Life after Calc Core
Change

Kohei Yoshida <kohei.yoshida@collabora.com>
Topics

- What change was made in Calc core.
- Affected areas.
- Expectation vs reality.
- Going forward.
What change was made in Calc core.
Old model
New model

ScDocument

ScTable

ScColumn

mdds::multi_type_vector

svl::SharedString block

double block

EditTextObject block

ScFormulaCell block

Text width

Script type

Cell value

Broadcaster
// Do something in all formula cells in a column.

for (SCSIZE i = 0; i < maItems.size(); ++i)
{
    // Check every single non-empty cells of all types.
    ScBaseCell* pCell = maItems[i].pCell;
    if (pCell->GetCellType() == CELLTYPE_FORMULA)
    {
        ScFormulaCell* pFCell = static_cast<ScFormulaCell*>(pCell);

        // Do something with this formula cell.
    }
}
Iterating over cells (new way)

// Do something in all formula cells in a column.

struct MyFormulaHandler
{
    void operator() (size_t nRow, ScFormulaCell* pCell)
    {
        // Do something with this formula cell.
    }
};

// ...

// Only inspect formula cell blocks and skip // all the other blocks.
MyFormulaHandler aFunc;
sc::ProcessFormula(maCells, aFunc);
Before shared formula
After shared formula

ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell

ScFormulaCellGroup

ScTokenArray
Shared string concept

svl::SharedString

svl::SharedString

svl::SharedString

svl::SharedStringPool

Original string pool

Upcased string pool
String comparison (old way)

```cpp
util::TransliterationWrapper* pTransliteration = NULL;
OUString aStr1, aStr2;

if (bCaseSensitive)
    // Case sensitive transliterator.
    pTransliteration = ScGlobal::GetCaseTransliteration();
else
    // Case insensitive transliterator.
    pTransliteration = ScGlobal::GetpTransliteration();

// Parse both strings to check equality.
bool bEqual = pTransliteration->isEqual(aStr1, aStr2);
```
String comparison (new way)

svl::SharedString aStr1, aStr2;

const rtl_uString* p1;
const rtl_uString* p2;

if (bCaseSensitive)
{
    // Get pointers to original strings in the pool.
    p1 = aStr1.getData();
    p2 = aStr2.getData();
}
else
{
    // Get pointers to upcased strings in the pool.
    p1 = aStr1.getDataIgnoreCase();
    p2 = aStr2.getDataIgnoreCase();
}

// Compare pointer values.
bool bEqual = p1 == p2;
Bottom line...

- Largest refactoring effort in Calc code.
- Most critical parts of Calc code have changed.
  - **Shared formula** - ScFormulaCell no longer owns ScTokenArray at all times.
  - **Shared string** - all string objects need to be pooled.
  - Random access to cell array is no longer O(1).
Affected areas
Calc

95 %
Non-scientific estimate

Affected by core change
### What features are affected?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Undo / Redo</th>
<th>Content Rendering</th>
<th>Cell Comment</th>
<th>Formula Dependency</th>
<th>Reference Updates</th>
<th>Copy &amp; Paste</th>
<th>Database Range</th>
<th>Find &amp; Replace</th>
<th>Cell Editing</th>
<th>ODS Import / Export</th>
<th>Named Range</th>
<th>Conditional Formatting</th>
<th>SYLK Import / Export</th>
<th>VBA API Layer</th>
<th>Chart Data Provider</th>
<th>Cell Validation</th>
<th>HTML Import / Export</th>
<th>CSV Import / Export</th>
<th>External Reference</th>
<th>UNO API Layer</th>
<th>RTF Import</th>
<th>Case Conversion</th>
<th>Quattro Pro Import</th>
<th>CppUnit Test</th>
<th>DIF Import / Export</th>
<th>Change Tracking</th>
<th>...</th>
</tr>
</thead>
</table>
Expectation vs Reality
20 to 50% of the code change would cause regressions.

Almost all code changes caused regressions in some form. Some did more than others.
The worst part is behind us. We just need to fix trivial bugs.

The worst part was yet to come. Lots of medium-to-large follow-up refactoring ensued.
Check #3

Just focus on regressions during the 4.2 releases, and go back to normal for the 4.3.

We did fix the worst ones in the 4.2 cycle. But many still remain.
We already wrote a lot of unit tests for Calc core prior to the change. They should keep us safe.

We ended up doubling the amount of unit tests for Calc during the 4.2 period. We still don’t have enough.
As a responsible coder, I will fix all the bugs my change caused.

Calc’s core is too large for one coder to handle. We need multiple people who can handle bugs in Calc core.
Going forward
Challenges we face

• Squash remainder of regressions caused by the core change. The number still too high.
• Gather more people becoming comfortable handling bug fixes in Calc core.
• Build the culture of writing unit test for each and every bug fix.
Unit test is key

- One unit test is worth 20 future bug fixes.
- A bug fix does not finish until you write a test.
- Writing unit test is courtesy for your fellow developers.
- We really don’t have a choice.
Unit test is key (repeated)

- A bug fix without a unit test **WILL GET BROKEN AGAIN** in the next release.
- A bug fix with a unit test **WILL REMAIN FIXED FOREVER**.

Which one will be your choice?
THANKS!