Here how deepwiki Devin trained on LibreOffice-core code-base responds to my report.

Answer
The LibreOffice features involved are the Sidebar framework (sfx2/sidebar) and the AutoRecovery service. After a crash, AutoRecovery restores documents; the Sidebar's SidebarController then restores the sidebar's open/closed state and width. The width logic uses `mnSavedSidebarWidth` and `SetChildWindowWidth`, and initial defaults are set in `SidebarChildWindow` [1](#0-0) [2](#0-1) .

How the sidebar width is determined
- Initial creation: `SidebarChildWindow` sets the initial size using `GetDefaultWidth` and, for first-start Writer/Calc, collapses the sidebar to `TabBar::GetDefaultWidth()` [3](#0-2) .
- Runtime width: When a deck is opened, `SidebarController::UpdateDeckOpenState` restores the saved width (`mnSavedSidebarWidth`) or falls back to the default width [4](#0-3) .
- Saving width: When the deck is closed or the splitter is used, the current width is stored in `mnSavedSidebarWidth` [5](#0-4) .
- Deck switching: `OpenDeckThenToggleDeck` and `SwitchToDeck` ensure the sidebar is wide enough for the requested deck, using `mnSavedSidebarWidth` or the deck's minimal width [6](#0-5) .

Crash recovery interaction
- AutoRecovery restores documents after a crash; it does not directly manage sidebar UI state [7](#0-6) .
- After recovery, when the sidebar is opened, `SidebarController` uses its persisted `mnSavedSidebarWidth`. If that value was not correctly saved or was reset during the crash/recovery sequence, the sidebar may fall back to a default or minimal width, or in some cases expand excessively if the saved width is unexpectedly large [8](#0-7) .

Likely contributors to issue B
- Incomplete or incorrect persistence of `mnSavedSidebarWidth` during the crash.
- Re-initialization of `SidebarChildWindow` on recovery treating the session as a first start, applying the collapsed-to-tabbar width or a default width [9](#0-8) .
- Deck-switching logic choosing a larger width if `mnWidthOnSplitterButtonDown` is zero or if the deck's minimal width exceeds the saved width [10](#0-9) .

Notes
- The crash itself (issue A) is likely outside the sidebar code; the sidebar width reset (issue B) is a side effect of state persistence and restoration.
- Sidebar state is saved per context/application; master documents with multiple sub-documents may involve multiple contexts, which could complicate state restoration [11](#0-10) .

- The sidebar includes many Writer-specific panels (e.g., `WriterInspectorTextPanel`), but the width management is generic in `sfx2/sidebar` [12](#0-11) .

Citations

File: sfx2/source/sidebar/SidebarChildWindow.cxx (L33-61)
```text
SidebarChildWindow::SidebarChildWindow(vcl::Window* pParentWindow,
sal_uInt16 nId,
                                       SfxBindings* pBindings,
SfxChildWinInfo* pInfo)
    : SfxChildWindow(pParentWindow, nId)
{
    auto pDockWin = VclPtr<SidebarDockingWindow>::Create(
        pBindings, *this, pParentWindow, WB_STDDOCKWIN |
WB_OWNERDRAWDECORATION | WB_CLIPCHILDREN
                                             | WB_SIZEABLE |
WB_3DLOOK);
    SetWindow(pDockWin);
    SetAlignment(SfxChildAlignment::RIGHT);

    pDockWin->SetHelpId(HID_SIDEBAR_WINDOW);
    pDockWin->SetOutputSizePixel(Size(GetDefaultWidth(pDockWin),
450));

    if (pInfo && pInfo->aExtraString.isEmpty() && pInfo->aModule !=
"sdraw"
        && pInfo->aModule != "simpress" && pInfo->aModule !=
"smath")
    {
        // When this is the first start (never had the sidebar open
yet),
        // default to non-expanded sidebars in Writer and Calc.
        //
        // HACK: unfortunately I haven't found a clean solution to
do
        // this, so do it this way:
        //
        if (!comphelper::LibreOfficeKit::isActive())
        {
            pDockWin->SetSizePixel(
                Size(TabBar::GetDefaultWidth(),
                     pDockWin->GetSizePixel().Height()));
        }
    }
```

File: sfx2/source/sidebar/SidebarController.cxx (L685-700)
```text
    // Make sure the sidebar is wide enough to fit the requested
content
    if (mpCurrentDeck && mpTabBar)
    {
```

```
        sal_Int32 nRequestedWidth = mpCurrentDeck->GetMinimalWidth()
+ TabBar::GetDefaultWidth();
        // if sidebar was dragged
        if(mnWidthOnSplitterButtonDown > 0 &&
mnWidthOnSplitterButtonDown > nRequestedWidth){
            SetChildWindowWidth(mnWidthOnSplitterButtonDown);
        }else{
            // tdf#150639 The mnWidthOnSplitterButtonDown is
initialized to 0 at program start.
            // This makes every call to take the else case until the
user manually changes the
            // width, but some decks such as Master Slides have the
mnMinimalWidth too low which
            // makes them too narrow for the content they should
display to the user.
            SetChildWindowWidth(nRequestedWidth >
mnSavedSidebarWidth ? nRequestedWidth

: mnSavedSidebarWidth);
        }
    }
```

File: sfx2/source/sidebar/SidebarController.cxx (L1280-1410)
```text
void SidebarController::UpdateDeckOpenState()
{
    if ( ! mbIsDeckRequestedOpen.has_value() )
        // No state requested.
        return;

    const sal_Int32 nTabBarDefaultWidth = TabBar::GetDefaultWidth();

    // Update (change) the open state when it either has not yet
been initialized
    // or when its value differs from the requested state.
    if ( mbIsDeckOpen.has_value() && *mbIsDeckOpen ==
*mbIsDeckRequestedOpen )
        return;

    if (*mbIsDeckRequestedOpen)
    {
        if (!mpParentWindow->IsFloatingMode())
        {
            if (mnSavedSidebarWidth <= nTabBarDefaultWidth)

SetChildWindowWidth(SidebarChildWindow::GetDefaultWidth(mpParentWind
ow));
            else
                SetChildWindowWidth(mnSavedSidebarWidth);
        }
        else
        {
            // Show the Deck by resizing back to the original size
```

```
(before hiding).
            Size aNewSize(mpParentWindow->GetFloatingWindow()-
>GetSizePixel());
            Point aNewPos(mpParentWindow->GetFloatingWindow()-
>GetPosPixel());

            aNewPos.setX(aNewPos.X() - mnSavedSidebarWidth +
nTabBarDefaultWidth);
            aNewSize.setWidth(mnSavedSidebarWidth);

            mpParentWindow->GetFloatingWindow()-
>SetPosSizePixel(aNewPos, aNewSize);

            if (comphelper::LibreOfficeKit::isActive())
            {
                // Sidebar wide enough to render the menu; enable
it.
                mpTabBar->EnableMenuButton(true);

                if (const SfxViewShell* pViewShell = mpViewFrame-
>GetViewShell())
                {
                    const std::string uno =
UnoNameFromDeckId(msCurrentDeckId, GetCurrentContext());
                    if (!uno.empty())
                        pViewShell-
>libreOfficeKitViewCallback(LOK_CALLBACK_STATE_CHANGED,

OString(uno + "=true"));
                }
            }
        }
    }
    else
    {
        if ( ! mpParentWindow->IsFloatingMode())
            mnSavedSidebarWidth =
SetChildWindowWidth(nTabBarDefaultWidth);
        else
        {
            // Hide the Deck by resizing to the width of the TabBar.
            Size aNewSize(mpParentWindow->GetFloatingWindow()-
>GetSizePixel());
            Point aNewPos(mpParentWindow->GetFloatingWindow()-
>GetPosPixel());
            mnSavedSidebarWidth = aNewSize.Width(); // Save the
current width to restore.

            aNewPos.setX(aNewPos.X() + mnSavedSidebarWidth -
nTabBarDefaultWidth);
            if (comphelper::LibreOfficeKit::isActive())
            {
                // Hide by collapsing, otherwise with 0x0 the client
might expect
```

```
                // to get valid dimensions on rendering and not
collapse the sidebar.
                aNewSize.setWidth(1);
            }
            else
                aNewSize.setWidth(nTabBarDefaultWidth);

            mpParentWindow->GetFloatingWindow()-
>SetPosSizePixel(aNewPos, aNewSize);

            if (comphelper::LibreOfficeKit::isActive())
            {
                // Sidebar too narrow to render the menu; disable
it.
                mpTabBar->EnableMenuButton(false);

                if (const SfxViewShell* pViewShell = mpViewFrame-
>GetViewShell())
                {
                    const std::string uno =
UnoNameFromDeckId(msCurrentDeckId, GetCurrentContext());
                    if (!uno.empty())
                        pViewShell-
>libreOfficeKitViewCallback(LOK_CALLBACK_STATE_CHANGED,

OString(uno + "=false"));
                }
            }
        }

        if (mnWidthOnSplitterButtonDown > nTabBarDefaultWidth)
            mnSavedSidebarWidth = mnWidthOnSplitterButtonDown;
        mpParentWindow->SetStyle(mpParentWindow->GetStyle() &
~WB_SIZEABLE);
    }

    NotifyResize();
}

bool SidebarController::CanModifyChildWindowWidth()
{
    SfxSplitWindow* pSplitWindow = GetSplitWindow();
    if (pSplitWindow == nullptr)
        return false;

    sal_uInt16 nRow (0xffff);
    sal_uInt16 nColumn (0xffff);
    if (pSplitWindow->GetWindowPos(mpParentWindow, nColumn, nRow))
    {
        sal_uInt16 nRowCount (pSplitWindow-
>GetWindowCount(nColumn));
        return nRowCount==1;
    }
    else
```

```
        return false;
}

sal_Int32 SidebarController::SetChildWindowWidth (const sal_Int32
nNewWidth)
{
    SfxSplitWindow* pSplitWindow = GetSplitWindow();
    if (pSplitWindow == nullptr)
        return 0;

    sal_uInt16 nRow (0xffff);
    sal_uInt16 nColumn (0xffff);
    pSplitWindow->GetWindowPos(mpParentWindow, nColumn, nRow);
    const tools::Long nColumnWidth (pSplitWindow-
>GetLineSize(nColumn));

    vcl::Window* pWindow = mpParentWindow;
    const Size aWindowSize (pWindow->GetSizePixel());

    pSplitWindow->MoveWindow(
        mpParentWindow,
        Size(nNewWidth, aWindowSize.Height()),
        nColumn,
        nRow,
        false);
```

File: sfx2/source/sidebar/SidebarController.cxx (L1591-1601)
```text
void SidebarController::saveDeckState()
{
    // Impress shutdown : context (frame) is disposed before sidebar
disposing
    // calc writer : context (frame) is disposed after sidebar
disposing
    // so need to test if GetCurrentContext is still valid regarding
msApplication
    if (GetCurrentContext().msApplication != "none")
    {
        mpResourceManager->SaveDecksSettings(GetCurrentContext());
        mpResourceManager->SaveLastActiveDeck(GetCurrentContext(),
msCurrentDeckId);
    }
}
```

File: framework/source/services/autorecovery.cxx (L123-147)
```text
 * AutoRecovery handles 3 types of recovery, as well as periodic
document saving
 *   1a) timed, automatic saving of the documents (aka UserAutoSave)
 *   or more commonly:
 *   1b) timed, ODF, temporary recovery files created in the backup
folder (default setting)
```

```
 *      -temporary: deleted when the document itself is saved
 *      -handles the situation where LO immediately exits (power
outage, program crash, pkill -9 soffice)
 *          -not restored immediately (user needs to restart
LibreOffice)
 *          -no guarantee of availability of recovery file (since
deleted on document save)
 *           or original document (perhaps /tmp, removeable disk,
disconnected server).
 *          -TODO tdf#57414: if SessionSave not desired, don't
recover unmodified files.
 *   2) emergency save-and-restart immediately triggers creation of
temporary, ODF, recovery files
 *      -handles the situation where LO is partially functioning
(pkill -6 soffice)
 *          -restore attempted immediately, so try to restore entire
session - all open files
 *          -always create recovery file for every open document in
emergency situation
 *      -works without requiring AutoRecovery to be enabled
 *   3) session save on exit requested by OS or user
 *      -triggered by OS's shutdown/logout
 *          -appears to be purely theoretical: no known working OS
at the moment.
 *          -also no known way for user to initiate within LO
(tdf#146769)
 *      -same as emergency save, except maybe more time critical -
OS kill timeout
 *          -not restored until much later - the user has stopped
doing computer work
 *          -always create recovery file for every open document:
needed for /tmp, disconnected docs
 *
 * All of these use the same recovery dialog - re-opening all the
files listed in the RecoveryList
 * of the user's officecfg settings.
```

File: sw/source/uibase/sidebar/WriterInspectorTextPanel.cxx
(L20-59)
```text
#include "WriterInspectorTextPanel.hxx"

#include <doc.hxx>
#include <ndtxt.hxx>
#include <docsh.hxx>
#include <wrtsh.hxx>
#include <unoprnms.hxx>
#include <unocoll.hxx>
#include <unosection.hxx>
#include <editeng/unoprnms.hxx>
#include <com/sun/star/text/XBookmarksSupplier.hpp>
#include <com/sun/star/text/XTextSectionsSupplier.hpp>
#include <com/sun/star/text/XTextRange.hpp>
```

```
#include <com/sun/star/text/XTextRangeCompare.hpp>
#include <com/sun/star/beans/XPropertySet.hpp>
#include <com/sun/star/beans/XPropertyState.hpp>
#include <com/sun/star/style/XStyleFamiliesSupplier.hpp>
#include <com/sun/star/table/BorderLine2.hpp>
#include <com/sun/star/lang/IllegalArgumentException.hpp>
#include <com/sun/star/rdf/XMetadatable.hpp>
#include <com/sun/star/rdf/XDocumentMetadataAccess.hpp>
#include <com/sun/star/container/XChild.hpp>

#include <unotextrange.hxx>
#include <comphelper/string.hxx>
#include <comphelper/processfactory.hxx>
#include <i18nlangtag/languagetag.hxx>
#include <vcl/settings.hxx>
#include <inspectorproperties.hrc>
#include <strings.hrc>
#include <rdfhelper.hxx>
#include <unotxdoc.hxx>
#include <unobookmark.hxx>
#include <unostyle.hxx>
#include <unoxstyle.hxx>

namespace sw::sidebar
{
static void UpdateTree(SwDocShell& rDocSh, const SwEditShell&
rEditSh,
                       std::vector<svx::sidebar::TreeNode>& aStore,
sal_Int32& rParIdx);
```