ITEM Changes in LibreOffice

Armin Le Grand

Senior Software Engineer FLOSS

Armin.Le.Grand.Extrern@allotropia.de

Collabora Online

LibreOffice Technology

Technical Day COOL days

Technical Day
COOL days

ITEM Changes in LibreOffice

Thanks for giving me the chance to do that!

- Support is much appreciated
- Many intense changes included
- Will go over the most important ones here
- Cannot go over all of them :)
- Use ‘git log|grep ITEM’, they are all tagged with ‘ITEM’
- If interested: *read* the commit comments, I put a lot of work in documenting motivation of and change itself in detail
- Pretty complete ‘overhaul’ of Item/ItemSet/ItemPool System

Technical Day
COOL days

Buffer the last access-range in ItemSet

This immediately speeds up accesses to Items in ItemSets by 6%-8%

- Not complicated
- Not dangerous
- Was the ‘starting point’ of deeper changes

Centralize add/remove Item to ItemSet

Instead of doing changes to Items in dozens of places in a non-uniform way, do them in just two places.

Those places are re-used everywhere and are a place for central changes and optimizations (instead of going over ItemSet code all the time).

Collabora Online

Technical Day
COOL days

Add STL Container Interface

Internal accesses to the array of Items used in ItemSets use now iterators wherever feasible.

That makes accesses much more readable/understandable/safer.

NOTE: It *would be* possible to go to a unorederd_set implementation now with not too much risk...

Hold Items at the ItemSet, NOT at the Pool

This breaks up the old paradigm that in reality all Items were held *not* at the ItemSet, but at the ItemPool.

That again was due to historical reasons (see extensive explanations in the commit comments).

No more double-layered calls to ItemSet and ItemPool to access Items.

Technical Day
COOL days

Remove various old hacks

Nearly all 'hacks' that somehow dealt with Items in a combination of Pool/ItemSet/local usage are solved.

Cannot go in detail over all of them here :-)

There is no more 'cleanup' code to not crash when shutting down Pools.

Some Items that did not even profit from being Items were removed from that mechanism completely, allowing removal of obstacles in Item stuff itself and further optimization of these in their specialized usage.

Technical Day
COOL days

Add SfxPoolItemHolder

This is an element of that whole Item/ItemSet/Pool mechnism/paradigm that was always missing:

It allows safe holding an Item with lean overhead - you might see it as 'ItemSet for a single Item'.

It controls refcounting and thus lifetime of the Item, by using the mentioned tooling 100% guaranteed compatible with what ItemSet does.

It solved many Item lifetime situations already and cleaned up code/understandability significantly.

There are still many situations in code that would need to be changed from using a pointer to an Item to this tooling class (not all could be changed, too many) what would also secure those Item's lifetime.

So when you have trouble with Item lifetime: Overhaul to use SfxPoolItemHolder!

Technical Day
COOL days

Make Pool-Defaults ‘global static’

97% of the defaults for Pools are now static globals again.

3% is dynamic globals, bound to the Pool. There are Items that *cannot* be Pool-independent since they are based on a SfxItemSet themselves (SfxSetItem).

This allows to globally share these, they do not even need/get ref-counted anymore. We talk about ca. 11 incarnations for Pools, with the four big ones for Writer, Calc, DrawingLayer and EditEngine.

Before, those 'static' Items (yes they were misleadingly be called that) were incarnated/deleted with each Pool, making Ref-counting unavoidable and sharing complicated and error-prone.

That was one of the causes for cleanup trouble with ItemPools/ItemSets.

Technical Day
COOL days

Add mechanism to globally share referenced Items

One of the mechanisms was to avoid double incarnations of Items by runtime costs:

Compare a newly added Item to all existing ones (histiorical reasons with saving the Pool, read commit comments if interested).

This is in principle not necessary at all - no Items needs to be shared, LO works with all Items being individual incarnations.

But it showed that there are circumstances where this makes sense due to memory footprint.

Thanks to the changes done I could re-introduce this, but this time Items get shared globally (in all apps in one LO instance), on-demand (with defined usage-count bounds to starrt at a representative point in runtime for Items where that starts to make sense).

There is a simple mechanism everyone can use to do that – overload a virtual method at the Item incarnation and implement.

There is a default implementation that is/may be used for *any* Otem with operator==.

Fun fact: For SfxBoolItems there are only *two* incarnations (true and false) globally for each used WhichID during office runtiome now :-)

NOTE: This could even get better, the WhichID is *not* a needed member of the Items, it's association to an Item is defined by the ItemSet holding it (but that is too much work to do for now).

Technical Day
COOL days

Global Static mapping of SlotIDs to WhichIDs

As a result of all that cleanups I was able to add global static hashed tables for the mapping of SlotIDs to WhichIDs (same number as global static Item defaults, 11 tables, four important/bigger ones).

Before, that mapping was done by linearly looping over the pool's definition of Items every time.

Since these mappings are massively used in UNO API all areas using that profit:

- **Load/Save**
- **UI**
- **A11y...**

To check yourself, load any presentation (e.g. the one you have to show) with the last release and a pro-master.

Biggest effect is for Draw/Impress, them Writer, then Calc. Also Chart is faster :-)

Technical Day
COOL days

Strategical Benefits Achieved

The whole mechanism is now in a form that makes it undestandable and thus allows to do bigger changes if needed/wanted.

In the form before the cleanup there was no chance to ever understand the whole mechanism completely enough to even allow do do safe changes. Everyone was ‘poking and hoping’...

This allowed most of the speedups done close to the end, planned and safe.

I *strongly* recommend to do more of those basic core cleanups in all other areas that are currently not globally understandable and thus - if we are realistic - allow no controlled/safe changes at all.

I do not need to name areas, unfortunately pretty much all are are affected. This is a natural effect for huge code bases (entropy?).

I would suggest to have people with the needed knowledge (if available) stuffed with the needed ressources to drive that forward to gain back control over our codebase.

Thank you!

Armin Le Grand (ALG)

@CollaboraOffice
hello@collaboraoffice.com
www.collaboraoffice.com

Collabora Online

LibreOffice Technology

Technical Day
COOL days