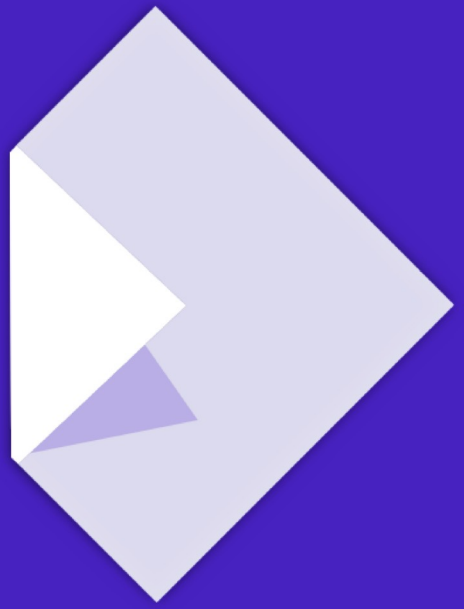
Collabora Office as an iOS app on the iPad

LIBOCON19



Collabora Productivity

By Tor Lillqvist

Software Engineer at Collabora Productivity

@TorLillqvist

“Men are afraid that women will laugh at them.
Women are afraid that men will kill them.”

—Margaret Atwood

About us

Collabora Ltd.

- Leading Open Source Consultancy
- 10+ 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

Details of LibreOffice cross-compilation

- Configure script run twice, for “build” and “host” (run-time) platforms
- For the build platform we build only build-time tools
- For iOS (and Android) the build produces only static archives, no dynamic libraries
- No unit testing
- No app code in the core repo

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 sponsored the iOS effort around 2014 for some time, but before that resulted in any real product the company was acquired, their product plans changed, and the work fizzled out

History, overview, continued

- Not much happening for many years; a barebones test app was kept more or less working, and a different approach was started by Jan Iversen, but not finished
- In 2018 a fresh start based on the Collabora Online codebase, both its C++ “server” code (as applicable) and JavaScript user interface code. Partially funded by Adfinis SyGroup from Switzerland

LibreOfficeKit

- Originally intended to be a mostly C-like API for the basic functionality of 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 by the core code on request by client code, and other features
- The CloudOn app used LibreOfficeKit
- Used by Collabora Online

Collabora 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) for the “kit” processes
- Browser-based client, with lots of JavaScript
- Client-server communication uses WebSockets
- Also communication between processes uses WebSockets

Combining all the above in an iOS app

- LibreOffice core C++ code and Online server-side C++ code run as the app process
- Some 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. (Also, I don’t know Swift yet.)
- 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 an iOS app, continued

- Communication between JavaScript and C++ using platform-provided APIs. The native code requests the WebView JavaScript engine to perform a snippet of JavaScript. JavaScript code invokes a callback in the native code
- Communication between the parts of server code that in normal Online are different processes uses in-process buffers (no sockets or other system IPC mechanism)

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 only Linux might be interested in working on the JavaScript side of the code

Details of building the app

- All static libraries built in LibreOffice core get listed in a file
- Name of that file is passed to linker when building the app binary
- UNO component instantiation does not use dynamic linking but a map from component (or constructor) name to function pointer, thus all required UNO components get statically linked in

Details of building the app, library list

/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libcomphelper.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libconfigmgrlo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libcppcanvaslo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libctllo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libcuilo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdatelo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdbtoolslo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdeployment.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdeploymentgui.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdeploymentmisclo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdesktopbe1lo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libdrawinglayerlo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libeditenglo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libembobj.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libemboleobj.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libemfiolo.a
/Volumes/TML13/lo/ios-optimised-cp-6.0/instdir/program/libepoxy.a

Details of building the app, linker command in Xcode

/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang++ -arch arm64 -isysroot /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS12.4.sdk -L/Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Products/Debug-iphoneos -F/Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Products/Debug-iphoneos -filelist /Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Intermediates.noindex/Mobile.build/Debug-iphoneos/Mobile.build/Objects-normal/arm64/Mobile.LinkFileList -Xlinker -rpath -Xlinker @executable_path/Frameworks -Xlinker -map -Xlinker /Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Intermediates.noindex/Mobile.build/Debug-iphoneos/Mobile.build/Mobile-LinkMap-normal-arm64.txt -miphoneos-version-min=11.4 -dead_strip -Xlinker -object_path_lto -Xlinker /Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Intermediates.noindex/Mobile.build/Debug-iphoneos/Mobile.build/Objects-normal/arm64/Mobile_lto.o -Xlinker -export_dynamic -Xlinker -no_deduplicate -stdlib=libc++ -fobjc-arc -fobjc-link-runtime **`-filelist /Volumes/TML13/lo/online-ios-co-4/ios/../lobuilddir-symlink/workdir/CustomTarget/ios/ios-all-static-libs.list`** -L /Volumes/TML13/lo/online-ios-co-4/ios/../pocolib-symlink -lPocoFoundationd -lPocoUtild -lPocoXMLd -lPocoJSONd -lPocoNetd -framework MobileCoreServices -framework WebKit -lz -liconv -Xlinker -dependency_info -Xlinker /Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Intermediates.noindex/Mobile.build/Debug-iphoneos/Mobile.build/Objects-normal/arm64/Mobile_dependency_info.dat -o /Users/tml/Library/Developer/Xcode/DerivedData/Mobile-gpxwjbpmxlnjkxafxmjgsdmsxwnz/Build/Products/Debug-iphoneos/Mobile.app/Mobile

Details of building the app, UNO component instantiation

```
    assert(factory != nullptr && !factory->is());
#if defined DISABLE_DYNLOADING
    assert(!environment.isEmpty());
    if (constructor.isEmpty()) {
        css::uno::Environment curEnv(css::uno::Environment::getCurrent());
        css::uno::Environment env(getEnvironment(environment, implementation));
        if (!(curEnv.is() && env.is())) {
            throw css::loader::CannotActivateFactoryException(
                "cannot get environments",
                css::uno::Reference<css::uno::XInterface>());
        }
        if (curEnv.get() != env.get()) {
            std::abort();//TODO
        }
        rtl::OUString name(prefix == "direct" ? implementation : uri);
        SAL_INFO("cppuhelper.shlib", "prefix=" << prefix << " implementation=" << implementation << " uri=" << uri);
        lib_to_factory_mapping const * map = lo_get_factory_map();
        component_getFactoryFunc fp = 0;
        for (int i = 0; map[i].name != 0; ++i) {
            if (name.equalsAscii(map[i].name)) {
                fp = map[i].component_getFactory_function;
                break;
            }
        }
        if (fp == 0) {
            SAL_WARN("cppuhelper", "unknown factory name \"" << name << "\"");
#if defined IOS && !defined SAL_LOG_WARN
            // If the above SAL_WARN expanded to nothing, print to stderr...
            fprintf(stderr, "Unknown factory name %s\n", OUStringToOString(name, RTL_TEXTENCODING_UTF8).getStr());
#endif
```

Details of building the app, continued

- A Python script in core generates those maps, based on what the app is observed to need, based on error messages displayed when runing the app. This is somewhat ad-hoc, sure
- Only functions actually needed get linked in
- Configuration files, rc files, etc are mostly as in a normal LibreOffice

Details of building the app, Python script to generate maps to UNO component constructor functions

```
core_factory_list = [
    ("libembobj.a", "embobj_component_getFactory"),
    ("libevtattlo.a", "evtatt_component_getFactory"),
    ("libcomphelper.a", "comphelp_component_getFactory"),
    ("libconfigmgrlo.a", "configmgr_component_getFactory"),
    ("libdrawinglayerlo.a", "drawinglayer_component_getFactory"),
    ("libemfiolo.a", "emfio_component_getFactory"),
    ("libfilterconfiglo.a", "filterconfig1_component_getFactory"),
    ("libfsstoragelo.a", "fsstorage_component_getFactory"),
    ("libhyphenlo.a", "hyphen_component_getFactory"),
    ("libi18npoollo.a", "i18npool_component_getFactory"),
    ("libi18nsearchlo.a", "i18nsearch_component_getFactory"),
    ("libinvocadaptlo.a", "invocadapt_component_getFactory"),
    ("liblnglo.a", "lng_component_getFactory"),
    ("liblnthlo.a", "lnth_component_getFactory"),
    ("liblocalebe1lo.a", "localebe1_component_getFactory"),
    ("libooxlo.a", "oox_component_getFactory"),
    ("libpackage2.a", "package2_component_getFactory"),
    ("libsmlo.a", "sm_component_getFactory"),
    ("libsrtrs1.a", "srtrs1_component_getFactory"),
    ("libstoragefdlo.a", "storagefd_component_getFactory"),
    ("libucb1.a", "ucb_component_getFactory"),
solenv/bin/native-code.py
```

Details of building the app, Python script output

```
void * filterconfig1_component_getFactory( const char* , void* , void* );
void * fsstorage_component_getFactory( const char* , void* , void* );
void * hyphen_component_getFactory( const char* , void* , void* );
void * i18npool_component_getFactory( const char* , void* , void* );
void * i18nsearch_component_getFactory( const char* , void* , void* );
void * invocadapt_component_getFactory( const char* , void* , void* );
void * lng_component_getFactory( const char* , void* , void* );
void * lnth_component_getFactory( const char* , void* , void* );
void * localebe1_component_getFactory( const char* , void* , void* );
void * mcnttype_component_getFactory( const char* , void* , void* );
void * msword_component_getFactory( const char* , void* , void* );
void * mtfrenderer_component_getFactory( const char* , void* , void* );
void * odfflatxml_component_getFactory( const char* , void* , void* );
void * oox_component_getFactory( const char* , void* , void* );
void * package2_component_getFactory( const char* , void* , void* );
void * pdffilter_component_getFactory( const char* , void* , void* );
void * pricing_component_getFactory( const char* , void* , void* );
void * proxyfac_component_getFactory( const char* , void* , void* );
workdir/CustomTarget/ios/native-code.h
```

Details of building the app, Python script output

```
void * com_sun_star_chart2_LogarithmicScaling_get_implementation( void *, void * );
void * com_sun_star_chart2_PowerScaling_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_GraphicExporter_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_VisioImportFilter_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_BasicPaneFactory_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_BasicToolBarFactory_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_BasicViewFactory_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_PresentationFactoryProvider_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_ResourceID_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_configuration_ConfigurationController_get_implementation( void *, void * );
void * com_sun_star_comp_Draw_framework_module_ModuleController_get_implementation( void *, void * );
void * com_sun_star_comp_MemoryStream( void *, void * );
void * com_sun_star_comp_SequenceInputStreamService( void *, void * );
void * com_sun_star_comp_SequenceOutputStreamService( void *, void * );
void * com_sun_star_comp_Svx_GraphicExportHelper_get_implementation( void *, void * );
void * com_sun_star_comp_Svx_GraphicImportHelper_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_RtfFilter_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_WriterFilter_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_XMLOasisContentExporter_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_XMLOasisContentImporter_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_XMLOasisImporter_get_implementation( void *, void * );
void * com_sun_star_comp_Writer_XMLOasisMetaExporter_get_implementation( void *, void * );
```

Details of building the app, Python script output

```
void makeArgEdit();
void makeAutoCompleteMultiListBox();
void makeAutoCorrEdit();
void makeAutoFormatPreview();
void makeBackgroundPreview();
void makeBmpWindow();
void makeBookmarksBox();
void makeCaptionComboBox();
void makeCategoryListBox();
void makeClassificationEditView();
void makeColorConfigCtrl();
void makeColorFieldControl();
void makeColorPreviewControl();
void makeColorSliderControl();
void makeColumnEdit();
void makeColumnValueSet();
void makeCommandCategoryListBox();
void makeConditionEdit();
```

Details of building the app, Python script output

```
static struct { const char *name; void(*func)(); } custom_widgets[] = {
    { "makeArgEdit", makeArgEdit },
    { "makeAutoCompleteMultiListBox", makeAutoCompleteMultiListBox },
    { "makeAutoCorrEdit", makeAutoCorrEdit },
    { "makeAutoFormatPreview", makeAutoFormatPreview },
    { "makeBackgroundPreview", makeBackgroundPreview },
    { "makeBmpWindow", makeBmpWindow },
    { "makeBookmarksBox", makeBookmarksBox },
    { "makeCaptionComboBox", makeCaptionComboBox },
    { "makeCategoryListBox", makeCategoryListBox },
    { "makeClassificationEditView", makeClassificationEditView },
    { "makeColorConfigCtrl", makeColorConfigCtrl },
    { "makeColorFieldControl", makeColorFieldControl },
    { "makeColorPreviewControl", makeColorPreviewControl },
    { "makeColorSliderControl", makeColorSliderControl },
    { "makeColumnEdit", makeColumnEdit },
    { "makeColumnValueSet", makeColumnValueSet },
```

Code walk-through: Initialisation

- AppDelegate.mm, application:didFinishLaunchingWithOptions method
- Initialise Poco logging
- Locale and UI language setup
- Template download for customer-specific cases
- 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, 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, and the document URL, the UI language, and some other things as query parameters
- Here is the send2JS() code that sends what corresponds to a WebSocket “message” to the JavaScript bits
- Send2JS() executes one JavaScript expression in the WebView that “receives” a base64-encoded message (as in the WebSocket protocol used in normal Online)

Code walk-through: Document view

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

Code walk-through: HTML shown in the WebView

- Generated from loleaflet.html.m4 (yikes)
- Mostly the same as in normal Online
- M4 is used to do some conditional stuff that sets flags used in the JavaScript to check whether it is running in the iOS app

Code walk-through: Localisation

- Very different from in normal Online
- Localisation for all bundled UI languages is massaged by a Perl script (yay) into one JSON structure
- The _ function in the iOS app case looks up the translation in that

Code walk-through: JavaScript

- global.js: gets inserted into loleaflet.html. The _ function is here
- Rest of JavaScript is all minimized and bundled into bundle.js
- L.Map.loadDocument(): This starts the magic. No real WebSockets are used, but a “FakeWebSocket” that uses the WebView JS side API to talk to the embedding app

Code walk-through: JavaScript, more

- Most of the JavaScript works exactly the same in normal Online and in the app. The differences are described below
- As the app is developed for the iPad, there is less need to desperately save screen space than on the iPhone. The menubar is shown all the time for instance

Code walk-through: more differences to Online

- As there is no HTTP involved in the app, functionality that in normal Online uses separate HTTP requests must use other mechanism to cause code in the embedding app to be invoked. For instance printing and PDF export

Building the app

- First you build the core part, separately, using a normal “make”. This produces the static archives mentioned earlier
- The JavaScript bits are massaged into a bundle.js etc, so far on a Linux machine because of npm etc that is easier to use on Linux. Then copied over to the Mac
- Then you build the Online C++ and app-specific Objective-C++ parts in Xcode

Debugging?

- The C++ and Objective-C++ code can be debugged in Xcode, the JavaScript code in Safari on a Mac the iPad is attached to. Occasional minor hiccups but in general works as expected

Collabora Productivity

Thank you

Tor Lillqvist

tml@collabora.com @TorLillqvist

> “The man who prays is the one who thinks that god has arranged matters all wrong, but who also thinks that he can instruct god how to put them right.”
>
> —Christopher Hitchens, Mortality