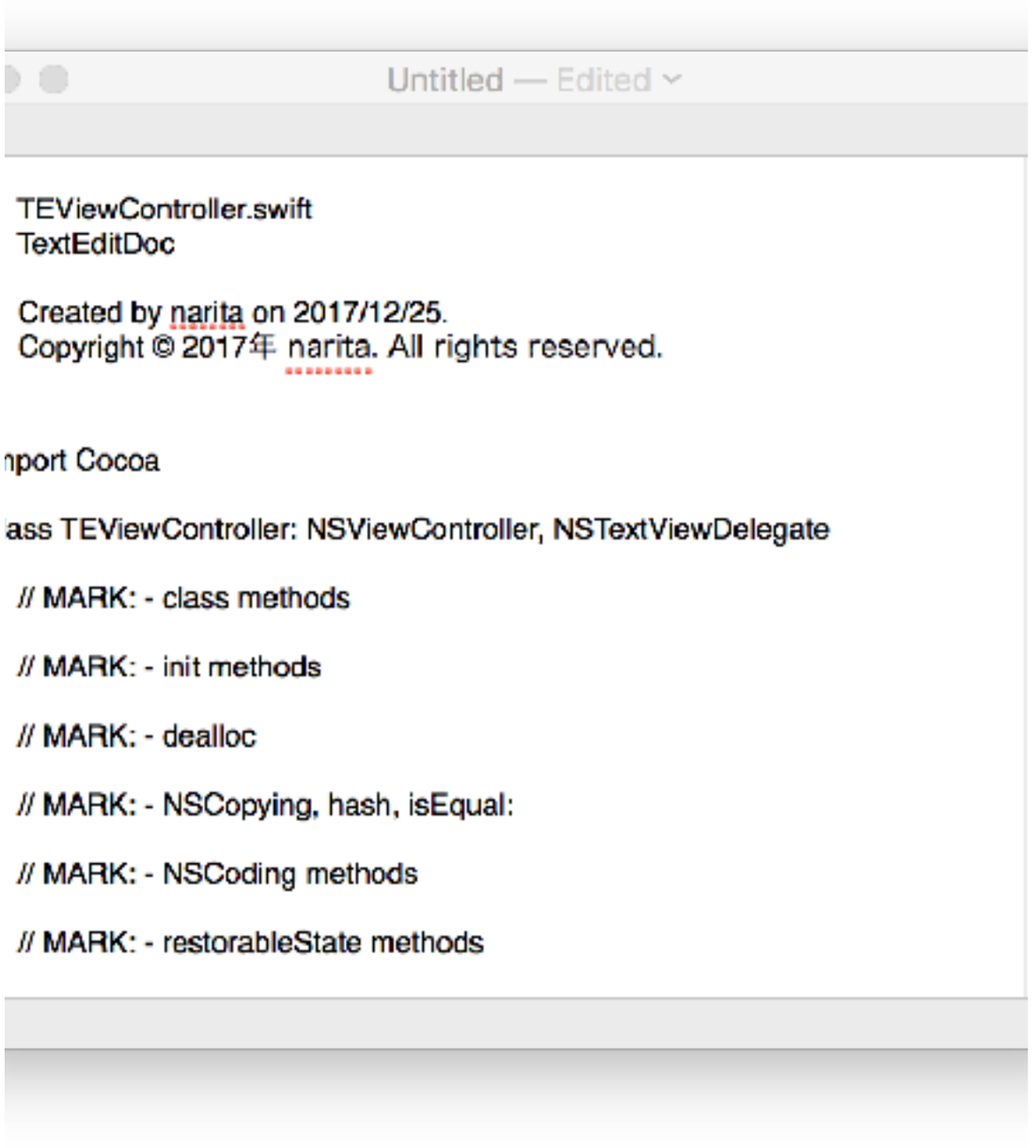
# 目的

- 作成予定のアウトラインプロセッサにOPML形式ファイルをサポートしたい。



# 目標

- 前回作成した最小限TextEditorを足がかりに以下の2つの実装の実験を行う
- 複数ファイルフォーマット書類の新規ドキュメント作成
- OPMLのimport/exportの実装も考慮する。

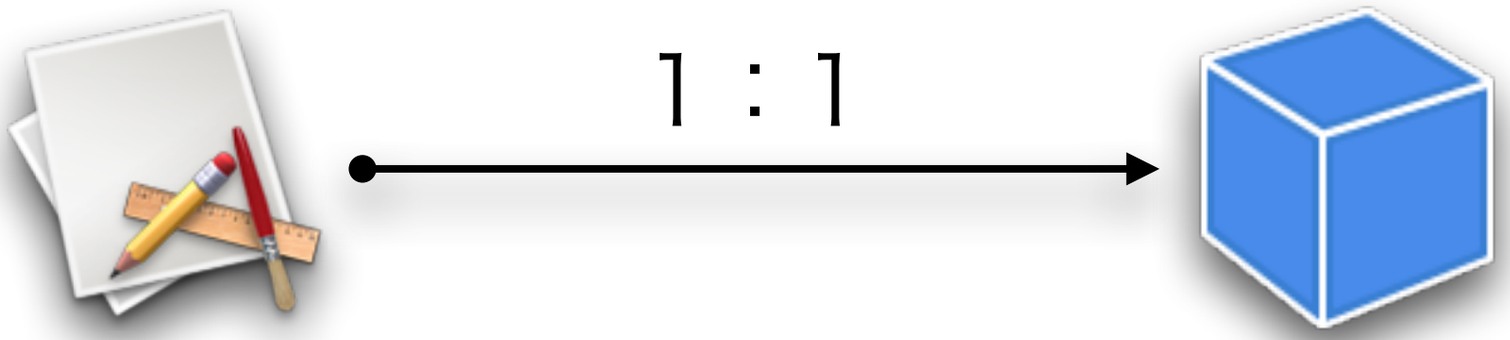


# 作業工程予定

- 新規書類メニュー(New…)を複数作る。New TextやNew RTF
- OPMLをReadOnlyでサポートファイルフォーマットにする
- OPMLをWriteOnlyで(Save To..)でサポートファイルフォーマットにする

# 最小限TextEditor

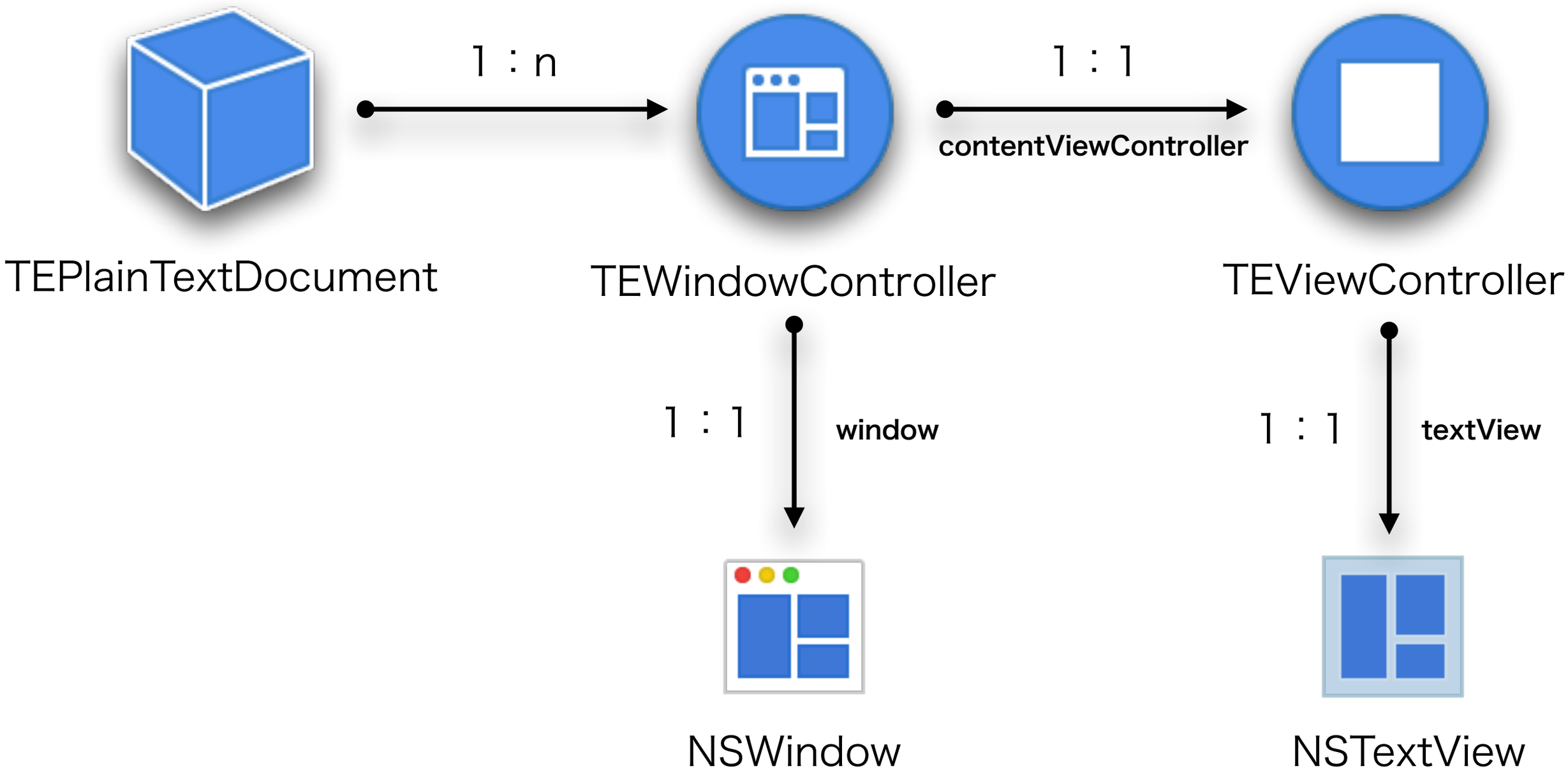
# Object図1




NSApplication　　TEAppDelegate

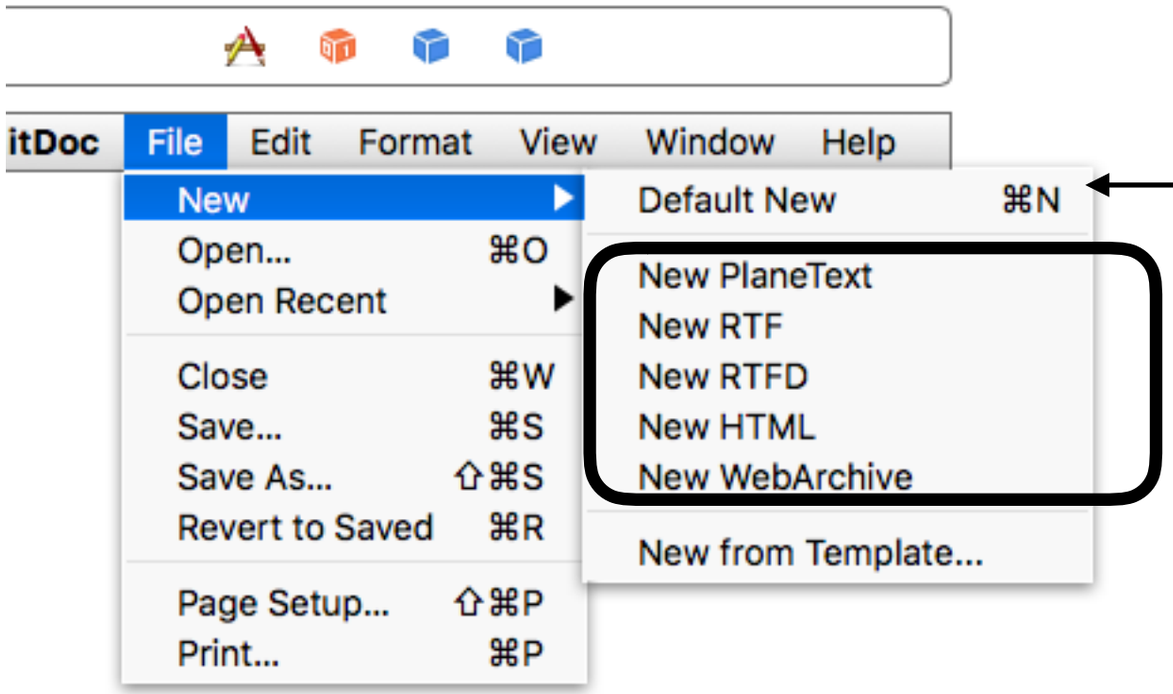
いつも通り

# Object図2




いつも通り

# 新規ドキュメント




デフォルトでは、
info.plistの最初の項目

任意のフォーマットで新規作成
したい場合

# 新規ドキュメント

**NSDocumentController.makeUntitledDocument(ofType: typeName as String) -> NSDocument**

**を呼び出すと、Untitled documentが出来る。newDocument:と同等の動きにするには、**

**- addDocumentを呼び出し、**
**- makeWindowControllersを呼び出し、**
**- showWindow()を呼び出す必要がある。**

# 新規ドキュメント

```
    func newDocumentOfType(ofType typeName: String)  -> TESingleDocument
    {
        var theDocument: TESingleDocument

        do
        {
            theDocument = try NSDocumentController.shared.
             makeUntitledDocument(ofType: typeName as String) as! TESingleDocument

            NSDocumentController.shared.addDocument(theDocument)
            theDocument.makeWindowControllers()
            theDocument.showWindows()

            return theDocument
        }
        catch
        {
            NSException(
                name: NSExceptionName.invalidArgumentException,
                reason: "\(typeName) is not a recognized document type.").raise()
        }

        return TESingleDocument()// ここにきてはいけない。書き方を考える事。
    }

    @IBAction func newPlaneTextDocument(_ sender: NSObject)
    {
        let theDocument = self.newDocumentOfType(ofType: kUTTypePlainText as String)
        theDocument.isRichText = false
    }
```

# Import Document

# Import Document

Appleのドキュメント Document-based applicationでの手法

読み出せるが、書き出せないファイルタイプを自動的にNativeなファイルに変換するには、以下の手順に従う。

1.Info.plistのDocumentTypeにreadOnlyタイプとして登録する。CocoaDocumentクラスは、ここでは、TESingleDocument。

2.read(from fileWrapper: FileWrapper, ofType typeName: String)で、上記のTypeを処理する。もちろん、read(xxxx)の他のメソッドで処理しても良い。

3. setFileType:をNativeのTypeに再設定する。setFileURL:のファイルパスの拡張子をNativeのTypeに設定する。setFileURL:nilにしないと新規ファイル扱いにならない。

# Import Document

1. Info.plistのDocument TypeにreadOnlyタイプとして登録する。Cocoa Documentクラスは、ここでは、TESingleDocument。

| | | | |
|---|---|---|---|
| ▶ Item 4 (Apple Web archive) | | Dictionary | (6 items) |
| ▼ Item 5 (org.opml.opml) | | Dictionary | (6 items) |
| ▼ CFBundleTypeExtensions | ⇕ | Array | (1 item) |
| Item 0 | | String | opml |
| Document Type Name | ⇕ | String | org.opml.opml |
| Document is a package or bundle | ⇕ | Boolean | NO |
| ▼ Document Content Type UTIs | ⇕ | Array | (1 item) |
| Item 0 | | String | org.opml.opml |
| Role | ⇕ | String | Viewer |
| Cocoa NSDocument Class | ⇕ | String | $(PRODUCT_MODULE_NAME).TESingleDocument |
| Executable file | ⇕ | String | $(EXECUTABLE_NAME) |

# Import Document

2. read(from fileWrapper: FileWrapper, ofType typeName: String)で、上記のTypeを処理する。もちろん、read(xxxx)の他のメソッドで処理しても良い。

```
        else if theWorkspace.type( "org.opml.opml", conformsToType: typeName)
        {
            // OPMLからTextを構築してゆく
            let theXMLDoc: XMLDocument = try XMLDocument(data: fileWrapper.regularFileContents!)
            let theXMLNodes = try theXMLDoc.nodes(forXPath: "opml/body/outline")

            var theText: String = ""

            func readFromXML(inXMLNodes: [XMLNode])
            {
                for i: XMLNode in inXMLNodes
                {
                    if let localText: String? = (i as! XMLElement).attribute(forName: "text")?.objectValue as! String!
                    {
                        theText.append( localText! )
                        theText.append( "\r" )
                    }

                    // ここで、iにsubnodeがあれば再帰する
                    if i.children != nil
                    {
                        readFromXML(inXMLNodes: i.children! )
                    }
                }
            }

            readFromXML(inXMLNodes: theXMLNodes)

            self.textStorage = NSTextStorage(attributedString: NSAttributedString(string: theText))

            self.isRichText = false
        }
        else
        {
            .....
        }
    }
```

# Import Document

2. read(from fileWrapper: FileWrapper, ofType typeName: String)で、上記のTypeを処理する。もちろん、read(xxxx)の他のメソッドで処理しても良い。

```
    public convenience init(contentsOf url: URL, ofType typeName: String) throws
    {
        self.init()

        try self.read(from: url, ofType: typeName)

        // typeNameがOPMLとそれ以外で分岐
        if NSWorkspace.shared.type("org.opml.opml", conformsToType: typeName)
        {
            self.fileURL = nil
            self.fileType = kUTTypeRTF as String

            self.fileModificationDate = Date()

            // ファイルタイプを変更したので、"Undo不可能な編集"をした事にする
            self.updateChangeCount(NSDocument.ChangeType.changeDone)

            self.isDraft = true
        }
        else
        {
            self.fileURL = url
            self.fileType = typeName

            let theAttr = try FileManager.default.attributesOfFileSystem(forPath: url.path)

            self.fileModificationDate = theAttr[FileAttributeKey.modificationDate] as? Date
        }
    }
```

# Import Document

3. setFileType:をNativeのTypeに再設定する。setFileURL:のファイルパスの拡張子をNativeのTypeに設定する。setFileURL:nilにしないと新規ファイル扱いにならない。

```
    public convenience init(contentsOf url: URL, ofType typeName: String) throws
    {
        self.init()

        try self.read(from: url, ofType: typeName)

        // typeNameがOPMLとそれ以外で分岐
        if NSWorkspace.shared.type("org.opml.opml", conformsToType: typeName)
        {
            self.fileURL = nil
            self.fileType = kUTTypeRTF as String

            self.fileModificationDate = Date()

            // ファイルタイプを変更したので、"Undo不可能な編集"をした事にする
            self.updateChangeCount(NSDocument.ChangeType.changeDone)

            self.isDraft = true
        }
        else
        {
            self.fileURL = url
            self.fileType = typeName

            let theAttr = try FileManager.default.attributesOfFileSystem(forPath: url.path)

            self.fileModificationDate = theAttr[FileAttributeKey.modificationDate] as? Date
        }
    }
```

# Import Document

ファイルと結び付けない場合は、以下のエラーがログに記録される。

2018-02-15 20:53:52.072223+0900 TextEditDoc[17998:23615980] MessageTracer: load_domain_whitelist_search_tree:73: Search tree file's format version number (0) is not supported
2018-02-15 20:53:52.072274+0900 TextEditDoc[17998:23615980] MessageTracer: Falling back to default whitelist

現在は、原因は不明。
ファイルと結びつけられないために、autoSaveが出来ない。state restoreが出来ない等でログを吐いていると想像している。

解決方法は、
NSDocumentController
の内部の時点で、分岐させて、
NSDocumentController.makeDocument(for: nil, withContentsOf: contentsURL, ofType: typeName)
を使って、Documentを作成する方法で、読み込みファイルのファイルパスに依存させない方法を取れば良いかもしれない。

# Exportメモ

1. Info.plsitへ追記。
2. メニューに”Save To”項目を追加
3. 書き出しのコードはfileWrapper()に書く

# Exportメモ

## 1. Info.plsitへ追記。

古い日本語ドキュメントでは、”NSExportableAs”を使えと書いている。今は、”*NSExportableTypes*”に置き換えられている。

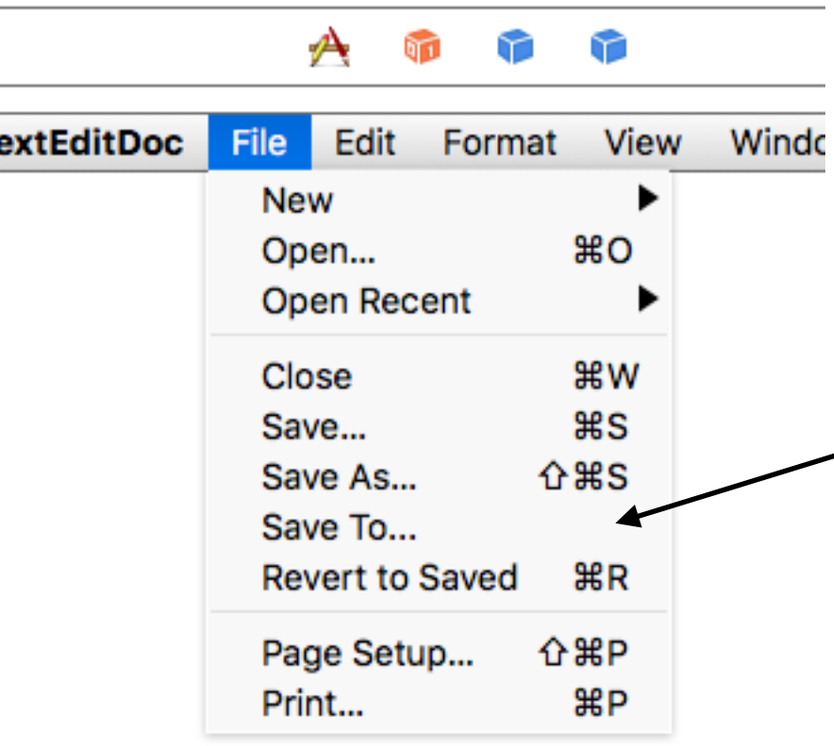
*NSExportableTypes*キーをつかって書き出せるファイルのタイプをセットします。

| | | |
|---|---|---|
| ▼ Document types | Array | (7 items) |
| ▼ Item 0 (NSStringPboardType) | Dictionary | (7 items) |
| ▶ CFBundleTypeExtensions | Array | (1 item) |
| Icon File Name | String | |
| Document Type Name | String | NSStringPboardType |
| Role | String | Editor |
| ▼ Document Content Type UTIs | Array | (1 item) |
| Item 0 | String | public.plain-text |
| Cocoa NSDocument Class | String | $(PRODUCT_MODULE_NAME).TESingleDocument |
| ▼ Exportable Type UTIs | Array | (1 item) |
| Item 0 | String | com.adobe.pdf |
| ▼ Item 1 (NSRTFPboardType) | Dictionary | (7 items) |
| ▼ CFBundleTypeExtensions | Array | (1 item) |
| Item 0 | String | rtf |
| Icon File Name | String | |
| Document Type Name | String | NSRTFPboardType |
| Role | String | Editor |
| ▼ Document Content Type UTIs | Array | (1 item) |
| Item 0 | String | public.rtf |
| Cocoa NSDocument Class | String | $(PRODUCT_MODULE_NAME).TESingleDocument |
| ▼ Exportable Type UTIs | Array | (1 item) |
| Item 0 | String | com.adobe.pdf |

**PDFに書き出せると**
**表記してます**

# Exportメモ

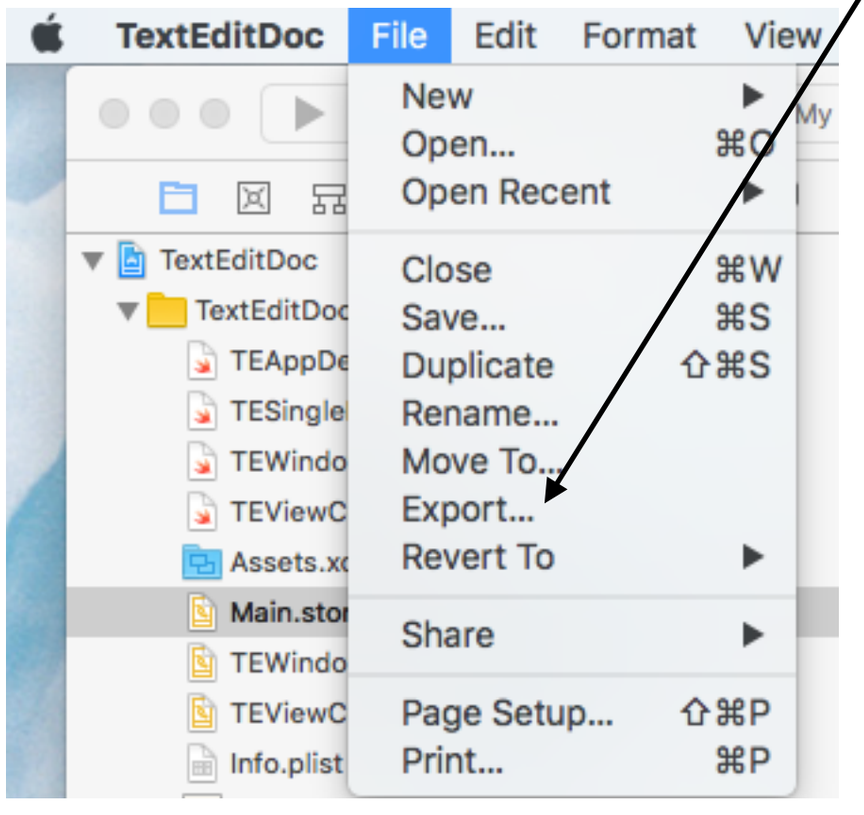
メニューに”Save To”項目を追加




saveDocumentTo:に設定

なぜか、名称が書き変わるの

注意



# Exportメモ

## 3. 書き出しのコードはTESingleDocument.fileWrapper()に書く

```
    override func fileWrapper(ofType typeName: String) throws -> FileWrapper
    {
        .
        .
        .


        }
        else if theWorkspace.type(kUTTypePDF as String, conformsToType: typeName)
        {
            // PDFで出力。ただしめんどくさいので1ページのみ
            if let theView: NSView = self.windowControllers[0].window?.contentView!
            {
                let theData = theView.dataWithPDF(inside: theView.bounds)

                return FileWrapper(regularFileWithContents: theData)
            }
        }

        .
        .
        .
    }
```

# 今後の拡張

- 現状の方法で、ファイルフォーマットを拡張すると、Documentクラスも、AppDelegateクラスもパンクしそう。
- ファイルフォーマットが増える場合は、ストラテジパターンで別クラスにするべき