Viewer API (java, wpf and .net)

Meeting September '16

Decisions:

- No additional interface changes for now
- make viewerR2 default viewer for new customers
- implement other features when requested by customers
- Maybe show of a sample using viewer & printer, because viewer can no longer print

Future Roadmap (B priority)

- Annotations for WPF/Winforms + Documentation
- OCX Interface
- Annotation additional features
  - Other annotation types (stamp, line, squiggly etc.)
    - Inherent issues of freetextannotations not resolved yet
  - additional control over annotation editing (user restrictions etc.)
  - usability/design improvements
- Compare documents
- prematurely canceling rendering

Architecture

viewerarchitectureconcept.pdf

Feature List

List of Features, which might be interesting to integrate in future iterations

Feature	RE2	RE3	RE4	RE5	Postponed
Touch (wpf viewer only)	✓				
Load PageLayout and OpenAction from pdf	✓				
Save PageLayout in user registry and load it when starting viewer	✓				
Licensing	✓				
Outlines	✓				
Thumbnails	✓				
Text extraction from marked rect ~~(markierte rects loest event aus mit pdf coord -> EXPA)~~	✓				
TextSearch	✓				
Samples & Test automation (or semi-automation)	(✓)				
Update manual	x				
Dynamic Cache management (manage unloading stuff etc. slidingwindowcaching). Implement generic Cache		✓			
Dynamic Document Loading (Do not directly load all pages when opening a file, needs canvas calculation based on guesses)		✓			
Dynamic Textfragments (Dynamically load textfragments when they are needed)		✓			
Bitmap reusing (reuse last drawn bitmap and only render the parts that are missing)		✓			
Proper Error Feedback (Errormessages, that can be dispalyed in pop-ups)		✓			
Update manual		✓			
Annotations (read and write)			(✓)		
Compare documents			x		
Update manual			x		
Make CursorIcons overridable				x	
Mark rectangle	✓			(x)	
Highlight text	✓			(x)	
Text Selection		✓		(x)	
Style customisation (where applicable, e.g. WPF)				x	
Embedded documents and collections				x	
Update manual				x	
OCX interface					x
Blurry prerendering (Showing an approximation of the image before accurately rendering it					
Processing notification (Some form of letting the user know that the viewer is doing smth)					
~~Printing~~					
Extraction / selection of other pdf-objects (images etc.)					
Lanugage support					
Keyboard shortcuts (New ones, because the old ones are weird(or even rebindable ones))					
Extract document title from DocInfoDict and DisplayDocTitle from ViewerPrefDict					
Fullscreen mode					

Annotations

Maybe it's possible to not implement a custom annotations rendering thing and instead just save the annotation to file and rerender every time?

If not maybe its possible to have a method from rendering engine available, which only paints an annotation.

How do we handle the multitude of Annotations?

- Separate class for each annotation, with a abstract super class PdfAnnotation

    - Most implementation will actually be on level of PdfAnnotation, as most features are shared amongst all annotations
- Annotations will also share GUI's as there are few categories:
    - Draw rectangle with mouse
    - insert text
    - sequence of points
    - ...

Roadmap

1. Familiarize with Java viewer API R2
2. Get list of annotations per page from pdf
3. Handle link annotations to change cursor on HoverOver and GoToDestination when clicked
4. have a createAnnotMethod on API and be able to create an annotation (textannotation only?), that then gets shown.
    I. For starters just write the arguments manually (e.g. createAnnotMethod(TextAnnot, "Text", rectOnPage, [...])
    II. This makes also a addAnnotation method on API necessary
5. Implement other annotation types. Maybe do this after the next step or in parallel
6. create GUI in sample that allows to create annot interactively
    I. Prioritize Sticky notes and highlight annotation.
    II. Other annotations for later:
        a. Textannotation, stampannotation, Freetextannotation, linkannotation
        b. highlightannotation, circleannotation, squareannotation
        c. polygonannotation, polylineannotation, lineannotation
        d. widget annotations
        e. file attachment annotation, sound, movie, whatever
7. Allow customer to modify user rights for editing annotations based on arbitrary criteria (creator, date, user etc.)
    I. Probably done by some sort of event when trying to modify a annotation

Compare Documents

What options are there as a general approach?

- **Native level switching**:
    - On native interface add option for 2nd document name
    - use all metainfo from first document (pageRects etc.)
    - then draw both documents with same arguments of Draw(...)
    - Then we need special rendering stuff for differences (red n green dots)
    - GUI stuff (shortcuts to switch view and option to open 2 documents)
- API level switching:
    - have 2 native interfaces, one for drawing each document.
    - otherwise same as native level switching i think?
- Application level switching
    - Create a sample, which uses 2 API's, one for drawing each document.
    - each application does the navigation bit for its own document and we just compare the resulting bitmap in component.
    - There will be some issues with mouse events and such, because we have to manually forward them to both components, since they cannot be in the same space
        - Basically we need a new Pane, which forwards everything to both components and consolidates the resulting visuals.

Interface adaptation

When using the WPF interface, some issues with the API have surfaced. The interface should be adapted as such:

- asynchronous tasks instead of separate event handlers

Issues

Here are the issues that need to be solved for final release listed it doesnt

- Searching more than one words doesn't return a match (maybe problems with spaces?)
- Restrict moving annotations to page on all 4 sides.
- Disable navigation/layout/view buttons and menus when no document is opened
- ~~Check if recDepth infinity loop is also an issue in Java~~ (seems not)
- Zoom shouldn't change when scrolling/changing page (doesn't always happen)
- There are white border artifacts on certain pages when zooming in and out
- When on last page of a document → page down jumps to the top of the last page.
- ~~Remove initialization from structs in pdfviewerapi_c.h - does not exist for C!~~
- ~~GetOpenActionDestination exists both inside the java api and c api. Currently I see no nice way to use the c-api method from the java-api. The problem are the double pointers and the fact I can not identify them if they have been untouched in a reliable way (I think).~~ Found nice way.
- If in FIT_MODE having a very tall but narrow (few pixels) panel slows the viewer considerably down due to the fact that the number of pages shown increases based on the zoom factor. There can be easily over 100 pages shown in the panel and their rendering takes a lot of time.
- ~~JavaViewer thread crashes if panel size is < 0 (e.g. by resizing window)~~
- **Manual ~~barely exists~~ incomplete**
    - ~~structs, events and delegates dont exist in context. Find mapping rule java↔C#~~
- ~~Java and~~ cSharp textfragments arent freed properly in some cases
- Javaviewer: when opening a large document and scrolling around, quitting the program sometimes leaves a large heap behind that is not cleaned up.
- (I need a WPF sample that does not use dependencyproperties to test the INotifyPropertyChanged stuff)
    - Do i really **need** that? → no?
- (Make PageCount on Document behave the same as everything else (dont bypass queue and make cache in documentManager))

- (I think there's an error: if you start highlighting text (drawing rectangle), but meanwhile scroll the first page out of viewport, then text on that page will not be highlighted (because canvas takes firstpage/lastpageOnviewport as arguments))
- (performance of loading pages when scrolling to a page for the first time)
- (There is not a Cache that releases its objects without explicitely invalidatig the cache (Only alwaysRememberCache, which always remembers))
- ~~Javaviewer seems to have issues with loading pages of different size and adjusting positioning when scrolling there~~
  - I think thats solved?
- ~~WinForms Viewer opening of new document sometimes shows old one in a different zoom/fitmode for brief time before showing new document (See video)~~
  - send new version to fabian and check whether problem still exists
- ~~Winforms outlines clicking doesnt move viewport to destination~~
- ~~**Some methods on API's are public, but are only used internally by controller. These should not be publicly accessible**~~
  - I found only one such case? Well actually in java there are plenty
- ~~(I think we need minimal samples (minimal amount of code to get the api to show anything, with nothing fancy))~~
- ~~WPF viewer cant open password protected files~~
- ~~Test what happens if there is an error when opening file (when i had the licensing issues there was "Exception: null" messagebox, which is not helpfull)~~
- ~~Sample compilation fails if there are spaces in path (we need magic quotes or something for build events)~~
- ~~**Produce Kits** with bindings for tfs etc. resolved~~
- ~~**Java viewer opening new document shows parts of old document**~~
- ~~Searching for empty string crashed (NullPointerException)~~
- ~~OpenMem isnt functional yet~~
- ~~**split up PdfPoint**~~
- ~~**Terminating viewer mid-execution often (always?) crashes it in winforms**~~
- ~~**Terminating viewer mid-execution often (always?) crashes it in java**~~
- ~~**In WPF viewer when in SinglePage mode textselection only seems to work on some pages**~~
- ~~javaviewer seems to try and load pages from source although they should already be loaded? (maybe?)~~ - no
- ~~WPF Viewer Dependency properties are static and cant trigger non-static events (well you could do a hack...)~~
  - You can just have a private static reference to the "one" instance and call the event on that instance if it is not null
    - Could there ever be not just one instance though? yes! and then there would be chaos. There would be chaos anyway though because dependencyProperties are static
  - I need dependencyproperties for backwards compatibility though!
    - **I should have a parallel implementation with normal properties & INotifyPropertyChanged à la WinForms.**
- ~~In wpf viewer on startup singlepage mode is used but oneColumn is shown in toolbar~~
- ~~If i load huge.pdf then the content appears, disappears for a few seconds and reappears again?~~

Exception throwing issue

Numerous Dependencyproperties etc. will occasionally throw PdfNoFileOpenedException. This has to stop.

The exceptions will still have to occur, but i need to catch them. However on which level should i catch em?

In controller? Or one level above controller?

- controller:
  - controller methods call other controller methods though. So some exceptions will be cought by lower call and not recognized by higher one
    - However: how often do i actually need to know that there has an exception occured. can't i just ignore it and move on?
- above controller:
  - here we can be sure which calls actually come from customer

- But in some cases, wouldn't i rather have an exception instead of a nonsense value (e.g. null), to avoid all the null checks?
  - **Policy: avoid exceptions for properties, but not for other methods**
    - This directly implies, that the exception handling has to be done above controller, since controller doesnt know what is a property and what not.

The open close multithreading problem

If one Opens a file and closes it before opening has completed we get unhandled exceptions from the PdfDocument, because the handling of e.g. onOpenCompleted expects an open file, but it isnt.

I need better handling of files not being open and i need to centralize its handling.

There is an isOpen in PdfDocument AND ind PdfCanvas. The one on pdfCanvas level is probably not accurate and should be removed/replaced

Another issue: closecompleted event should only come when there was an actual close request. especially there should be no event when opening

The exception issue

Currently exceptions such as PdfViewerNoFileOpenedException occur in regular use (at start when no file is open yet.

We have to know at upper layers whether a file is (or better: will be) open or not.

Just simple boolean in controller, setting true when open, false when close?

There will of course be issues if a open failed... Also some commands have higher priority in queue, so the file might not be open anymore and such

Basically we say in regular use (opening succeds) no exceptions will occur and we can do all of the stuff as we want to. When opening fails, then exceptions will happen, because subsequent calls after open will have gone through, even thought the opening was ultimately not successful

However, these exceptions should only happen for stuff the customer does actively on the interface, and not for stuff we do automagically: i.e. scrolling and such should only be enabled when there is truely a document open, however NextPage() should always try to go to next page and throw exception when failed.

There are mixed cases though: e.g. rotation: we want the user to be able to say "OpenFile(file.pdf);rotation=90", but rotation also implies a setDestination command, which will fail, if no document is opened. This means, i need a second bool, which indicates that a file *should* be open.

The page loading issue

Issue: in dcve-client when scrolling through thumbnails, gui freezes.

Cause: SinglePagemode recalculates the canvas every single time we change the page. This necessitates loading the pagerectangle, which must wait for completion of a pending draw request (which might be somewhat time consuming).

Measures: The predictionalgorithm, which causes adjacent pages to be loaded, necessitates that the gui thread waits even for these other adjacent pages to load, effectively increasing wait time. We should only wait for the requested page and load adjacent ones in separate requests.

ListenerLists

Why do they even exist? why not events? Is it cause of static dependency properties? Do the events even have to be static? i mean just bc the props are static doesnt mean the events have to be, does it?

Its bc. the dependencyproperty has a onPropertyChanged delegate and that one has to be static and trigger the event (which he can only do if its static)

Or i can just do the INotifyPropertyChanged thing as in winforms? its kinda verbose though... and old

Text selection

What do we need to get online text selection running (assuming that the order of content in pdf stream is correct)?

Main problem: map from location on screen (or page) to textfragment or to index in onPage text.

- What are the rules to which textfragment a location gets mapped to?
  - If location is within textfragment, map it to said textfragment (inclusive)
  - If location is not within textfragments, map it to the closest one (exclusive?)
  - that is not all that simple thoug, because we dont want to check all textfragments
- ~~Maybe i would have to save the textfragments 2D instead of 1D~~
  - ~~Dict<double, Dict<double, TextFragment> >, with the center point (x,y) and thus dict[x][y]=textFragment. We should group together y-offsets that are "close", which means that their y-spans intersect~~
  - ~~now we can directly find the two lines that the location is in between and then two candidates for exact textfragment~~
- We should make some kind of binary search on the textfragments
  - from the ~~above~~ below pseudo code we can learn what we need to know to make the algorithm work:
    - isAbove/isRight etc. methods (ez)
    - haystack representation (list<PdfTextFragment>)
    - haystack.indexOf, haystack.subrange and haystack.middle function (ez)
    - needle.line.end and needle.line.start function (that requires the textFrags to be stored in 2d dict. Actually we dont need a dict, just lists List<List<PdfTextFragment» where each inner list is a line)
      - Still this would'nt be a fast access, since i would need to check every line whether its the line with needle in it.
      - **Actually the only thing I really need from line is first and last, i don't even need the 2D structure, if we e.g. would just save lineFirst/lineLast as direct references in each fragment**
    - get the closer of 2 textfragments to a point (ez)
  - So what we primarily need to do is set the PdfTextFragment.FirstOnLine and PdfTextFragment.LastOnLine correctly when loading.
  - As we do this online and quite often it may be wise to use the needle from last time as hint and start with said needle instead of haystack.middle

```
location  //the location we try to match
haystack = all text fragments
while(true)
{
  //iteration termination when haystack trivial
  if(haystack.size <= 2)
  {
    if(haystack.size < 2) throw asdfgjsjdg;

    return element in haystack that is closer to location
  }

  needle  = haystack.middle //the fragment we currently look at
  if (location isWithin needle) //maybe reorder that to be at bottom
    return needle;
  else if(location isAbove needle)
  {
    if(location isRightOf needle)
      haystack = haystack.subRange(0, hastack.indexOf(needle.line.end));
    else
      haystack = haystack.subRange(0, needle);
  }
  else if(location isBelow needle)
  {
    if(location isLeftOf needle)
      haystack = haystack.subRange(hastack.indexOf(needle.line.start), haystack.end);
    else
      haystack = haystack.subRange(needle, hayStack.end);
  }
  //all elses beyond this point know, that the location is on the same line as needle
  else if(location isLeftOf needle)
    haystack = haystack.subRange(max(0, haystack.indexOf(needle.line.first) ), needle);
  else //if(location isRightOf needle)
    haystack = haystack.subRange(needle, min(hastack.end, haystack.indexOf(needle.line.last) );

  if(oldhaystackSize == newHaystacksize)
    break; // we can't reduce the relevant frags anymore
}
```

- At the end of the algorithm, if we use the break func, I claim, that:
  - **Only the first two and the last two elements in haystack are possibly the closest neighbour**
  - Actually there exist some special cases, where this is wrong... but i think they wont matter and that in those cases I would actually not want the geographically closest one anyway.
  - Actually thats completely wrong, there exist real cases where you dont want that result
  - For now i had to enable searching all elements in the remaining haystack for distance, which kinda sucks...

- maybe i should switch to something that just first searches the correct lines and then continues with the rest?
- Also one might still consider saving the frags sorted smartly (linewise?)

What i have implemented now:

- Algorithm that works under some limitations:
  - Textfragment order in pdf stream is assumed to be in order with y-coordinates (i.e. if B is after A in pdf stream, we assume it is below or on same height as A)
    - This assumption is already broken by a simple multi-column text
  - ~~Pages are unrotated~~
  - Text flows left to right.

What I have implemented eventually:

- trivial solution, which just goes through all textfragments and minimizes distance
  - it performs surprisingly well

Text search / extraction assumptions

Short list of assumptions that we make about the pdf-file for our implementation of text search and text extraction:

- Words are saved in the file in correct order (read-direction)
- Consecutive text fragments are read as having at least one space character in between them (i.e. no word can be split over multiple text fragments)
- Consecutive space characters are read as a single space character (both in pdf-text and in input-search-string)

Small things todo one day

- **wpfviewer outlines are not clickable**
- **WinForms viewer outlines clicks dont GoTo anywhere**
- **implement interfaces for top-level api, one for customer, one for controller**
- find out why canceling doesnt work (alt f4 or esc)
- Make 2 interfaces for PdfDocument, one for non-rendering stuff, that the manager forwards a reference to and another for the internal stuff
- Remove Weird selectionRectangle in WPF viewer (entire viewpane has a dotted rectangle around it, has to do something with focusable, but we need that for key navigation) - Just occured, when tabbing back to it during text selection
- Save more stuff to registry
  - Rotation
  - Fitmode
- Rework PdfExceptions to be useful
- Change pageRanges to be sets to avoid double pageloads and make stuff more efficient
- check every object that can be obtained on api via reference, that changing it wont break anything within api (whenever possible return clones).
- bitmap lags behind highlightrectangles (we only render the highlightrectangles to the new positions, when the bitmap is ready)
- OpenMem not implemented
- set/get destination from api (get could be nice for some people, who want to return to the exact location later)
- test non-standard resolutions and mid-execution resolution changes
- eliminate race-conditions for all events (a simple null check is not good enough!) Link [http://www.codeproject.com/Articles/61878/How-to-Safely-Trigger-Events-the-Easy-Way]
- Make better implementation of request priorities (should be centralised, so we have an overview who has witch priority maybe?)
- Check winforms performance issues of 32 bit debugging version with debugger attached (very long startup time and stuttering while scrolling)
- free textfragments correctly in java viewer
- Winformsviewer: the first time you click on zoomin/zoomout button, nothing happens
- Use a generic as bitmap and parametrize depending on WPF/Winforms
  - We have to parametrize this like everywhere and its really annoying. Alternatives?
    - I could handle it as a IGenericBitmap everywhere where it doesnt matter
  - Problem: there are some weird errors in the finished implementation, that i can't debug ⇒ moved to a shelveset
- ~~clean up memory leaks in native interfaces~~
- ~~**Fix the whole CSharpAPI vs CSharAPI thing**~~
- ~~**Fix issues of javaviewer when changing pagelayoutmode to singlepage (too large canvas)**~~
- ~~release resources properly in native api~~
- ~~Some outlines are missing (all viewers) or go to wrong destination (PDFreference15_v6 chapters 5.7.2 and 5.9)[cannot reproduce? seems to work?!]~~
- ~~rework javaviewer sample pagelayoutmode selection to be the same as other viewer samples~~
- ~~**Rework/unify RequestQueues (c# and java implementations are different ATM!):** * Define what should happen to pending drawrequests if close and open request get enqueued/executed * either open/close flush the entire queue * or just execute everything in order. it is supposed to be a priority queue though... * deal with missing merge implementations (define whether merging should even exist)~~
- ~~PdfRects for javaViewer?~~ - No!
- ~~Replace textfragment 270deg rotation, that is now a 180 and 90 rotation.~~
- ~~**WinForms viewer lincensing not done yet (getLicenseIsValid not implemented)**~~
- ~~Make thubnailview disableable, due to its high performance impact on large files~~

Bitmap reusing

Whenever we have to draw the viewport, we take the last rendered image of the viewport and only render the missing part

Remember that I already almost implemented that: TFS Link [http://tfs.pdf-tools.com:8080/tfs/Development/PDF/_versionControl/changeset/935#path=%24%2FPDF%2Flibpdf%2FPdfViewerAPI_R2%2Fjava%2Fcom%2Fpdftools%2Fpdfviewer%2FPdfPag

Where do we put that implementation?

- GUI thread (before enqueueing)
  - - We cant predict which bitmap will be rendered, as there might be multiple request enqueued and we dont know which one gets executed and which ones get removed
  - - We cant use the last rendered bitmap, because that is only returned after we enqueued this command.

    - -
- **native thread (in drawRequest.execute or even directly in native part)**
    - - More work on native thread, that is not actually native
    - + We can Cache the last bitmap on this level and reuse the bitmap, that we just finished rendering

Current implementation has tearing issues, due to rounding errors:

- I think we should do the reusableRect calculation in target coordinates, to avoid rounding issues and then transform the rects back to source
    - Actually you cant do that calc in target coords. you need to match content, which only works in source.
    - So i would have to calc in source, transform to target (implicitely rounding) and then transform back to source, to avoid tearing.
        - Can we always avoid tearing?
            - Actually no, Imagine you paint a page once and then move it 10.5 pixels to the left. It is impossible to fill in the old page correctly, since it is differently adjusted to pixels and you cant paint half pixels, so you naturally get half a pixel of tearing in between the reused part and the newly painted part
                - This implies that we should actually **restrict the ways to move the viewport, to be only in entire pixels**.
                    - however that might be silly, and cause stuttering when scrolling? Actually upon closer examination of adobe reader, it seems to do exactly that.
                    - Well actually our viewer also already does that anyway
                - By introducing this pixel-scrolling, do we automatically resolve the rounding problems?
                    - Well obviously not, since we still have the issues.
                    - When zooming we can still get weird in between values. Then adding a integer (scrolling), can still result in another in between value.
        - We should be able to always avoid tearing, if we adjust the rectangles that come from subtracting the reusable one from the targetRect.

Sometimes reusableTargetRectOnLastBitmap and reusableTarget dont have the same dimensions. Why?

- They are both transformations of the same sourceRect, but they use a different set of source and target to calculate their positioning. However source and targets of both sets are supposed to have the same zoomFactor
    - But basically the error has to happen while zooming? try precalculating all zoomFactors and giving them as argument, instead of calcing them on the fly

There are still heavy tearing issues...

- I think i should just forward a viewport clone to the drawing method. We can then get the correct zoomfactor and transform all of our stuff exactly from source to target and vice versa
    - we have to forward the entire viewport, to also get a notion of offsets, which are used to calculate target rectangles.
    - We still have to keep the normal target/source rect calculation in the viewerController, as we need to get the pageRect information from cache, which is located in DocumentManager

Even if i dont reuse bitmap, but instead draw that rectangle newly also, i get a gray line when scrolling upwards. Why?

- Because `(int)( (double)intvalue * zoom / zoom) == intvalue` is not necesarrily true
- I think `(int)Math.Round( (double)intvalue * zoom / zoom) == intvalue` is true though. - no it isnt!

We still have tearing. It appears that always, whena page comes newly within the viewport (not necessarily for the first time), the height of the targetrectangle is 1 pixel higher than of all following slices. Well actually it looks more like the targetrect is accurate, but the sourceRect is too big

- the problem stil seems to be in rounding stuff. Basically we can never get rid of that, because there cannot be a bijective transformation from int to double and back. But we should be able to make it damn unlikely
    - Maybe we can profit from the limited range of zoomfactors?
    - I have to isolate the issue exactly!
    - Essentially thats the problem : `lastTargetRects[lastIndex].containsInt(calcTransformedTargetRect(lastSourceRects[lastIndex], lastTargetRects[lastIndex], lastSourceRects[lastIndex], sourcePageHeight, zoom) )` is not necessarily true
        - that means that calcTransformedTargetRect isnt correct. which is hardly surprising. The issue is, that in viewerController the transformation is done, when in canvas coordinates, before offsetting the coordinates to get page and viewport coordinates respectively.
        - Can we fix that? Maybe use the same transformation or something?
            - For that I would need the pagerect and the viewportrect
            - Actually, i could just change the structure (once more) and instead of calcing source/targetrects in controller, hand down pagrect and viewportrect copies and only in document calc the source/targetrects, which would then be consistent with the ones from last time
                - - More stuff to do in Worker
                - + needs less memory per unexecuted drawcall, since we dont need source/targetrects, but only pagerects + 1 viewport rect copies
                - + consistency (no tearing! which i kinda dont know how else to achieve)

Turns out there is an additional underlying thinking error:

- the sourceRectangle's are not consistent over multiple draw commands. e.g:
    - Changing pagelayoutmode moves the sourcerectangle around
    - dynamically loading pages before the one you look at, move the one you look at on the canvas.
- This means we cant just calc reusablesSourceRect on the old bitmap.
- Instead we have to calculate the rectangleOnPage and then get the sourceRect on the old canvas with that

There is a chaos of the different lists:

- Lists for drawing: sourcRects/targetRects/pages these lists are parametrizing all draw commands and contain each page multiple times
- Lists for reusing next time: visiblePages/visibleRects/pages these lists are used as references in the next drawRequest to reuse the old bitmap. Each page should only be contained once
- The chaos has been resolved

the problem is, that sometimes the rectOnPage is not entirely within the viewport and it thus gets cropped a bit by one of the viewports, yielding inconsistent size. The larger problem is that reusableTargetRect and reusableTargetRectOnLastBitmap have sometimes different sizes, They get generated by

call `GetTargetRect(reusableRectOnPage, pageRect, viewport)` whith viewport = lastViewport for the OnLastBitmap one.

- the base question is why is the reusableRectOnpage not entirely within both viewports (its because rounding errors)
- And what can we do, that it is in future?
  - `PdfSourceRect reusableRectOnPage = lastVisibleRectOnPages[lastIndex].intersectDouble(visibleRectOnPage);` , so the problem is, that one of those is not within its viewport
  - Which in turn is of course, because it has been intersected with the viewport at its natural location instead of at the origin (yielding different rounding behaviour)
    - but even if i move both, the viewport and the rect near the origin we can still get rounding errors due to their different size.

IMHO these problems are farther reaching than just these isolated cases, i need an overall concept of how to deal with the conversion inconsistency. Somethin along the lines of calcing everything in one coordinate system would be ideal

- can i calc everything in one coordsystem?
  - RectOnPage coords
    - i had that before, but then had issues, that transforming the differences (rectsToDrawOnPage) would yield results that do not connect to the reusedTargetRect, leaving space for gray lines
    - i could fix that maybe by doing some weird special case handling and afterwards adjust transformed results maybe?
    - But already the transformation to rectOnPage coordinates is iffy:
      - `pageRect.intersectDouble(viewport.Rectangle.GetSourceRect(viewport.ZoomFactor) ).Offset(-pageRect.dX, -pageRect.dY)`
      - depending on the offset of the viewport and the offset of the pageRect, the resulting rect will be inconsistent in size (i think, lets proove that...) - indeed its proven that that yields inconsistent results.
      - that might not matter though...
    - We have given the new pagecoordinates of what we want to draw and a suitable transformation function, resulting in the targetRect on viewport. We have also given the page and targetcoordinates of the last draw request as well as the resulting bitmap.
      - We need to intersect the pagerects to get the reusable area and then map the corresponding targetRects on previous and present request to one another to copy bitmap.
      - We need then to subtract the reusable area from new targetRect. the resulting differences are the rectangles to be drawn (of which we also need the sourceRects!) I can do this in OnPage coords though! (this is the point where i get the problem that the transformed differences together with the transformed reusableRect dont equal the targetRect, yielding gray lines)
  - Canvas Coords
    - For reusing old bitmap i would have to map old targetCords to new canvasCoords (and possibly the page is at another location in canvas now!)
  - Viewport Coordinates
    - I have to calculate the reusableRect in sourceCoordinates (because the information of which image part is the same is only available in sourcecoordinates!), so how do i get to targetcoords?
    - OK, so we have that implemented now and it seems to be fairly accurate (still very slight tearing issues, but almost exclusively at page borders)
    - 2 issues remaining (or **more**):
      - reusablebitmap on this and last bitmap is not always the same size and then the size of source and target destination of copy operation is not the same, yielding errors→ we have to take the smaller one or something
        - I can't just take the smaller one, because i dunno how to position it then (was the line of pixels i just removed left or right of rectangle?)
        - Why does this happen in the first place? - because What does it mean if this happens?
      - The first part of a page that gets shown is still slightly different rendered then all the rest that gets appended afterwards
      - there are some gray lines sometimes (how can that even happen, if i subtract in viewport coordinates???) → find out why
        - We calc the visibleRectOnPage by intersection of viewport and page, transforming to source and intersecting with the same of the last bitmap. However when we then transform this reusableRectOnPage back to targetCoords, the result is sometimes slightly too small
        - Solution: I just took the entire pageRect for calculation of the differences instead of the visibleRectOnpage, as it is later cropped with the viewport anyway (When getting s/t rects)

And now the entire thing again for java... - Done

There are 2 more things to consider:

- We could also reuse the old bitmap if the rotation changed, by transforming the buffer
- With a similar approach we could forward the unfinished bitmap early to the viewerPanel (before the last rectangles are drawn)
- With a similar approach we could guess the look of a part of the viewport when the zoomfactor changes and forward that to the viewerpane as well

The documentIsOpenFlag

Where do we actually need it and where should it be delegated to children?

- PdfDocument should have it.
- PdfCanvas and above should not have it
- Should documentmanager have it?
  -

Make bitmap in c#viewer generic

Now we use WPF bitmaps and then later copy the backbuffer over to a winforms bitmap for winforms viewer, which might be somewhat inefficient

We should parametrize the type of bitmap as a generic on c#Viewer and instantiate accordingly for winforms/wpf

Dynamic caching - Generic implementation

We want an implementation of a dynamic caching component, used for multiple things:

- document pages

- textfragments
- search matches
- thumbnail images
-

Basically make an interface and multiple implementations:

- dummy implementation: actually doesnt cache anything, but just goes to get the data from source every time
- AlwaysRemember implementation: dynamically loads, but never forgets, unless explicitely told to forget
- LRU implementation: caches things
- maybe even more implementations

This way we can restructure our architecture using the dummy until everything works and then tackle caching-issues separately, when switching to the other implementation

Things to consider about the implementation:

- Sliding window caching might be useful (preload some data, which is "close" to data we load anyway)
- explicit invalidation of cached values is necessary (e.g. opening new document, searching for different word)
- Cache should maybe allow for custom prediction algorithm, that tries to predictively load sources

Generic implementation - interface:

- We need some generic form to pass the loadArguments (object array?)
- we need to be able to set a delegate for the method that the cache should call to load things
- Prediction algorithm should also be set using delegate maybe?
- For some objects we need a "GuessObject", which represents a guess of the actual object. E.g. PageRectangles should spit out a guess on their size without loading it from source

Just try out something...

- The object array as dictionary key is kinda shitty
  - Do i ever even need that? wont int keys suffice?
    - pages→ int
    - search matches → two ints?
    - textfragments → two ints?
  - Cant i just use a bloody generic? (yes!)

How to restructure for pageloading using genericCache:

- remove cache in Document.class
- Cache must be inserted before transitioning to native thread
  - In DocumentManager ~~or Canvas~~
- Tie up loose ends down in the document
  - How to do the changing of zoomFactor?
  - How to do the changing of rotation?
  - pagerectangles will come unrotated and unzoomed from document.getFromSource
  - idea: cache them unrotated/unscaled and give out clones, that get rotated/scaled when getting
    - that idea is dumb, because canvas needs to change the rectangles to adjust positioning
  - idea: have a OnFirstLoad delegate to execute stuff whenever something is first loaded
    - But then we also need ChangeAllLoaded delegate to change rotation/zoomFactor
    - If possible these two should be within one method, which both, sets the modifications of new loaded pages and changes all already loaded ones

If i load multiple cache entries at once, should it:

- return as IList
- return as Dictionary

Dynamic Document loading

We want to calculate the canvas without loading all pages, even in scrolling layoutmodes

Integrate a feature into caching or separate?

- **Integration: new function "getGuess(arguments)", guessing function can be given via delegate**
  -
- separate: canvas gets all available pages and fills in blanks itsself
  - that could get kinda complicated

How to calculate canvas in a world of guesses? Register canvas to listen for visiblePageRange changes. if changed, ensure that all visible pages are loaded from source (not guessed!) ant then recalc canvas How do we start (when no pages are loaded)? Just load 1 page (FirstPageOnViewport) which will cause the event etc.

The adaption problem (when our guesses were wrong and we scroll backwards, the viewport jumps around, because canvas size changes

- If we scroll backwards, we should jump as much as the canvas changes to counteract
- if we scroll forwards, do nothing
- How do we recognise that though?
  - I can recognice the size change by givin both the old and the new canvas via the canvasChangedEvent (EZ)
  - I can also pass the causing page as an argument and then by analyzing visiblePageRange deduct if its above or below viewport, to act accordingly
    - But: can i recognise the causing page? → yes
    - But prediction algorithm will sometimes load things that arent required and mess up everything

We seem to have memory leak issues:

- Debug Diagnostic tool seems to be jibberish at best
- visualstudio thing claims that Refresh() has many ressource allocations, but removing the refresh calls still crashes the program. I conclude, that the error is not in the managed part
- actually, it was just when copying bitmaps in viewerPanel

There are issues with calculateCanvasRect:

- Somehow, sometimes reducing the number of visible pages can change the canvas size (which it shouldn't)
- ATM complexity scales O(#pages) and it gets called a lot (at every visiblePageChange)
  - I cant work with one does-all calc-method anymore
  - we need specific updates for specific events:
    - visiblePageRangeChanged → correct positioning if necessary
    - rotationchanged, pageLayoutModeChanged, Open, SetPageNoInNonScrollingMode, bordersizeChanged → recalc everything
  - ok, so actually we need just one specific method for visiblepagerangechanged and can keep the does-all solution for the rest
    - Can i keep the implementation lazy? (with dirty bit) Well i would have to make it kinda half-lazy (lazy in every case except visiblePageRangeChanged, which precalcs the canvas based on changed)
      - you could make it completely lazy, if you would implement a mechanism to store the visiblePageRange updates (well actually just the last one). **actually that should be EZ**
      - basically thats already done, i just need a new bool, which tells me that only the pageRects are dirty and that no other events happened.
      - if only the pageRects are dirty, i can then do a simplified calculateCanvas

also, i think i know why the thumbnails are wrong when document has different rotations:

- getpagerect in canvas only returns guess, which is unrotated, as long as the pages are not visible on viewport
- t'is true! Also GoToDestination(PdfRect) should probably get the exact rectangle
- Solutions?
  - We need new methods on canvas that allow exact rects to be getted
  - Just get the rectangle from source, calcCanvas and then return the rectangle (get before calccanvas! otherwise its position might be wrong!)
    - Actually thats bloody stupid and just wrong. The calcCanvas needs to know which pages are new so he can check for adjustments to the rest of the pages / canvas
  - give a list to canvas of rects that need to be exact
    - calcCanvas with dirtyRects gets more complicated...

We still have this shitty stuttering b/c of pageloading/canvas calcing:

- best implementation:
  - the cache will load pages in background (NOT when waiting for a page) Actually, im not that sure if thats important... because we cant draw and load pages at the same time anyway, who cares if the GUI-thread is blocked for a bit?
  - the canvas can deal with arbitrary pages being loaded for the first time any time
    - I could whenever one is loaded add it to the list of rects that need to be getted exactly. BUT thats dumb, because then they would in turn cause their neighbours to be loaded and chainreact to load everything
    - **have some event from the cache, which informs caller of load. caller must listen to event and in case of canvas assemble a list of newly loaded rects, to know what to update.**
    - always return a boolean with newlyloaded (as it now is only for getGuess). BUT thats dumb if there is more than 1 caller that uses the cached items

Recurring problem:

- After FitViewport, we call OnVisiblePageRangeChanged
- this causes new thumbnails to load
- this causes canvas resizing (and scrollablerect resizing)
- this causes viewport to not fit anymore (and trips up the scrollbar trackranges)

How to solve?

- maybe the thumbnails should't be loaded isntantly anymore. That introduces latency anyway
  - queue new task on dispatcher to load them later?
  - I can pair that with using the cache for thumbnails

When the cache loads pages due to predictionAlgorithm, it can load pages that have already been passed by CalculateCanvas, that will only be considered in the next call to CalculateCanvas (which is bad and leads to scrollbarTrack problems)

- alter CalculateCanvas to be able to jump back if new pages were loaded during calculation.

The zoomfactor thing

Right now each PdfRect has a local zoomFactor variable for target↔source coordinate transformation

However: is this even necessary, or could we just have 1 global zoomFactor (the one in viewport), that is always used for transformation?

Which rectangles actually need to not use this global zoomFactor?

- source/target rects for drawing: These need to have the zoomFactor from the time of creation of the request, changes to the zoomFactor after that (but before execution of the request) should be ignored

For rotation we can just do the same thing? i think? Have to pay attention though: rotating multiple times will yield wrong result (in contrast to setting the zoomFactor multiple times) Thus we need explicit Location in code, where to set these things. but where?

- right when loading from source (but we don't really have the zoomFactor or the rotation available there?)
- At first use (that's kinda just a shitty & complicated version of the one above)
- Whenever we use it, we clone and rotate/zoom the clone.
  - That doesnt work, because the canvas has to modify the pageRectangles, that are written to cache
- **Solution:** We set a "parametrization" for the cache, which is something that needs to be applied to every element in the cache (e.g. zoomFactor and rotation for pageRectangles). Application works with 2 delegates:
  - FirstLoadParametrizer: will apply the parameters to a newly loaded element.

- ParametrizationChanger: will update all elements that are already in cache, whenever the parameters change.

Semi-automatic tests

The idea is to build an assisting tool/extension/whatever that will do all actions on the viewer systematically and the user has to verify that what he sees is ok.

randomness vs. pre-scripted tests:

- randomness will find things that i would never have thought of. ideal for testing new developments and final-testing. discovers random interactions.
- pre-scripted is very handy for developing / bug-fixing, as you want to do the same thing over and over again with every little code-change.
- i prob even need both, both things seem very helpful. We can combine them to reduce effort, by defining actions (see below) and using them both for random, but also for deterministic testing

Randomness idea:

- have a library of actions that the tool can execute and execute them randomly
  - The action has to be represented somehow, such that the user can see what is done, this can be achieved via log or visually highlighting things (rectangles n stuff)

implementation as unit tests idea (for systematic tests, not for random ones)

- unit tests already incorporate behavior to not execute a test that already succeeded again.

The integration problem:

- If possible we would want to use the existing sample and "click" the buttons n stuff for each action, which would not only test the API, but also the sample]
  - If possible we would like to not integrate this code into the sample, but have it separate, while still having access to internal members of the sample (the buttons etc.)
    - That is possible! we just implement a new viewer class that is an extension of the standard sample. I just have to make all the stuff i need from the sample protected and we're fine.
    - What i actually did was write a wrapper and just expose everything of the old sample, so i can access it.

UserActions: (A UserAction is an action that on the normal viewer, a user could perform and can be executed in any order)

- ZoomingActions
  - zoomin/out
  - zoomToRect
- GotoAction
  - Go to start/previous/next/last
- ChangePageDisplayModeAction
  - set mode
- OpenAction
  - open new file
- MarkTextAction
  - mark text in random rectangle on screen
- extractTextAction
- TextSearchAction
- View Outlines/Thumbnails Action
  - Maybe subactions to scroll around or select something
- ResizeWindowAction
- ScrollAction (3 kinds)

UserActionImplementation:

- as classes
- has to be adaptable to varying window locations/sizes
- i need to be able to execute a user action with specific arguments for deterministic testing

2Do:

- ~~Mark mouselocation~~
- ~~log~~
- ~~pause feature~~
- ~~save and load registry~~
- ~~repair ScrollWheelAction~~

Discovered bugs:

- ~~doing something that requires calculateCanvas and then opening different file invalidates textfragments used within calculateCanvas before finishing calculation. Maybe it was also the pageissue, that we tried to get pages that didnt exist?~~
- ~~old highlightedRects remain visible when changing page in singlepage mode~~
- Textfragmentsdict in canvas is sometimes null ???
- ~~rotating or selecting thumbnailelements early will mess up layout of thumbnails~~
  - ~~This happens because thumbnail dimensions and images are loaded dynamically by backgroundworker and we dont check for that appropriately, but that only happens because I did it wrong~~
- Selecting thumbnail as first thing doesnt work (jumps to wrong page)
  - This is probably, because loading of thumbnails is not complete yet
  - however this is a somewhat unrealistic case, since when scrolling manually there the thumbnails get loaded. only the scrollintoview doesnt work. (So i'll prob. not fix that, as we later have to change dynamic caching of thumbnails anyway)
- There are multiple problems with dynamically loading stuff (see above, outlines is the same, only that there it's even recursive!)
  - We need some standard mechanic, that allows access to dynamically loaded

Textsearch previous: howto:

- match all and prepend pages
  - indexes and offsets will change constantly
  - maybe overall use matchAll and make a list of all matches within pages
    - but with each page we prepend all indexes of searchresults must be updated
    - maybe set indexes as in page indexes?
      - that causes problems with matches that start and end on different pages
    - matchall will have to be recalled with each additional page we add and then search the concatenation of all added pages
      - Maybe only search pages close to newly added page (+/- one or based on searchstring length)
- How to save found matches:
  - index within page

Open Design Questions

- How should NextPage() in 2pages mode work? (special case, when the two pages do not have space on the viewport)
  - Status Quo: the viewport aligns to the left page and jumps to the next left page
  - Adobe Reader: jumps to the next not fully visible page, which might be the page to the right or the next left page
- Which scrolling functions should change the page in PageMode?
  - Status Quo: All scroll functions (middle mouse click, scrollwheel, mouseDrag)
  - Adobe Reader: only scrollwheel can change pages, other scrolls can only scroll within the visible page

"Real" page layout mode (insights)

How things were before: Regardless of active layoutmode (page or document), we load all pages and calculate the documentcanvas. The canvas for pagemode is merely emulated.

Thoughts: Make some optimisations for when a document is only viewed in pagemode:

- do not calculate the documentcanvas
  - Actually this calculation is incredibly cheap, even for very large documents. Not calculating the documentcanvas is barely beneficial.
- do not load all pages
  - Pages loading is actually very costly
  - Page loading operations have to be piped through the request queue and be executed by the DocumentManagerWorker
  - loading pages only when we need them causes waiting times on the dispatcher thread (which would cause the GUI to freeze)
    - Keep loading pages as low priority requests (load pages, whenever worker is idle)
    - Heuristically predict pages that are needed in near future (window of pages somewhat bigger than the range of pages on the viewport)
    - Combination of the two above

Status quo: We still calculate the documentCanvas and use it as reference. However, opening a file no longer loads all pages. Updating the canvas now causes PageLoadRequests. In case of DocumentMode, all pages have to be loaded (to calculate the canvas size etc.), but in case of PageMode, only the visible 1-2 pages have to be loaded to display results. Another thread, queues pagerequests in low priority, to load pages when the renderingengine is idle (atm, each page is loaded separately, but the architecture is prepared to load a range of pages at once).

Not blocking UI while loading pages from source

Status quo: if the controller/canvas requests pageRects from the documentmanager and these are not cached, then the call blocks until the pagerects are loaded from source.

Idea: have the document throw an error if page not cached, documentmanager launches loadFromSourceRequest and either:

- throw exception even further, so callers have to handle what to do without the requested info
  - There are a lot of potential callers, even the customer/viewerApplication may have requested the pageRect
  - How to handle? that might sometimes be fairly hard. Most times i would just want to wait for the value anyway (maybe even in pretty much all cases)
    - BUT: if i wait, i freeze the UI again, but we have prooven, that UI freezing is inevitable (see below)
- somehow suspend the task and return on onCompleted event (technically even possible?)
  - Fundamental problem: ViewerAPI is not suited to be accessed multithreaded!
  - I would have to somehow register the originating call as callback, to be executed on OnLoadCompletedEvent
    - (What if the calls originates from the customer, though? I should solve that in top level and block the UI anyway)
    - WAIT! NO! some updates have already been made (viewport scrolled etc), you cant just reExecute the entire call!
    - You would have to somehow reenter the call at the right code line
      - but thats dumb, this implicitely freezes the UI, since we cant do any other updates, as we are not multithreadable (see fundamental problem above)
      - ⇒ So its literally impossible

Botom Line: we have to freeze the UI thread, because:

- **We cannot complete the user action, as we are missing information from the pagerectangle**
- **We cannot terminate the user action, as the present state is not stable (We would have to undo all changes made by this user action)**
- **We cannot pause the user action without blocking, as the state is not stable and other user actions would mangle up the state further.**

Solution for now:

- DocumentManager blocks and waits until all required pages are loaded

How to ever solve that:

- We have to never depend on too much information, that has to be loaded from source.
- This means we have to be able to render

Error Management Design

What throws errors which are forwarded to the application?

- File not found
- NoFileOpened (when trying to do things while no file is open)
- LicensError
- Parameter out of range (a la adobe reader coordinates > 2**16)

whats with the InteruptedExceptions in event.WaitOnEvent?

Graveyard for resolved stuff

Resolved Page

development/projects/viewerapi.txt · Last modified: 2017/01/18 15:41 by pgl