



Collabora Productivity

Dialog Tunneling in LibreOffice Online

By Jan Holešovský

Collabora Productivity

kendy@collabora.com @JHolesovsky +holesovsky skype: janholes

LibreOffice Online: Server part

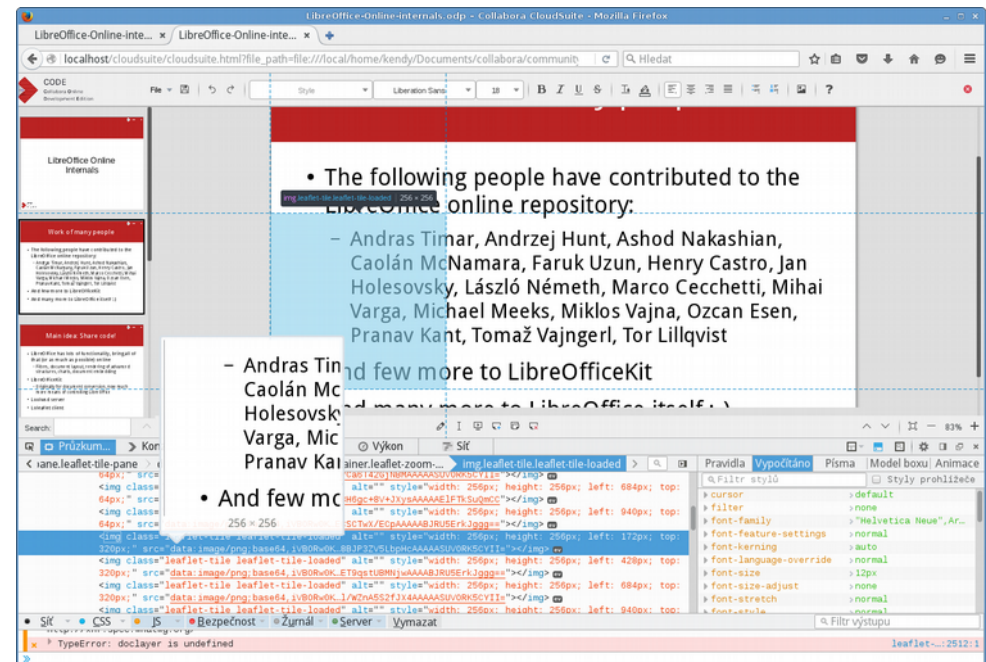
The Websocket Daemon - loolwsd

- Manages communication with file storage via WOPI protocol
- Spawns LibreOffice instances via LibreOfficeKit (LOK) and manages their lifecycle
 - These take care of rendering of the document
- Manages the user's interaction with the document
 - Passing commands to LOK
 - Passing callbacks back to the JavaScript clients
- All this is in C++

LibreOffice Online: Client part

Loleaflet

- Written in JavaScript, based on 'leaflet' - framework for map rendering
- Communicates with loolwsd
- The document itself consists of tiles:
 - Menus, toolbars, status bar
 - All that is JS

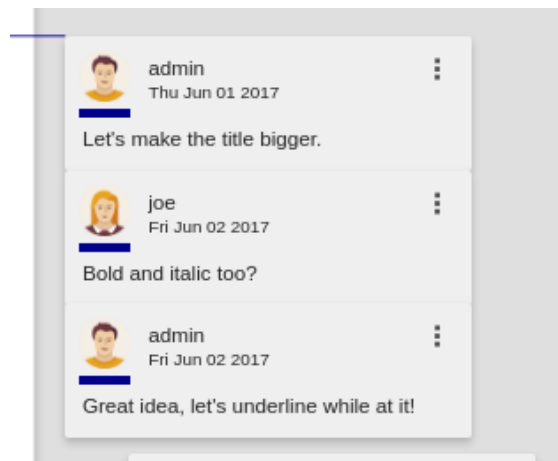


But: it's very impractical to reimplement everything in JS..

Finding the Right Balance: JS vs. Core

Initially everything was rendered by LibreOffice

- In the early prototypes – no tiles, just gtk Broadway
- Then we decided to use the tiled approach
- Cursors, selections – all that turned to be impractical in tiles, and we started rendering that separately, in an overlay
- Comments and redlining were next, those needed too much interaction when in tiles
 - Also they look better in JS (possibility to animate etc.)



But what about dialogs?

We started adding JS ones

- Find / replace, special character, insert table, ...
- Lengthy process! Needed something better...

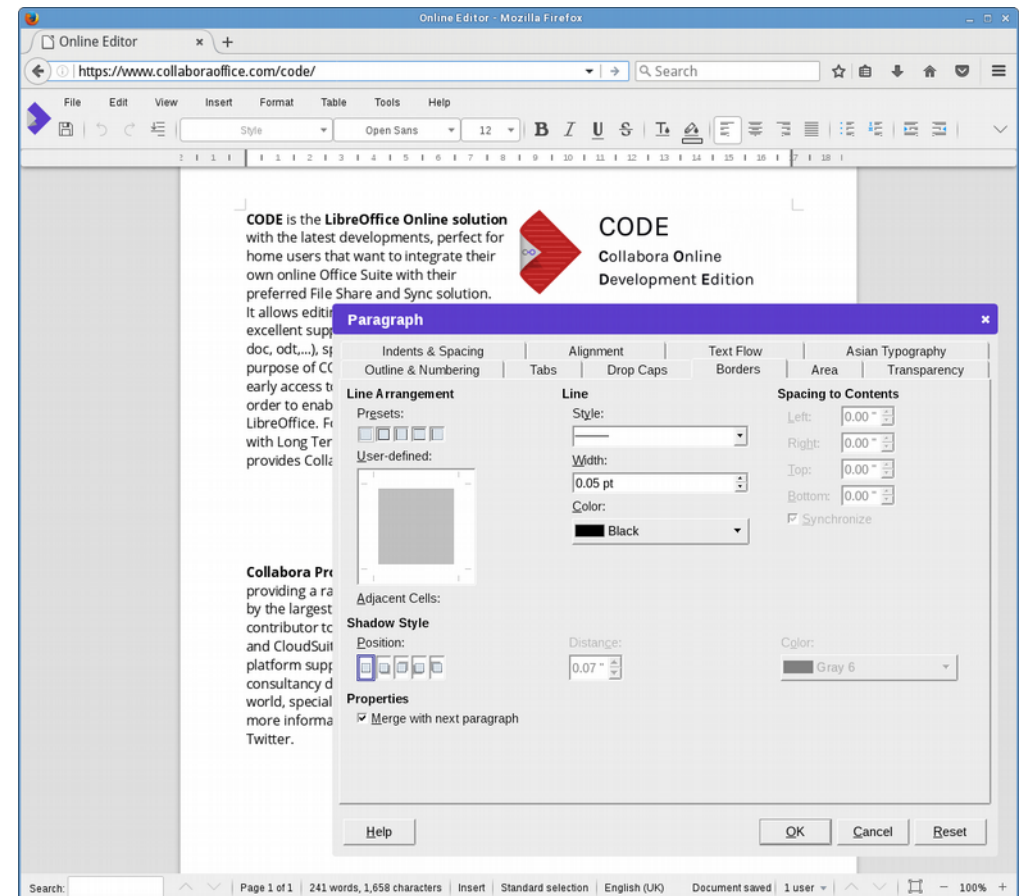
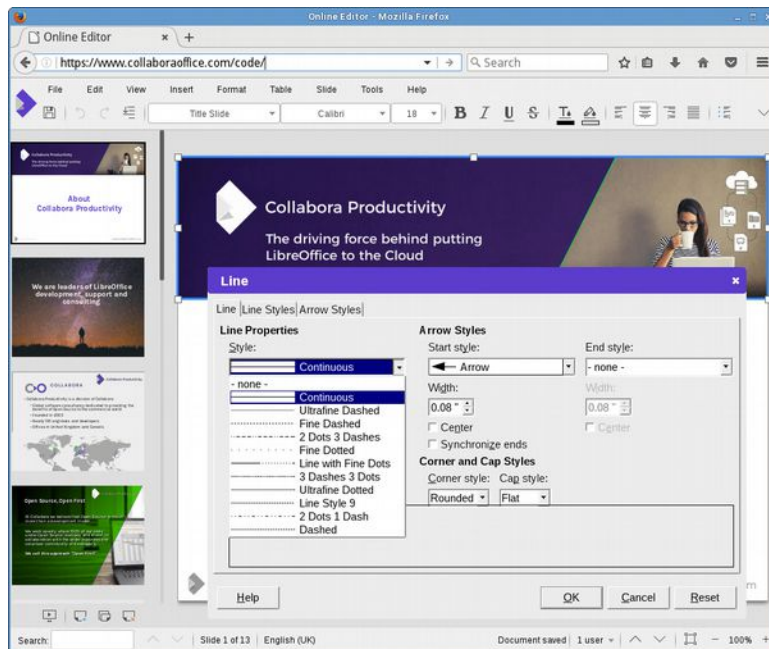
Dialog tunneling!

- Just reuse all the dialogs that are already there in LibreOffice
- The plan: Let the core render them, and pass them as bitmaps to Online
 - Nearly a year later: finally done ;-)
 - Most of the hard work done by Pranav Kant, big thanks!

LibreOffice master

The following features are now exposed

- Advanced character, paragraph and page properties
- Line, fill, cell properties, etc.
- All that collaboratively!



Technical Details

How Does it Work?

Nearly everything is done down in VCL

- Added various callbacks – dialog created, invalidate, etc.
- Reusing the dialog screenshotting feature for rendering the content
- Added a concept of LOKNotifier
 - Most of the LOK notification is done in sfx2 – but that is a higher layer
 - LOKNotifier is an interface that is instantiated in sfx2, but can be used in VCL – for the notifications about dialog creation, what was invalidated, where to paint
- LibreOfficeKit extended accordingly

LibreOfficeKit Extensions for Dialog Tunneling

Methods

- `void paintWindow(unsigned nWindowId, unsigned char* pBuffer, const int x, const int y, const int width, const int height)`
- `void postWindow(unsigned nWindowId, int nAction)`
 - General events, so far only closing the window
- `void postWindowKeyEvent(unsigned nWindowId, int nType, int nCharCode, int nKeyCode)`
- `void postWindowMouseEvent(unsigned nWindowId, int nType, int nX, int nY, int nCount, int nButtons, int nModifier)`
- `void postWindowMouseEvent(unsigned nWindowId, int nType, int nX, int nY, int nCount, int nButtons, int nModifier)`

Callbacks

- `LOK_CALLBACK_WINDOW`, with a JSON payload
 - Indicating actions like “created”, “title_changed”, “size_changed”, “invalidate”, “cursor_invalidate”, “cursor_visible” and “close”

Challenges: Language Support

One document can be co-edited by multiple users

- And each of them can have their UI in a different language
- LibreOffice used static objects for the text resources
- ~All the places had to be converted:
 - `static std::locale loc(Translate::Create("cui"));`
 - `return Translate::get(pKey, loc);`
 - + `return Translate::get(pKey, Translate::Create("cui"));`
- Similarly SfxModule had to be adapted to be able to switch language when the view switches to a different user

Challenges: Modal Dialogs

Non-modal dialogs are straight-forward

- But the modal ones call `Execute()` which blocks
 - Not that events would stop flowing – `Yield()` called inside `Execute()`
 - Meaning that 2 (or more) users can open the same dialog just fine from different views
 - The problem is when they are to be closed & the changes have to be applied
 - All the `Execute()`'s have to end first before the execution continues
 - Problem! - one of the users can go for lunch in the meantime...

Modal → Modal Async Execution

The solution is to convert the modal dialogs to async

- Still they stay modal, but do not block in Execute() any more
- LibreOffice already had StartExecuteModal which was working fine, but lead to big amount to changes
- Introduced a new StartExecuteAsync() with a lambda
 - ScopedVclPtr<SfxAbstractTabDialog> pDlg(pFact→CreateScAttrDlg(...));
 - + VclPtr<SfxAbstractTabDialog> pDlg(pFact→CreateScAttrDlg(...));
 - [...]
 - short nResult = pDlg→Execute();
 - + std::shared_ptr<SfxRequest> pRequest(new SfxRequest(rReq));
 - + pDlg->StartExecuteAsync([=](sal_Int32 nResult){
 - [... the code that was previously following after Execute ...]
 - + });

Usual Caveats

“I issued a dialog via .uno: command, but it does not appear in the Online”

- Most probably the dialog does not have a parent – uses nullptr
- Solution: Assign it a parent, ideally window of the view shell

“The dialog does not switch languages for users”

- Most probably there is static variable holding the locale
- Solution: Find it & de-static-ize

Anything else

- Happy to help on the dev mailing list or on the IRC!



Collabora Productivity

Thank You for Listening!

And the following people for working on this:

Pranav Kant (main author of the tunneling), Henry Castro, Michael Meeks

By Jan Holešovský

kendy@collabora.com @JHolesovsky +holesovsky skype: janholes