Regression Fix: Missing Lines in DOCX

Hossein Nourikah

Mon, 19 Jul 2021

Minor updates: 30 Jan 2022

Interoperability is a very important aspect of the LibreOffice. Today, LibreOffice can load and save various file formats from many different office applications from different companies across the world. But bugs (specially regression bugs) are inevitable parts of every software. There are situations where the application does not behave as it should, and a developer should take action and fix it, so that it will behave according to the expectation of the user.

What if you encounter a bug in LibreOffice, and how does a developer fix the problem? Here we discuss the steps needed to fix a bug. In the end, we provide a test and make sure that the same problem does not happen in the future.

The article is presented in 3 parts:

1. Understanding the Bugs and QA

2. Developing a Bug Fix

3. Writing the tests and Finishing the Task

Table of Contents

1. Understanding the Bugs, Regressions and the QA

Bugs exist in any software and with a broader view, in every technological product. There are situations that a system behaves in a way that the user does not like or expect, and then the engineers come in, take action and fix that. No software can be completely bug-free, because there is no exact way to describe all the aspects of a software, even mathematically using formal methods. This has been mathematically proven.

Although we can not avoid the bugs completely, we can improve the situation. The important thing here is to document these bugs, reproduce them using sample documents, investigate the cause(s), and hopefully fix them as soon as possible while considering the priority. The process of dealing with the bugs and improving the quality of the software has the name of “Quality Assurance”, or simply “QA”.

You can take a look at LibreOffice Wiki QA section (https://wiki.documentfoundation.org/QA) and ask QA questions in IRC channel #libreoffice-qa at libera.chat network.

1.1. Bug Report

So, you have encountered a problem in the LibreOffice! In this case, first of all you should try to report the bug to Bugzilla. If a bug is not present in Bugzilla, there is a little chance that it will receive a fix.

So, everything starts with a bug report in TDF’s Bugzilla. A detailed explanation of what a good bug report should contain is available in the TDF Wiki: https://wiki.documentfoundation.org/BugReport

In short, a good bug report should have suitable summary for the problem, appropriate description and specific components/hardware/version for the context of the problem. Steps to reproduce the bug are important parts of every report, because a developer should be able to reproduce the bug in order to fix it.

The bug reporter should carefully describe the “actual results” and why it is different from the “expected results”. This is also important because the desired software’s behavior is not always as obvious as it seems to be for the bug reporter.

Let’s talk about a recently fixed regression bug: The “NISZ LibreOffice Team” reported this bug. The National Infocommunications Service Company Ltd. (NISZ) has an active team in LibreOffice development and QA (https://numbertext.org/libreoffice/BuildALibreOfficeTeam_2018.pdf).

This is the bug title and description provided by the NISZ LibreOffice Team:

Bug 123321 - FILEOPEN | DOC, missing longer line when saved in LO (shorter line remains)
https://bugs.documentfoundation.org/show_bug.cgi?id=123321
Description:
When the attached original document gets saved in LO as doc the middle line gets its length resized.
Steps to Reproduce:

1. Open the attached doc in LO.
2. Save it as doc.
3. Reload.

4. Notice the changes.

Actual Results: The middle arrow gets smaller.
Expected Results: It should stay the same size even after saving in LO.
Reproducible: Always
User Profile Reset: No

Every bug has a number associated to it, and at TDF bugzilla, it is referred to by its number, like tdf#123321.

1.2. Bug Confirmation

After reporting a bug, someone else should check and see If it is reproducible or not. If so, this person then confirms the bug and sets its status to “New”. In this case, a user with the name of **Timur** from the QA team of volunteers (https://wiki.documentfoundation.org/QA/Team#QA_team_volunteers) has confirmed this bug. It is necessary that someone else other than the original bug reporter confirms the bug report.

Here, the bug reporter has provided several examples:

Attachments

- The original file (39.50 KB, application/msword)
- The saved file. (24.00 KB, application/msword)
- Screenshot of the original and exported document side by side in Writer. (298.50 KB, image/png)
- Minimized test document in docx format (19.66 KB, application/vnd.openxmlformats-officedocument.wordprocessingml.document)

Opening the first file, we see that it contains several shapes. 4 ellipses, 2 diagonal lines, and a vertical line. But if we look closely, we find out that the vertical line actually consists of 3 different vertical lines. A good way to understand this is by try selecting the line, and then pressing the tab to select the other lines.

The Shapelinelength_min.docx only contains 3 overlapped vertical lines (the overlap is not important).

- First one on the top with the length 0.17" (Verified in Word 2007)
- Second one in the middle with the length of 1" (Verified in Word 2007)
- Third one at the bottom with the length of 2.77" (Verified in Word 2007)

When you save it in LibreOffice and reload it, the first and the third vertical lines disappear, but if you select the second one (the only visible after save and reload), you can select the two other lines by pressing “tab” button. If you look at the size of these two lines, you will see that both have the length of 0″.

By opening the examples, saving and reloading them, we can verify that the bug is present even in the latest master build.

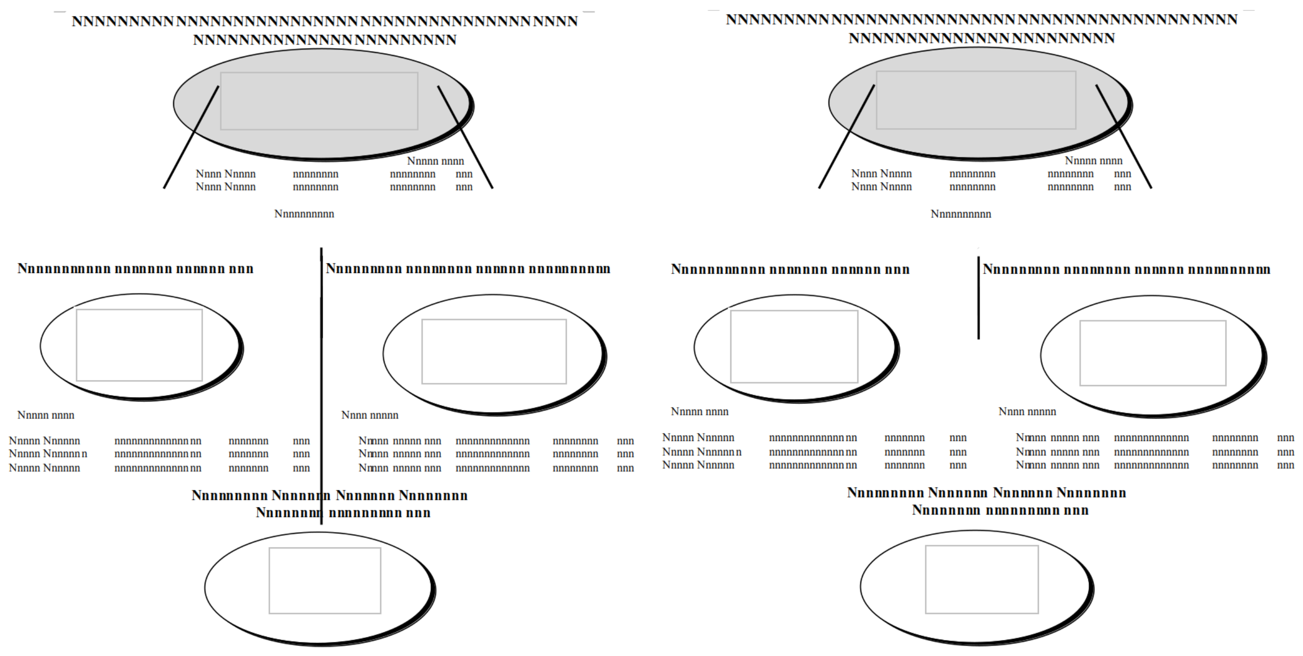


Figure 1. The visible lines in the middle becomes smaller after save and reload

1.3. Bisect / Bibisect for a Regression bug

Regression bugs (https://en.wikipedia.org/wiki/Software_regression) are special kinds of bugs. They describe a feature that was previously working well, but at some point a change in the code has caused it to stop working. They are source of disappointment for the user, but they are easier to fix for the developers compared to other bugs! Why? Because every single change to the LibreOffice code will remain in the source code management system (Git), it is possible to find that which change actually introduced the regression bug.

Git has a special tool for this purpose, which is call `bisect`. It uses a binary search to find the commit that introduced the bug. But for LibreOffice which is a huge piece of software consisting of roughly 10 million lines of code, this can take a lot of time. So, we use a trick: bisecting with the binaries! If you have access to every built LibreOffice for the commits, you can use git bisect command to find the source for problem in a very short time: This is called `bibisect`!

A very detailed description for bibisecting a regression can be found in the TDF Wiki: https://wiki.documentfoundation.org/QA/Bibisect

Aron Budea from Collabora Productivity Ltd’s (https://www.collaboraoffice.com/about-us/) core engineering team did the bibisect, and now we know the exact commit that caused the problem:

Bibisected to the following commit using repo bibisect-win32-6.0.

https://git.libreoffice.org/core/+/d72e0cadceb0b43928a9b4f18d75c9d5d30afdda

Watermark: tdf#91687 correct size in the .doc

Export:
* Watermarks saved using Writer were very small in the MSO.
  Export fUsegtextFStretch property in the Geometry Text
  Boolean Properties.
* tdf#91687: SnapRect contains size of Watermark after rotation.
  We have to export size without rotation.

Import:
* When import set height depending on used font and width.
  Text will keep the ratio. Remember the padding for export.

* Added unit test
* Introduced enum to avoid magic numbers for stretch and best fit
  properties.

This bug is a regression. The DOCX import/export was working well before this commit, but after that, it doesn’t work: Good catch!

In the next section, we talk about how we can create a fix for the bug.

2. Developing a Bug Fix for the Regression Bug

In this section, we try to develop a solution for the problem with our knowledge of C++.

The first thing we need is to build the LibreOffice core. Depending on the platform, you can find the instructions in the TDF wiki.

See the instructions for building LibreOffice on:

- Linux and *BSD systems: https://wiki.documentfoundation.org/Development/BuildingOnLinux
- macOS: https://wiki.documentfoundation.org/Development/BuildingOnMac
- on Windows: https://wiki.documentfoundation.org/Development/BuildingOnWindows
- for Android: https://wiki.documentfoundation.org/Development/BuildingForAndroid

2.1. Finding the Responsible Change

So, now we know the changed files through this commit. One (or possibly all) of the changes is responsible for the regression. So we try to find a minimize set of changes that lead to the problem. We can use `--name-only` option from git:

```
$ git show d72e0cadceb0b43928a9b4f18d75c9d5d30afdda –name-only
```

Changed files:
1. `filter/source/msfilter/escherex.cxx`
2. `filter/source/msfilter/msdffimp.cxx`

3. `include/svx/msdffdef.hxx`
4. `sw/qa/extras/ww8export/data/tdf91687.doc`
5. `sw/qa/extras/ww8export/ww8export2.cxx`
6. `sw/source/filter/ww8/wrtw8esh.cxx`

Sometimes we have to go back to this specific commit using `git checkout` command to be able to work on the exact commit before the one that created the bug. But, most of the time, reverting the specific commit is easier, and the build will be much faster. Also, building older versions of LibeOffice is tricky. We can instead invoke this command:

```
$ git revert d72e0cadceb0b43928a9b4f18d75c9d5d30afdda
```

Now we should try to resolve the possible conflicts, then build the code and make sure the problem has gone away! Then we can try to re-introduce one or more changes to achieve a minimal set of changes that reproduces the problem.

We can start at the file level to find the relevant files to work on. Files 4 and 5 are related to the tests, so they should have no impact. File 1 and 2 seem to be related, as both are from `filter/source/msfilter` folder, file 3 is also related to these two files. After leaving out all the files, we see that changes from 6 (`sw/source/filter/ww8/wrtw8esh.cxx`) is enough to reproduce the problem. So, we have few lines of code changes in front of us:

```
@@ -756,7 +756,12 @@ void PlcDrawObj::WritePlc( WW8Export& rWrt ) const
                OSL_ENSURE(pObj, "Where is the SDR-Object?");
                if (pObj)
                {
-                   aRect = pObj->GetSnapRect();
+                   aRect = pObj->GetLogicRect();
+
+                    // We have to export original size with padding
+                    const SfxItemSet& rSet = pObj->GetMergedItemSet();
+                    const SdrMetricItem* pItem = static_cast<const
SdrMetricItem*>(rSet.GetItem(SDRATTR_TEXT_UPPERDIST));
+                    aRect.SetSize(Size(aRect.GetWidth(), aRect.GetHeight()
+ pItem->GetValue()));
                }
            }
```

If we work further on this piece of code, we will find that the change from `pObj->GetSnapRect()` to `pObj->GetLogicRect()` is the source of regression: Good catch!

2.2. Creating a Fix for the Regression bug

As described in the previous section, going back to `pObj->GetSnapRect()` fixes the problem, but wait! Isn't that change supposed to fix a problem?

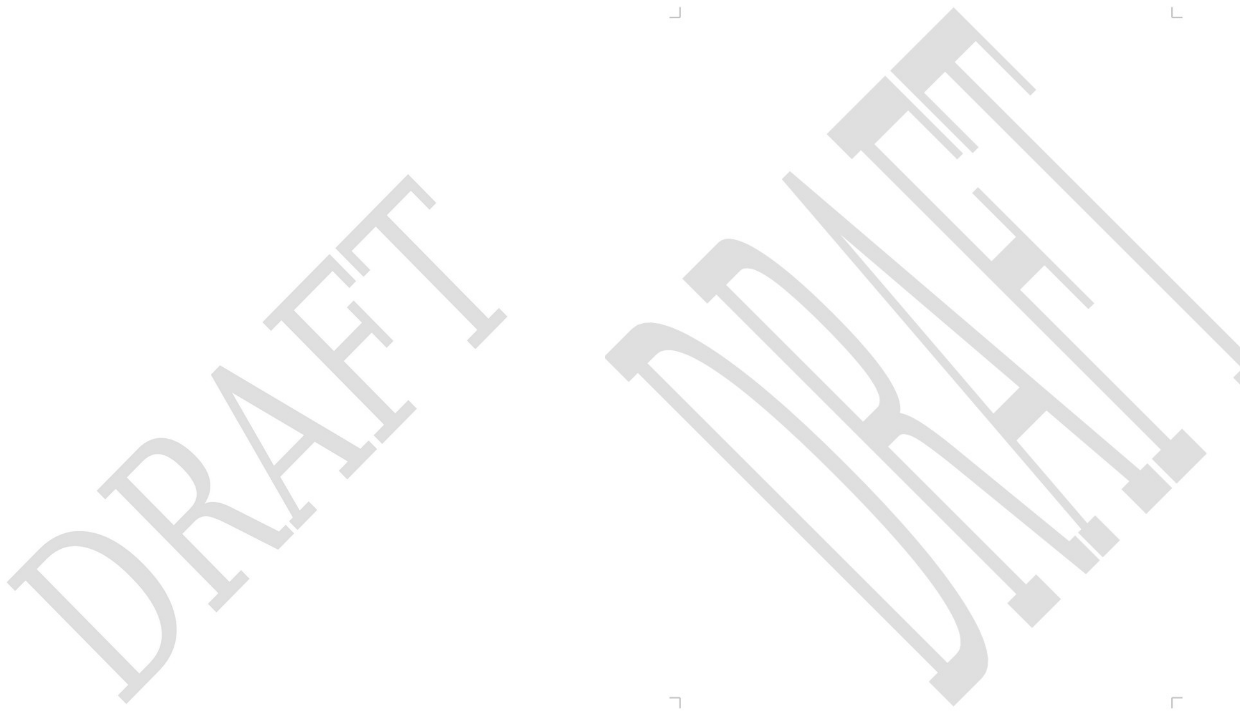


Figure 2: Wrong increase of watermark size after save and reload

If we go back to git master using:

```
git reset --hard HEAD^1
```

and then only change `pObj->GetSnapRect()` to `pObj->GetSnapRect()`, this test fails with the change. Try:

```
$ make CPPUNIT_TEST_NAME="testTdf91687" -sr CppunitTest_sw_ww8export2
```

If we take a look at the tdf#91687 (https://bugs.documentfoundation.org/show_bug.cgi?id=91687) in the Bugzilla, we see the example `sw/qa/extras/ww8export/data/tdf91687.doc` that contains a watermark (Figure 2) that gets bigger by save and reload! Looking more carefully to the changes, it becomes clear that rotation is important here. If we change the watermark example in order to set rotation to zero, nothing happens.

The problem is not specific to the watermarks: Saving and re-loading any custom shape creates such a wrong increase in size. On the other hand, this does not have any effect on the lines, rotated or not.

2.3. Fixing Side Effects

In order to fix the undesired side effects, we should make a difference between the lines and the complex shapes. So, what is the difference? If we look closely to the code, we find out that `pObj` object is from `SdrObj` class, which is abstract DrawObject. The documentation for this class can be found here:
https://docs.libreoffice.org/svx/html/classSdrObject.html

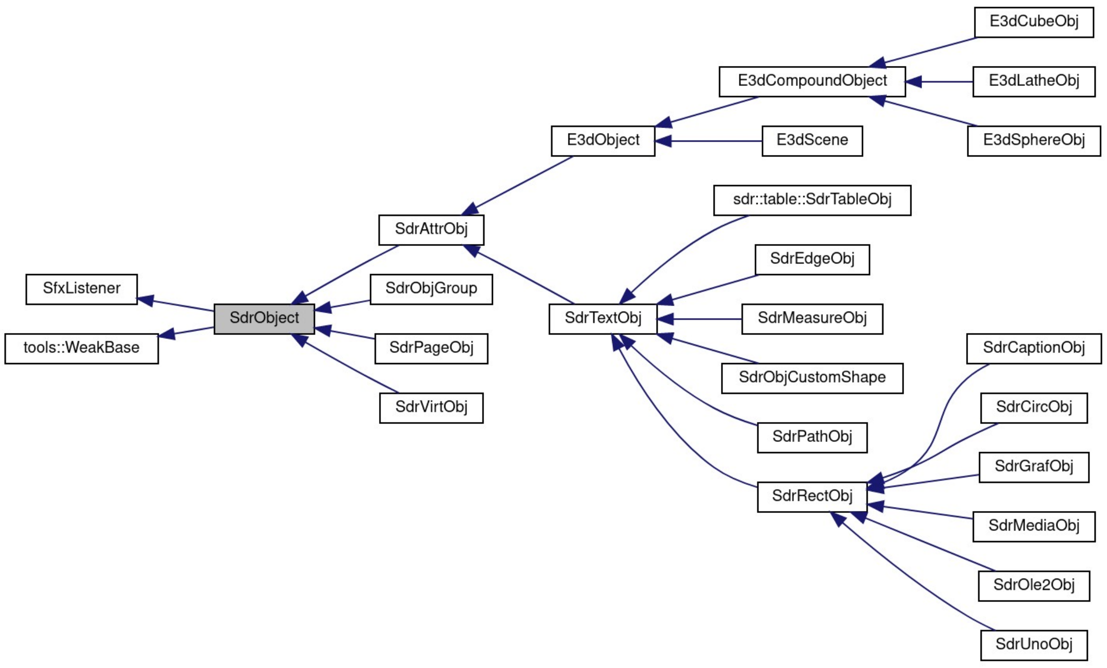


Figure 2. Class hierarchy for SdrObject

So, objects from different `SdrObject` subclasses are sent here, and appropriate `GetSnapRect()` or `GetLogicRect()` is called. But which method? According to the runtime polymorphism, this is determined at runtime. So without debugging, nothing becomes clear.

By setting a break-point at this modified line and stepping into the `GetLogicRect()` function, it becomes clear that two types of object are created for the shapes:

- `pObj` is `SdrObjCustomShape` (x4): representing 4 ellipeses
- `pObj` is `SdrPathObj` (x5): representing 5 lines

So, the trick would be creating a `GetLogicRect()` method for `SdrPathObj` and make it invoke `GetSnapRect()`. In this way, lines and custom shapes are treated differently.

We declare `GetLogicRect()` in `include/svx/svdopath.hxx`:

```
virtual const tools::Rectangle& GetLogicRect() const override;
```

and add its implementation in `svx/source/svdraw/svdopath.cxx`:

```
const tools::Rectangle &SdrPathObj::GetLogicRect() const
{
    return GetSnapRect();
}
```

Then we rebuild the LibreOffice by invoking `make`, and then start the LibreOffice from `instdir/program/soffice`. After loading, saving, and re-loading the example DOCX file, we see that this fix actually works!

In the next section, we talk about the steps needed to get our changes merged into the LibreOffice Code.

3. Writing Tests and Finishing the Task

In this section, we discuss how to write a test, submit the patch for fixing the regression bug to the Gerrit and try to get it into the LibreOffice code.

3.1. Writing a Test for the Regression

In order to make sure that this regression will not happen again, we should create test for every bug we fixed. If we don't do this, there is a big chance that this bug will occur again and again.

But how to start writing a test? The answer is that there are ~3000 tests available in `sw/qa/extras/` folder, and there is a tiny document `sw/qa/extras/README` that describes them. These tests can be a basis for writing a new test. For example, test for tdf#91687 consist of these lines:

```
DECLARE_WW8EXPORT_TEST(testTdf91687, "tdf91687.doc")
{    // Exported Watermarks were resized
    uno::Reference<drawing::XShape> xWatermark = getShape(1);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(5172), xWatermark→getSize().Height);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(18105), xWatermark→getSize().Width);
}
```

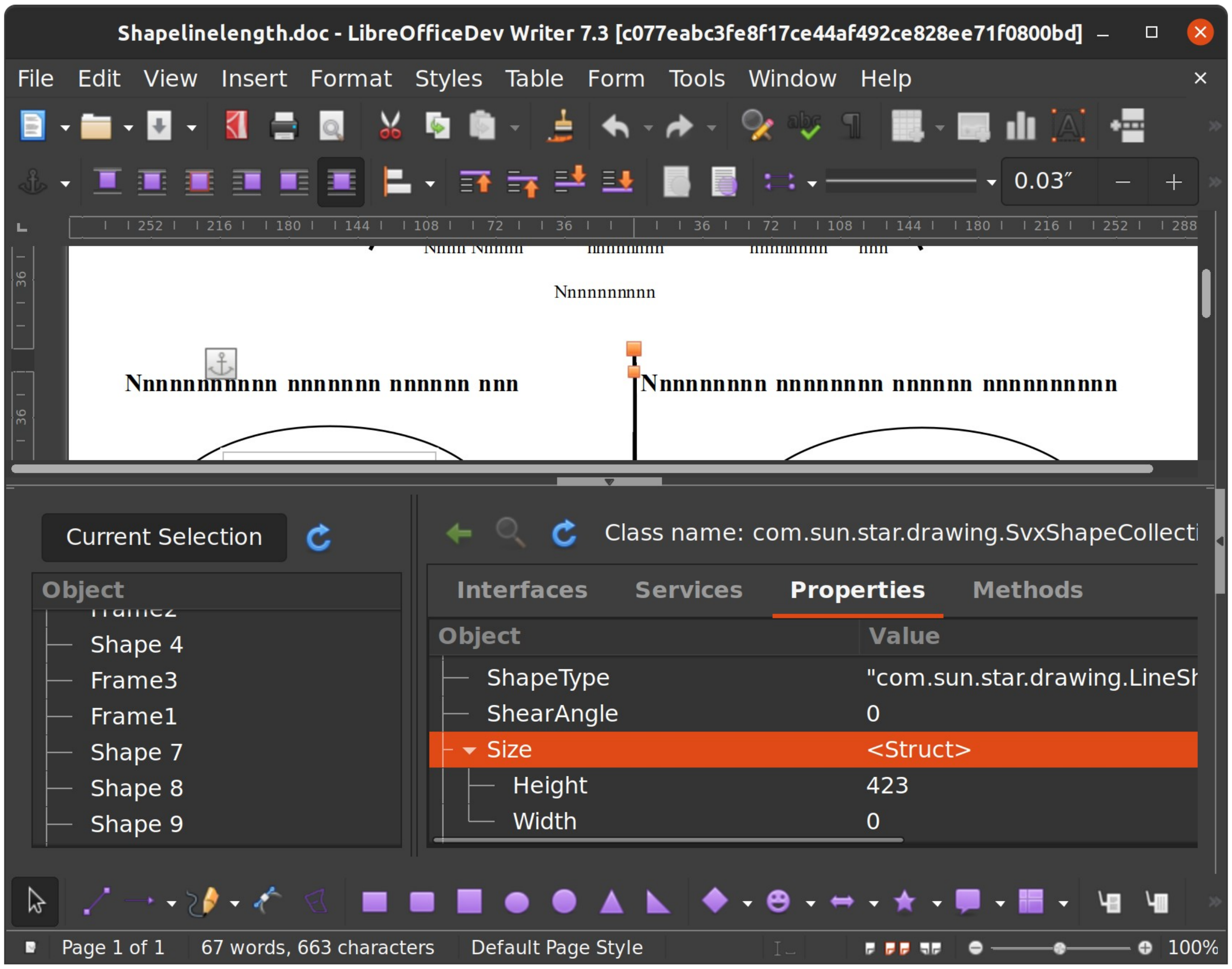
It uses a macro `DECLARE_WW8EXPORT_TEST` too, that checks some properties of the document after loading it, and then again after saving and reloading it. If some property is erroneously changed, the test will fail, and you will get noticed. As you can see, the name of the test, and the name of the test document in which here is .doc file, come from the bug number.

The test for this regression should check for the position and size of the shapes, so there are similarities between the our desired test and the test above. But how can we find the properties of the elements of the document? The answer is the new UNO object inspection tool for the LibreOffice:

https://blog.documentfoundation.org/blog/2021/03/02/update-on-tender-for-a-built-in-uno-object-inspection-tool-in-libreoffice/

As in figure 4, the size for Shape7 which is the first vertical line, is visible on the right.

In previous versions, tools called **XRAY** or **MRI** were used for object inspection.


Shapelinelength.doc - LibreOfficeDev Writer 7.3 [c077eabc3fe8f17ce44af492ce828ee71f0800bd]
File Edit View Insert Format Styles Table Form Tools Window Help
0.03″
Nnnnnnnnn
Nnnnnnnnnn nnnnnnn nnnnnn nnn
Nnnnnnnnn nnnnnnnn nnnnnn nnnnnnnnnn
Current Selection
Class name: com.sun.star.drawing.SvxShapeCollecti
Interfaces Services Properties Methods
Page 1 of 1 67 words, 663 characters Default Page Style 100%

Object
Frame2
Shape 4
Frame3
Frame1
Shape 7
Shape 8
Shape 9

Object	Value
ShapeType	"com.sun.star.drawing.LineSh
ShearAngle	0
Size	<Struct>
Height	423
Width	0

Figure 3. UNO object inspect tool for LibreOffice

The completed test comes below. It takes `Shape7`, `Shape8` and `Shape9` which are the three vertical lines, and ensures that they have the same size in the first load, and after save and reload. Without the fix, some of these test fail, which are size checking for the first and last vertical lines. Their size was reduced to ~0, so it is justifiable that previously they were erroneously disappearing after save and reload. The test passes after applying the fix.

```
DECLARE_WW8EXPORT_TEST(testTdf123321, "shapes-line-ellipse.doc")
{
    // These are the 3 lines in which 1st and 3rd one were disappearing before
    uno::Reference<drawing::XShape> l1 = getShape(7);
    uno::Reference<drawing::XShape> l2 = getShape(8);
    uno::Reference<drawing::XShape> l3 = getShape(9);

    // first line (smallest)
    // Fails without the fix: Expected: 423, Actual: 2
    CPPUNIT_ASSERT_EQUAL(sal_Int32(423), l1->getSize().Height);
    // Fails without the fix: Expected: 0, Actual: 2
    CPPUNIT_ASSERT_EQUAL(sal_Int32(0), l1->getSize().Width);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(7908), l1->getPosition().X);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(37), l1->getPosition().Y);

    // second line (larger)
    CPPUNIT_ASSERT_EQUAL(sal_Int32(2542), l2->getSize().Height);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(2), l2->getSize().Width);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(7916), l2->getPosition().X);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(289), l2->getPosition().Y);

    // third line (largest)
    // Fails without the fix: Expected: 7027, Actual: 2
    CPPUNIT_ASSERT_EQUAL(sal_Int32(7027), l3->getSize().Height);
    // Fails without the fix: Expected: 0, Actual: 2
    CPPUNIT_ASSERT_EQUAL(sal_Int32(0), l3->getSize().Width);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(7911), l3->getPosition().X);
    CPPUNIT_ASSERT_EQUAL(sal_Int32(231), l3->getPosition().Y);
}
```

As you can see, not all the tests fail without the fix, but they are in place to protect the correct load and save of the shapes in the future.

3.2. Cleaning up and submitting the changes

After we finished the fix, we commit the changes, and submit it to gerrit.

```
$ git add include/svx/svdopath.hxx svx/source/svdraw/svdopath.cxx sw/qa/extras/ww8export/data/shapes-line-ellipse.doc sw/qa/extras/ww8export/ww8export2.cxx
```

```
$ git commit
```

Let’s see our latest commit:

```
$ git show --name-only

commit b3cb8e3abef6cee2711581a7af12773a7cfcdc32 (HEAD -> master)

Author: Hossein <hossein@libreoffice.org>

Date:   Mon Jul 19 13:03:42 2021 +0200


    tdf#123321 Fix DOC/DOCX export bug which made some lines dissappear


    * Fix the wrong calculation of SdrPathObj (line) extent while exporting

      to .doc and .docx formats. This resolves tdf#123321 which cause some

      lines to disappear after save and reload. This fix does not break

      tdf#91687

    * added tests to make sure this does not happen again, based on the
```

```
      .doc file provided in BugZilla
    * Loading the bad exported file in MSWord is OK, but it does not work
      in LibreOffice, and some of the lines will be hidden. This needs
      additional work.

    The regression came from d72e0cadceb0b43928a9b4f18d75c9d5d30afdda

    Change-Id: Ic5a7af4d29df741204c50590728542e674ee9f91

include/svx/svdopath.hxx
svx/source/svdraw/svdopath.cxx
sw/qa/extras/ww8export/data/shapes-line-ellipse.doc
sw/qa/extras/ww8export/ww8export2.cxx
```

The actual commit is longer, but for making it shorter, we only listed the name of the changed or new files.

```
$ ./logerrit submit master
```

This is with the assumption that we have successfully configured Gerrit before. One should set up Gerrit before doing this. A step-by-step manual is presented in The Document Foundation Wiki.

https://wiki.documentfoundation.org/Development/gerrit/setup

3.3. Getting the Submission Merged

Developers and mentors review submissions from the contributors, and merge them into to the LibreOffice code after the submitter has done all the requested fixes and improvements. One of the people with commit access decides upon that. Commit access is granted to the developers who had a good record of contributions and are trusted to change the code base directly.

This is the code review for the submission in the Gerrit:

https://gerrit.libreoffice.org/c/core/+/119116

Jenkins builds the code for each submission in a CI (continues integration) process. If the build is successful, you will get a Verified +1 from Jenkins.

If, for any reason, one of the builds for several CI platforms fails, you will get Verified -1 from Jenkins. Then, you will have to fix the problem and re-submit again. Please note that there are situations that some tests fail because of the Jenkins problem itself. If that is the case, you should ask people in the #libreoffice-dev irc channel to invoke a resume for you, or try rebasing to initiate a rebuild.

The original submission (patch set 1) lacked the tests, so a test was requested. After doing the improvements, including a change in the title and uploading seven patch sets in total, it was given the Code Review +2 from the reviewer.

Here, Miklos Vajna from Collabora has done the reviews and merging. He had the appropriate access, and merged the proposed changes into the LibreOffice code. After that, everyone can pull the changes from Git:

https://git.libreoffice.org/core/commit/bda4d7a7c3fcc259e023f568606be5dcba818db9

3.4 Marking the Bug as Fixed

After merging the commit, the journey ends with marking the bug as fixed. Looking at the other possible registrations of the symptoms at TDF Bugzilla (https://bugs.documentfoundation.org/), it turns out that another bug can be considered a duplicate of this one:

Bug 127296 - FILESAVE: DOC Rotated line loses rotation when saved

https://bugs.documentfoundation.org/show_bug.cgi?id=127296

The above bug report was in 2019-09-02, and it is newer than the one we have fixed. So, we can mark tdf#127296 as a duplicate of this one. After that, we can safely change the status of tdf#123321 to “RESOLVED FIXED”, and we’re done!

3.5 Want to Get Involved?

If you want to help in LibreOffice development, you can help in fixing the bugs. Looking at the statistics from TDF’s Bugzilla, we see that there is a total of ~12k open bugs (https://bugs.documentfoundation.org/page.cgi?id=weekly-bug-summary.html), in which ~1.3k of them are regression bugs. Among them, around 1.1k are bibisected . They are usually easier to fix, because the root of the problem is visible.

You can go through the same process that was discussed in the series of articles here to fix the bugs and submit your patch, and if you had any questions, #libreoffice-dev (irc://irc.libera.chat/#libreoffice-dev) is a good place to ask. You should also join the LibreOffice development mailing list. See the instructions in the Wiki (https://wiki.documentfoundation.org/Development/Mailing_List).