Collabora Productivity

Collabora Office as an app on iOS

Tor Lillqvist

Software Engineer at Collabora Productivity

About us

Collabora Ltd.

- Leading Open Source Consultancy
- 13 years of experience. 100+ people

Collabora Productivity Ltd.

- Dedicated to Enterprise LibreOffice
- Developers with 10+ years of experience with the codebase
- Provides Level-3 support (code issues) to all SUSE LibreOffice clients
- Architects of OpenXML filters in LibreOffice

History, overview

- First LibreOffice cross-compilation efforts (to iOS, Android, and Windows) in 2011
- Initially just a spare time effort with few concrete plans
- CloudOn funded the iOS effort around 2014 for some time, but before that resulted in any real product the company was acquired, plans changed, and the work fizzled out
- LibreOffice Online started around the same time, with “tiled rendering” of document view
- For iOS a barebones test app was kept more or less working, and a different approach was started by Jan Iversen in 2017 or so, but not finished
- In 2018 a fresh start on iOS app based on LibreOfficeKit, and the LibreOffice Online codebase (both its C++ “server” code (as applicable) and JavaScript user interface code). Funded by Collabora Productivity and Adfinis SyGroup

Details of LibreOffice cross-compilation

- Configure script gets run twice, for “build” and “host” (run-time) platforms
- Build only build-time tools for the build platform
- For iOS (and Android) produce only static archives, no dynamic libraries
- No unit testing. (Interesting and relevant unit tests could be combined into an app of their own. One small such app is actually there already, in the Online repo.)
- No app produced when building in core

LibreOfficeKit

- Originally intended to be a mostly C-like API for very basic functionality like loading and saving documents
- For cases where the “normal” UNO API is seen as too complicated
- Later extended with the “tiled rendering” concept where rectangular tiles of a view of the document are rendered on request by a client, and other features
- Used by LibreOffice Online

LibreOffice Online

- Server-based solution with several processes: One master “wsd” process, one “broker” process, and one “kit” process per open document (with potentially multiple editing end user clients), with strict isolation (chroot etc) between the “kit” processes and the rest of the world
- Browser-based client, with lots of JavaScript
- Client-server communication uses WebSockets
- Also communication between the above server processes uses WebSockets

Combining all the above in a mobile app

- LibreOffice core C++ code (relevant parts) and Online server-side C++ code (relevant parts) run in the app process
- Also, a little additional platform-specific app code (in Objective-C) (not Swift, to make it easier to interface with the C++ bits of the Online server code)
- The HTML and JavaScript client parts run in a WebKit WebView that the platform-specific code manages. (On iOS each WKWebView is actually for safety and performance reasons a separate process, but that is mostly transparent.)

Combining all the above in a mobile app, continued

- Communication between JavaScript and native code (Objective-C, C++) using platform-provided APIs. The native code requests the WebView JavaScript engine to perform a snippet of JavaScript, JavaScript code sends a message that invokes a callback in the native code
- Communication between parts of server code (that in “real” Online are different processes, but in the app just threads) uses a small home-grown API with in-process buffers, mutexes etc (no sockets or other system IPC mechanism) that to the other code offers a mostly equivalent API as in “real” Online, to avoid significant code changes

Other platforms

- Same basic setup, just the platform-specific code (the Objective-C++ code in the iOS case) needs to be written separately.
- As an example and experiment, a rudimentary gtk+ one was written. Mainly in the (vain?) hope that people with Linux only would be interested in working on the JavaScript side of the code without even having to use any mobile device

Details of building the app

- All static libraries built in LibreOffice core get listed in a file that later is used when building the app in an Xcode project
- UNO component instantiation does not use dynamic linking but a map from component (or constructor) name to function pointer
- A Python script in core generates those maps, based on what the app is found to need. Somewhat ad-hoc, yes
- Only functions actually referenced get linked it
- Configuration files, rc files, etc are mostly as in a normal LibreOffice

Code walk-through: Initialisation

- AppDelegate.mm, application:didFinishLaunchingWithOptions method
- Initialise Poco logging
- Locale and language stuff
- Template download for customer-specific cases
- Initialise LibreOfficeKit
- Start a thread that creates a LOOLWSD object (corresponds to the “wsd” process in real Online), runs it, repeat.

Code walk-through: Document browsing

- On iOS, browsing documents in iCloud or on file storage extensions like Nextcloud comes “for free”: UIDocumentBrowserViewController
- DocumentBrowserViewController.mm
- When the user selects a document to edit, a Document object is created

Code walk-through: Document loading

- Document.mm
- Creates a DocumentViewController and passes it the URL of the HTML page that will contain the document view and the Online UI, and the document URL, the UI language, and some other things as query parameters
- In this file is the send2JS() code that sends what corresponds to a WebSocket “message” to the JavaScript bits
- Send2JS() xecutes one JavaScript expression in the WebView. It can not contain binary data, such is base64 encoded, and turned into an ArrayBuffer as the receiving code expects. (Here is a potential performance issue: the tile PNG images are base64 encoded, then unencoded, then encoded again for use as data: URLs.)

Code walk-through: Document view

- DocumentViewController.mm
- Creates the WebView (WKWebView)
- In this file is the callback that corresponds to receiving WebSocket messages in the server in real Online

Collabora Productivity

Thanks to Adfinis SyGroup for participating in funding this work

Collabora Productivity

Thank you for your attention

Keep Calm and Crush the Patriarchy

Tor Lillqvist

tml@collabora.com