Technical Note PT04
Bugs In MacApp? Yes, But I Love It!

CONTENTS

This Technical Note describes the latest information about bugs or unexpected "features" in MacApp. Where possible, solutions and fixes are noted. DTS intends this Note to be a complete list of all known bugs in MacApp and will update it as old bugs are fixed or new ones appear. If you have encountered a bug or unexpected feature which is not described here, be sure to let DTS know. Specific code examples and suggested fixes are useful.

This version of the Note reflects the state of MacApp 2.0.1. The latest version of this Note can always be found on AppleLink in the Developer Services Bulletin Board.

[Aug 01 1990]

Introduction

The MacApp Management would like to note that MacApp is a high velocity ride with many twists and turns (all alike). Please keep your hands inside at all times.

There are 107,744 lines of Object Pascal, C++, Assembly, and Rez code that go into the MacApp Library and Build system. As such, it is inevitable that a few bugs creep in. The purpose of this Note is to inform you of these bugs, not to scare you away from MacApp. There are dozens of commercially available programs that lead normal everyday lives which are built on top of MacApp as it stands today. Most of the bugs listed here do not show up in regular use (at least, they don't in our test programs), so they may not affect you. If they do, you can use the fixes or solutions identified here ("Fixes" are intended to be applied directly to the MacApp source, while "solutions" identify techniques to override or avoid a method the problem).

Back to top

MacApp.Lib Bugs

TApplication

1. When being suspended in MultiFinder, MacApp commits command objects which affect the clipboard, rather than checking if the scrap has changed when switching back in.

Solution:

Not yet determined. This is an area of serious consideration for the next version of MacApp.

2. MacApp should hide the clipboard window on a suspend event and redisplay it on a resume event.

Solution:

Override the `TApplication` methods `AboutToLoseControl` and `RegainControl`. `AboutToLoseControl` should remember whether or not the clipboard window is currently open and call `gClipWindow.Close` if it is. `RegainControl` should look at the state of the clipboard window saved by `AboutToLoseControl`, and call `gClipWindow.Open` if the window needs to be reshown.

3. There are problems with the value of the `mouseDidMove` parameter to `TCommand` when called by `TApplication.TrackMouse`. When the `TrackPhase` is `trackPress`, `TCommand.TrackMouse` is called with `mouseDidMove` set to `TRUE` even though the mouse hasn't had a chance to move. When the `TrackPhase` is `trackMove`, `mouseDidMove` is `FALSE` whenever the mouse moves back inside the hysteresis range. When the `TrackPhase` is `trackRelease`, `mouseDidMove` is `TRUE` even if the mouse never moved.

Fix:

In `TApplication.TrackMouse` (file UMacApp.TApplication.p):

* The first call to `TrackOnce` should read:

* The assignment of `didMove` should read:

```
    didMove := movedOnce &
    (NOT EqualVPt(previousPoint, theMouse)).
```

* The last call to `TrackOnce` should read:

```
    TrackOnce(trackRelease, didMove);
```

Once those changes have been applied, the parts of MacApp that assume `mouseDidMove = TRUE` when `aTrackPhase = TrackPress` need to be updated. In the methods `TCellSelectCommand.TrackMouse` and `TRCSelectCommand.TrackMouse` (file UGridView.inc1.p), replace:

```
    IF mouseDidMove THEN
```

With:

```
    IF mouseDidMove | (aTrackPhase = TrackPress) THEN
```

You should also make similar changes to your application's source, if applicable. For instance, in UCalc.inc1.p, `TColumnSelector.TrackMouse` and `TRowSelector.TrackMouse` need to check for `aTrackPhase = TrackPress`.

4. With these changes, it is possible to experience some feedback problems. For example, when resizing the column widths in a spreadsheet, `Calc` draws the initial vertical line, waits until the mouse moved outside the hysteresis range, and then, before drawing the vertical line in its new location, erases the old vertical line in the wrong

place. This leaves two vertical lines on the screen as garbage.

Fix:

In UMacApp.TApplication.p, replace the fourth occurrence of:

```
    previousPoint := theMouse;
```

With:

```
    IF didMove THEN
        previousPoint := theMouse;
```

5. The solution that previously occupied this spot caused more problems that it fixed. We removed it until we can get our act together.

TCommand (including subclasses)

1. Fixed in MacApp 2.0.1.
2. If a failure occurs in `TDocument.Revert`, `TRevertDocCommand.DoIt` tries to show the reverted document. This is the correct thing to do if the user canceled out of the revert if a silent failure is signaled (this could happen in `DiskFileChanged`). However if a real error occurred, you cannot leave the document open; you definitely must close it. Otherwise the application may bomb in the next operation involving the document (e.g., the next screen refresh).

We have to distinguish three classes of errors:

1. the user canceled out of the operation in `CheckDiskFile`,
2. a real error was discovered in `CheckDiskFile`,
3. a real error occurred during rebuilding the document in `DoInitialState` or `ReadFromFile`.

In the first and second cases, the memory-resident version of the document has not been changed when you reach `HdlRevertCmd`. In the third case, the document may be severely damaged. Therefore, in the first two cases there is no need to call `ShowReverted` (it doesn't hurt either), while in the third case you **must** close the document.

Case one is easy to recognize (`error = 0`), but for the second and third cases, `error <> 0`. To distinguish between them, you can pull a trick: you know that the Revert menu item is only enabled if `fDocument.fChangeCount` is greater than zero. Therefore, you move `SetChangeCount(0)` in `TDocument.Revert` before any operation that can clobber the document (i.e., before the call to `FreeData`). This way, you can distinguish between the second and third cases in `HdlRevertCmd` by checking `fChangeCount`.

Fix:

Change the failure handling procedure in `TRevertDocCommand.DoIt` (file UMacApp.TDocument.p) to:

```
PROCEDURE HdlRevertCmd(error: OSErr; message: LONGINT);
BEGIN
        {Check whether the document has already been clobbered }
    IF fChangedDocument.GetChangeCount = 0 THEN
        fChangedDocument.Close  {remove the debris
                                 left by fChangedDocument}
END;
```

In `TDocument.Revert`, move the line

```
    SetChangeCount(0);
```

before the line

```
    FreeData;
```

3. It is potentially problematic having Page Setup as an undoable command, since the view and printer driver context can change. An example of this is shown with the following steps:

1. Launch any MacApp application.
2. Access the Page Setup dialog box from the File menu.
3. Take notice of which printer driver is currently being used and make a change to the dialog box (i.e., switch to "landscape" printing), click on the OK button.
4. Access the Chooser desk accessory and change to a different printer driver.
5. Now select Redo Page Setup Changes from the Edit menu, then select Undo Page Setup Changes.
6. Open the Page Setup dialog box from the File menu and notice that the "landscape" printing icon is no longer highlighted.
7. Although the Page Setup dialog box is unaffected by Undo and Redo, the document itself is affected, as it prints out in landscape mode, while the Page Setup dialog box shows it is in non-landscape mode.

Solution:

Apple does not yet have a complete solution to this. If it bothers you, you could modify `IPrintStyleChangeCommand` to make page setup non-undoable.

TControl

1. MacApp's subclasses of `TControl` (defined in the file UDialog.inc1.p) don't pass on their `itsDocument` parameter to the `INHERITED IRes` method. This causes the `fDocument` field to get initialized with `NIL` rather than the `TDocument` reference.

Solution:

You can override the `IRes` method of your own controls to do an `INHERITED IRes` and then set the `fDocument` field to `itsDocument`:

```
PROCEDURE TMyButton.IRes(itsDocument: TDocument;
    itsSuperView: TView;
    VAR itsParams: Ptr); OVERRIDE;

BEGIN
    INHERITED IRes(itsDocument,
                   itsSuperView, itsParams);
    fDocument := itsDocument;
```

Then register your class in your `IYourApplication` method so that all Button references in your `'view'` resources result in `TMyButton`s being created, rather than `TButton`s:

```
    RegisterStdType('TMyButton', kStdButton);
```

However, this solution does not work if you depend on these views appearing in the document's `fViewList`.

Fix:

Replace the calls to `INHERITED IRes` in the `IRes` methods of subclasses of `TControl`:

```
    INHERITED IRes(NIL, itsSuperView, itsParams);
```

With:

```
INHERITED IRes(itsDocument, itsSuperView, itsParams);
```

2. Printing disabled controls, especially buttons, results in a gray pattern being printed over the control. This is not a bug in MacApp, but rather a limitation of the LaserWriter. The LaserWriter driver doesn't respect all QuickDraw transfer modes, including the one used to draw the grey text.

Solution:

Not yet determined. It may involve imaging the button into an off-screen bitmap, and then copying it to its destination.

TCtlMgr

1. Fixed in MacApp 2.0.1.
2. Fixed in MacApp 2.0.1.

TDeskScrapView

1. Fixed in MacApp 2.0.1.

TDialogView

1. `TDialogView` calls `DoChoice` on a disabled button as the result of a key press. If one disables the default button and presses Return, for example, the button's `DoChoice` method still gets called.

Fix:

The following lines of code appear in `TDialogView.DoCommandKey` and in `TDialogView.DoKeyCommand` (file UDialog.inc1.p):

```
IF cancelView.IsViewEnabled THEN
    TControl(cancelView).Flash;
TControl(cancelView).DoChoice(cancelView,
        TControl(cancelView).fDefChoice);
```

Replace them with:

```
IF cancelView.IsViewEnabled THEN BEGIN
    TControl(cancelView).Flash;
    TControl(cancelView).DoChoice(cancelView,
        TControl(cancelView).fDefChoice);
END;
```

Additionally, in `TDialogView.DoKeyCommand`, replace:

```
IF defaultView.IsViewEnabled THEN
    TControl(defaultView).Flash;
TControl(defaultView).DoChoice(defaultView,
        TControl(defaultView).fDefChoice);
```

Replace them

With:

```
    IF defaultView.IsViewEnabled THEN BEGIN
        TControl(defaultView).Flash;
        TControl(defaultView).DoChoice(defaultView,
            TControl(defaultView).fDefChoice);
    END;
```

Solution:

You can do this as an `OVERRIDE` if you hesitate to change MacApp.

TDocument

1. `TDocument.Save` fails if you lock a file after opening it with read and write access and then try to save. The file is closed and `fDataRefNum` and `fRsrcRefNum` contain their old (and now invalid) values.

Solution:

Not yet determined.

1. If `GetFileInfo` returns a result other than `noErr`, `TDocument.DiskFileChanged` maps it to `errFileChanged`, because there is no check for (`err = noErr`) in the `ELSE IF` branch. The resulting alert is misleading, as the file may also have been renamed, deleted, or the file server may have gone offline.

Fix:

The error checking code in `TDocument.DiskFileChanged` (file UMacApp.TDocument.p) should look like:

```
    err := GetFileInfo(fTitle^^, fVolRefNum, pb);
    ...
    IF (err = noErr) THEN
        IF checkType
        & (pb.ioFlFndrInfo.fdType <> fFileType) THEN
            err := errFTypeChanged
        ELSE IF pb.ioFlMdDat <> fModDate THEN
            err := errFileChanged;
        DiskFileChanged := err;
```

3. It is not possible to use the Pascal built-in filing function `Close` from within a `TDocument` method because the Object Pascal scoping rules always associate the name `Close` with `TDocument.Close`.

Solution:

It is likely that Apple will change the name in the future. After all, there are three distinct objects that implement a `Close` method, none of which have any relation to another; something like that needs to be cleaned up. In the meantime, you could make a global routine `MyClose` that would be a wrapper for the `Close` routine.

TEditText

1. If the first or only `TEditText` in a dialog has auto-wrap turned on and is not initially selected, tabbing to it after opening the window selects it, but the selection is not visible until the window is refreshed. This does not occur if auto-wrap is turned off for that `TEditText`.

Solution:

Not yet determined.

2. `TEditText` items in `TDialogViews` no longer get the first crack at events, as they used to in MacApp 2.0b9. The event handler chain is now `gTarget` -> `TDialogTEView` -> `TScroller` -> `TEditText` ->

TDialogView -> etc., so all events in which a subclass of TEditText might be interested are caught by TDialogTEView.

Solution:

Create your own subclass of TDialogTEView that handles the interesting characters by overriding TDialogView.MakeTEView and returning your own subclass of TDialogTEView.

Fix:

A clean fix to this problem might be to rearrange the event handler chain to gTarget -> TEditText -> TDialogTEView -> TScroller -> TDialogView -> etc. However, this approach is more work for TDialogTEView.InstallEditText, and the effects of rearranging the target chain are currently untested.

TEvtHandler

1. TEvtHandler.DoCreateViews now calls TView.AdjustSize on the root view it just created. This change was made to give views an early chance to make sure they are correctly sized. However, the change can cause your application to break if you override routines that perform AdjustSize calls (like TView.CalcMinSize) and those routines rely on information that is not initialized until after DoCreateViews returns to TYourDocument.DoMakeViews.

Solution:

Simply be aware of this change. If you initialize fields of your views in your DoMakeViews method, then overrides of methods such as TView.Resize, TView.ComputeSize, and TView.CalcMinSize should take into account that these fields may not yet be initialized. At the very least, TYourView.IRes should set these fields to NIL.

TGridView

1. Attempting to select a TGridViewcell for which CanSelectCell returns FALSE causes the current selection to be deselected.

Solution:

Override TGridView.DoMouseCommand to call IdentifyPoint. If a valid cell is returned, call CanSelectCell. If it returns TRUE, call INHERITED DoMouseCommand. This inhibits all tracking if the user initially clicks in a disabled cell.

Fix:

Replace the following line in TCellSelectCommand.TrackMouse (file UGridView.inc1.p):

```
    IF LONGINT(clickedCell) <> LONGINT(fPrevCell)
```

With:

```
    IF (LONGINT(clickedCell) <> LONGINT(fPrevCell))
    & fGridView.CanSelectCell(clickedCell)
```

2. Fixed in MacApp 2.0.1.
3. TGridView.DrawCell is called with the clip region set wide open, which allows any override of DrawCell to draw anywhere within the TGridView. It is likely that this is not desired, and the responsibility for clipping could be added to TGridView.

Fix:

In TGridView.DrawRangeOfCells (file UGridView.inc1.p), add a local RgnHandle called oldClip. Initialize oldClip with the following lines at the beginning of the method:

```
    oldClip := MakeNewRgn;
    GetClip(oldClip);

    Next, clip to the current cell by adding the following
    lines before the line that says aCell.h := i:

    {$IFC qDebug}
        UseTempRgn('TGridView.DrawRangeOfCells');
    {$ENDC}
    RectRgn(gTempRgn, aQDRect);
    SectRgn(oldClip, gTempRgn, gTempRgn);
    SetClip(gTempRgn);
    {$IFC qDebug}
        DoneWithTempRgn;
    {$ENDC}
```

Finally, add the following lines at the end of the method:

```
    SetClip(oldClip);
    DisposeRgn(oldClip);
```

TIcon

1. Fixed in MacApp 2.0.1.

TList

1. If you have a TList subclass with a String instance variable, it is not possible to use the Pascal string built-in function Delete on it because the Object Pascal scoping rules always associate the name Delete with TList.Delete.

Solution:

Apple will change the name in the future. In the meantime, you could make a global routine MyDelete that would be a wrapper for the string Delete routine.

TNumberText

1. When the length of the text in a TNumberText instance is 0, GetValue returns 0, and Validate returns kValidValue. The value is not checked against fMinimum or fMaximum, so your application may be fed with a value it is not prepared to handle.

Fix:

Ideas for solutions or fixes are outlined in the comment in TNumberText.Validate (file UDialog.inc1.p).

TPopup

1. Fixed in MacApp 2.0.1.
2. Fixed in MacApp 2.0.1.
3. Fixed in MacApp 2.0.1.
4. TPopup no longer calls DoChoice if the same item is reselected.

Fix:

In TPopup.DoMouseCommand (file UDialog.inc1.p) is the following line:

```
    IF (HiWord(result) <> 0)
    & (newChoice <> fCurrentItem) THEN
```

Remove the "`& (newChoice <> fCurrentItem)`" part.

`TPopup.SetCurrentItem` neither restores the port colors correctly nor uses the right rectangle to obtain the menu colors for the popup box.

Fix:

In `TPopup.SetCurrentItem` (file UDialog.inc1.p), declare the following two new local variables:

```
    newFColor: RGBColor;
    newBkColor: RGBColor;
```

Then, replace:

```
    IF redraw & Focus & IsVisible THEN BEGIN
        GetQDExtent(menuRect);
        GetMenuColors(menuRect, fMenuID,
                item, newFColor, newBkColor);
        SetIfColor(newFColor); SetIfBkColor(newBkColor);
        DrawPopupBox(menuRect);
        END;
```

With:

```
    IF redraw & Focus & IsVisible THEN BEGIN
        GetIfColor(oldFColor); GetIfBkColor(oldBkColor);
        CalcMenuRect(menuRect);

        GetMenuColors(menuRect, fMenuID, fCurrentItem,
            newFColor, newBkColor);
        SetIfColor(newFColor); SetIfBkColor(newBkColor);
        DrawPopupBox(menuRect);
        { Reset colors to their original state }
        SetIfColor(oldFColor); SetIfBkColor(oldBkColor);
    END;
```

TScroller

1. `TScroller.RevealRect` doesn't call `INHERITED RevealRect`. This has implications in situations where you have nested scrollers. If, for example, you run DemoDialogs, select the first menu item, press the Tab key, then begin typing, the `TEditText` item you are modifying is not scrolled into view. This is because while your selection is revealed within the context of the `TEditText`, the `TEditText` item itself is not scrolled into view.

Fix:

Add an `INHERITED RevealRect` call to `TScroller.RevealRect` (file UMacApp.TScroller.p):

```
    PROCEDURE TScroller.RevealRect(...);
    BEGIN
        ...
        ScrollBy(delta.h, delta.v, redraw);
        OffsetVRect(rectToReveal, -fTranslation.h,
                -fTranslation.v);
        INHERITED RevealRect(rectToReveal,
                minToSee, redraw); { add this call }
    END;
```

TStdPrintHandler

1. An extra blank page is printed if `TStdPrintHandler.fFixedSizePages = FALSE` and `fSizeDeterminer = sizeFillPages`. This is because `TView.ComputeSize` computes the view's size as a multiple of the printable page size for `sizeFillPages`, ignoring that the view need not use the full size of each page.

Solution:

Always set both Boolean components of `fFixedSizePages` to `TRUE`. These are initialized from the last two parameters you pass to `IStdPrintHandler`.

Solution:

Use `fSizeDeterminer = sizeVariable`.

2. Simply using the naked `DIV` operator for scaling `theMargins` in `TStdPrintHandler.CheckPrinter` introduces rounding errors. These errors may be disturbing if you need precise control over the margins used for printing.

Fix:

Insert the following local procedure in `TStdPrintHandler.CheckPrinter` (file UPrinting.inc1.p):

```
FUNCTION ScaleInteger(theValue, theMultiplier,
        theDivisor: Integer): Integer;
VAR
    intermediate: Longint;
BEGIN
    intermediate := IntMultiply(theValue, theMultiplier);
    IF intermediate >= 0 THEN
        intermediate := intermediate + ABS(theDivisor) div 2
    ELSE
        intermediate := intermediate - ABS(theDivisor) div 2;
    ScaleInteger := intermediate DIV theDivisor;
END;
```

In the implementation of `TStdPrintHandler.CheckPrinter`, replace the lines:

```
SetRect(theMargins,
    IntMultiply(theMargins.left, h) DIV oldMarginRes.h,
    IntMultiply(theMargins.top, v) DIV oldMarginRes.v,
    IntMultiply(theMargins.right, h) DIV oldMarginRes.h,
    IntMultiply(theMargins.bottom, v) DIV oldMarginRes.v);
```

With:

```
SetRect(theMargins,
    ScaleInteger(theMargins.left, fMarginRes.h, oldMarginRes.h),
    ScaleInteger(theMargins.top, fMarginRes.v, oldMarginRes.v),
    ScaleInteger(theMargins.right, fMarginRes.h, oldMarginRes.h),
    ScaleInteger(theMargins.bottom, fMarginRes.v, oldMarginRes.v));
```

3. `TStdPrintHandler.CheckPrinter` calculates `fMarginRes` incorrectly for scaled printing. It does not take into account any scaling factors imposed by the user in the `_PrStlDialog` dialog box.

Fix:

Use the following until Apple can come up with something better. Note that this fix relies on the undocumented fields `prStl.iPageV` and `prStl.iPageH`. Additionally, it implements a dubious technique that gets around the assumption that any printer supporting landscape printing also supports `_PrGeneral`, which is not always the case; therefore, this fix is considered temporary. You should already have applied the fix to the second bug in the `TStdPrintHandler` section.

Insert the following local procedure after `ScaleInteger` in `TStdPrintHandler.CheckPrinter` (file UPrinting.inc1.p):

```
PROCEDURE AdjustMarginRes;

PROCEDURE DoAdjustMarginRes;

VAR
    getRotationBlock: TGetRotnBlk;

    BEGIN
        WITH getRotationBlock DO BEGIN
            iOpCode := getRotnOp;
            lReserved := 0;
            hPrint := THPrint(fHPrint);
            bXtra := 0;
        END;
        PrGeneral(@getRotationBlock);
        IF (PrError <> noErr)
        | (getRotationBlock.iError <> noErr) THEN BEGIN
            WITH fPageAreas.thePaper DO
                getRotationBlock.fLandscape := right - left >
                bottom - top;
                PrSetError(noErr); { clear print error - Printing
                    Manager won't do it }
            END;
            WITH
            fPageAreas.thePaper,
            fMarginRes,
            THPrint(fHPrint)^^
            DO BEGIN
                {$PUSH} {$H-} { shut up, dumb compiler! }
                    { The undocumented fields prStl.iPageH &
                    prStl.iPageH seem unaffected by rotation, so
                    we have to rotate them }
                IF getRotationBlock.fLandscape THEN BEGIN
                    fMarginRes.h := ScaleInteger(iPrPgFract,
                        right - left,
                        prStl.iPageV);
                    fMarginRes.v := ScaleInteger(iPrPgFract,
                        bottom - top,
                        prStl.iPageH);
                END ELSE BEGIN
                    fMarginRes.h := ScaleInteger(iPrPgFract,
                            right - left,
                            prSt1.iPageH);
                        fMarginRes.v := ScaleInteger(iPrPgFract,
                            bottom - top,
                            prStl.iPageV);
                    END;
                    {$POP}
                END; { WITH }
            END;
        END;

    BEGIN
        DoInMacPrint(DoAdjustMarginRes);
    END;
```

In `TStdPrintHandler.CheckPrinter`, replace everything after `fPageAreas.thePaper := rPaper` and up to, but not including, the statement `fPrinterDev := iDev;` with the following lines:

```
    AdjustMarginRes;
    WITH prInfo DO BEGIN
```

Next, you have to take into account the fact that `CheckPrinter` can open and close the print driver. This can be bad when you are in the middle of printing because you are closing a driver that needs to stay open.

An ideal solution would include some sort of mechanism to keep track of whether the printer was already open when you open it again, or maintain a reference count on the number of nested calls to `DoMacInPrint`. However, for now, we can put in a simple check to avoid the one place where nesting occurs. First, add the following line to the beginning of `CheckPrinter`:

```
    if fPPrPort = nil then begin
```

Then, insert the following statement just before the end of `CheckPrinter`:

```
end;
```

Finally, you need to set `fPPrPort` to `NIL` when not printing. In `TStdPrintHandler.OneSubJob`, replace:

```
    PrCloseDoc(fPPrPort);{ This will close the port! }
```

With:

```
    PrCloseDoc(fPPrPort); { This will close the port! }
    fPPrPort := NIL;      { Lose the reference }
```

4. Fixed in MacApp 2.0.1.

TTEView

1. Fixed in MacApp 2.0.1.
2. Fixed in MacApp 2.0.1.
3. In a `TTEView` with non-zero bottom inset, only part of the second is displayed when text wraps to a new line.

Solution:

Always have a bottom inset of zero.

Fix:

Modify `TTEView.StuffTERects` (file UTEView.TTEView.p) to give the `TERecord` a bottomless `destRect` and `viewRect`. Replace:

With:

```
   BEGIN
       right := MAX(right, left + aFontInfo.widMax);
       bottom := MAXINT;              { give us a bottomless destrect }
```

4. `TTEView.SynchView` only updates the text if the line heights have changed. It calls `CalcRealHeight`, and if it has not changed, it doesn't do anything. If a program modifies the text directly, it must call `ForceRedraw`. For instance, say that you have a class `TMyTEView` has the following routine:

```
   PROCEDURE TMyTEView.TweekText;
   VAR
       myText : TextHandle;
   BEGIN
       myText := ExtractText;
          { do some munging of the text (e.g., search and replace) }
          { make TTEView display changed text }
       RecalcText;
       SynchView(kRedraw);
          { !!! We shouldn't have to force a complete redraw !!! }
       ForceRedraw;
```

Solution:

Call `ForceRedraw` as above, until Apple has a solution. It could be that removing the `fLastHeight <> theHeight` comparison in `SynchView` does the trick, but it may also result in unnecessary updates and flashing.

5. You may find it useful to use a `TTEView` as a Read-Only view. To do this, disable the view and set `fAcceptsChanges` to `FALSE`. However, with these settings, the Select All menu item is still enabled.

Fix:

In `TTEView.DoSetupMenus` (file UTEView.TTEView.p), replace the line:

```
   Enable(cSelectAll, (fHTE^^.teLength > 0));
```

With:

```
   Enable(cSelectAll, IsViewEnabled & (fHTE^^.teLength > 0));
```

6. Fixed in MacApp 2.0.1.
7. `TTEView` sometimes leaves the image of a caret behind as it scrolls. This is usually experienced when scrolling a

left-justified TEditText item for the first time.

Fix:

Modify TTEView.StuffTERects (file UTEView.TTEView.p) to give the TERecord a little more room on the left or right. Replace:

```
    right := MAX(right, left + aFontInfo.widMax);
```

With:

```
    IF (fSizeDeterminer[h] = sizeVariable) &
        NOT fStyleType & NOT fAutoWrap THEN
    BEGIN
        CASE GetActualJustification(fJustification) OF
            teJustLeft, teForceLeft:
                right := right + aFontInfo.widMax;
                teJustRight: left := left - aFontInfo.widMax;
                teJustCenter: right :=
                    MAX(right, left + aFontInfo.widMax);
            END;
        END
    ELSE
    right := MAX(right, left + aFontInfo.widMax);
```

Note:

You should make this modification only after making the changes described in TTEView #3.

Because the size of viewRect and destRect of the TERecord are now dependant on the justification being used, TTEView.SetJustification needs to call StuffTERects. Add a local Rect variable called r. Then, after the line that says fJustification := newJust, add:

```
    r.topLeft := fInset.topLeft;
    r.right := fSize.h - fInset.right;
    r.bottom := fSize.v - fInset.bottom;
    StuffTERects(r);
```

8. Fixed in MacApp 2.0.1.
9. Scrolling a TTEView quickly via cursor keys or by pasting new text does not immediately update the newly revealed regions, which can lead to unpleasant cosmetic artifacts.

Fix:

In TTEView.ScrollSelectionIntoView (file UTTEView.TTEView.p), add a call to Update after the call to RevealRect.

TView

1. TEvtHandler.DoCreateViews doesn't work right if you build your view tree in the "wrong" order (i.e., breadth-first order). If you declare them as a hierarchy of levels, like this:

```
        ViewA
        ViewB
            SubViewA-1
            SubViewA-2
            SubViewB-1
                SubViewA-1-1
```

DoCreateViews cannot find SubViewA-1 when creating SubViewA-1-1.

Solution:

Declare your views in this order (walking the tree) in the Rez file:

```
                       ViewA
                           SubViewA-1
                               SubViewA-1-1
                               SubViewA-1-2
                           SubViewA-2
                       ViewB
```

2. `TView.Focus` does not always work correctly in long coordinate situations. When dealing with view systems that stay entirely within QuickDraw's 16-bit coordinate plane, focusing works correctly. However, when dealing with larger view systems, `TView.Focus` does not always correctly switch over to MacApp's 32-bit coordinate system.

Fix:

In `TView.Focus` (file UMacApp.TView.p), replace:

```
     IF fSize.vh[vhs] > kMaxCoord THEN
```

With:

```
     IF (fSize.vh[vhs] > kMaxCoord)
     | (ABS(fLocation.vh[vhs]) > kMaxCoord)
```

Daring Fix:

You can try taking out short coordinate focussing altogether. This solution has not yet been fully tested, so there may be some side effects of which Apple is unaware. In `TView.Focus` (file UMacApp.TView.p), replace:

```
     FOR vhs := v TO h DO
         IF fSize.vh[vhs] > kMaxCoord THEN BEGIN
             tempLongOffset := gLongOffset.vh[vhs] - fLocation.vh[vhs];
             relOrigin.vh[vhs] := tempLongOffset MOD kMaxOriginFixup;
             gLongOffset.vh[vhs] := tempLongOffset - relOrigin.vh[vhs];
         END ELSE BEGIN
             relOrigin.vh[vhs] := gLongOffset.vh[vhs] -
                       fLocation.vh[vhs];
             gLongOffset.vh[vhs] := 0;
         END;
```

With:

```
    FOR vhs := v TO h DO BEGIN
        tempLongOffset := gLongOffset.vh[vhs] - fLocation.vh[vhs];
        relOrigin.vh[vhs] := tempLongOffset MOD kMaxOriginFixup;
        gLongOffset.vh[vhs] := tempLongOffset - relOrigin.vh[vhs];
    END;
```

3. TView calls _InvalRect and _ValidRect directly. These are Window Manager calls which assume that the current port (thePort) is a window. If thePort is not a window and these calls are made, all sorts of nasty fireworks happen. This bug only appears when a TView is placed in something other than a TWindow and the view calls TView.InvalidRect, TView.InvalidRect, or TView.ValidVRect.

For example, when using a TGridView as a subview of a TMenu, IGridView results in a call to TView.InvalidRect. Since TMenu carries its own GrafPort, the InvalRect on the TMenu GrafPort fails.

Fix:

In the file UMacApp.TView.p, modify the methods TView.InvalidRect, TView.InvalidVRect, and TView.ValidVRect to UMacApp.TView.p, as shown.

```
                {$S MAViewRes}

                PROCEDURE TView.InvalidRect(r: Rect);

                    BEGIN
                    IF IsShown & Focus THEN
                        BEGIN
                        VisibleRect(r);
                        IF NOT EmptyRect(r) THEN
                            InvalidateFocusedRect(r);
                        END;
                    END;

                {$S MAViewRes}

                PROCEDURE TView.InvalidVRect(viewRect: VRect);

                    VAR
                        r:                  Rect;

                    BEGIN
                    IF IsShown & Focus THEN
                        BEGIN
                        ViewToQDRect(viewRect, r);
                        VisibleRect(r);
                        IF NOT EmptyRect(r) THEN
                            InvalidateFocusedRect(r);
                        END;
                    END;

                {$S MAViewRes}

                PROCEDURE TView.ValidVRect(viewRect: VRect);

                    VAR
                        r:                  Rect;

                    BEGIN
                    IF IsShown & Focus THEN
                        BEGIN
                        ViewToQDRect(viewRect, r);
                    VisibleRect(r);
                    IF NOT EmptyRect(r) THEN
                        ValidateFocusedRect(r);
                    END;
```

Next, in UMacApp.TView.p, add `TView.InvalidateFocusedRect` and `TView.ValidateFocusedRect`. These are the routines that forward up the view hierarchy until finding a `TWindow`. You also take this opportunity to add `TView.ValidateRect`--a QuickDraw version of `TView.ValidVRect`--for completeness.

```
{$S MAViewRes}
PROCEDURE TView.InvalidateFocusedRect(r: Rect);

    BEGIN
    IF fSuperView <> NIL THEN
        fSuperView.InvalidateFocusedRect(r);
    END;

{$S MAViewRes}

PROCEDURE TView.ValidateRect(r: Rect);

    BEGIN
    IF IsShown & Focus THEN
        BEGIN
        VisibleRect(r);
        IF NOT EmptyRect(r) THEN
            ValidateFocusedRect(r);
        END;
    END;

{$S MAViewRes}

PROCEDURE TView.ValidateFocusedRect(r: Rect);

    BEGIN
    IF fSuperView <> NIL THEN
        fSuperView.ValidateFocusedRect(r);
    END;
```

In `TWindow`, you then override `TView.InvalidateFocusedRect` and `TView.ValidateFocusedRect` in UMacApp.TWindow.p to call the Window Manager routines.

```
{$S MAWindowRes}

PROCEDURE TWindow.InvalidateFocusedRect(r: Rect); OVERRIDE;

    BEGIN
    InvalRect(r);       { Call the ToolBox routine. }
    END;

{$S MAWindowRes}

PROCEDURE TWindow.ValidateFocusedRect(r: Rect); OVERRIDE;

    BEGIN
    ValidRect(r);       { Call the ToolBox routine. }
    END;
```

Finally, in UMacApp.p add the following declarations for the new routines:

```
          PROCEDURE TView.InvalidateFocusedRect(r: Rect);
               PROCEDURE TView.ValidateRect(r: Rect);
               PROCEDURE TView.ValidateFocusedRect(r: Rect);
               PROCEDURE TWindow.InvalidateFocusedRect(r: Rect); OVERRIDE;
               PROCEDURE TWindow.ValidateFocusedRect(r: Rect); OVERRIDE;
```

With those changes in place, all calls to `_ValidRect` in the rest of MacApp should now be calls to `TView.ValidateRect`. The only methods this affects are `TSScrollbar.Activate` and `TDeskScrapView.Draw`.

4. When the focus is invalidated during printing, MacApp is not able to restore it properly. For example, you could move a subview during printing because you don't know where it's supposed to go until you need it. When MacApp tries to refocus, the clip region is set to an empty region, and nothing gets printed from that point on.

Solution:

Not yet determined. It's not clear whether MacApp should handle such odd things as moving subviews during printing.

5. Fixed in MacApp 2.0.1.
6. When the call to `FocusOnSuperView` in `TView.Focus` returns `FALSE`, `Focus` tries to invalidate all focus information with the statements:

```
        ClipRect(gZeroRect);
```

The problem with these statements is that `InvalidateFocus` sets `gFocusedView` to `NIL` only if the focus is on some view in the subview hierarchy of `SELF`. Thus, if the focus is on some completely unrelated view in the same port, the clip region of the port of that view is set to `gZeroRect`, but `gFocusedView` is unaffected. If the `Focus` method of `gFocusedView` is called later, its call to `IsFocused` returns `TRUE`, but drawing does not work because the clip region is empty.

Fix:

In TView.Focus (file UMacApp.TView.p), remove the call to `_ClipRect`. It might also be a good idea to do the same in `TWindow.Focus` since the `_ClipRect` call is being made on an essentially random port.

TWindow

1. Fixed in MacApp 2.0.1.
2. Fixed in MacApp 2.0.1.
3. `TWindow.Center` can sometimes move large windows with title bars under the menu bar.

Fix:

In `TWindow.Center` (file UMacApp.TWindow.p), replace the following lines:

```
               IF forDialog THEN
                   { Put it in the top third of the screen }
                   top := ((screenSize.v - contentSize.v
                       + fContRgnInset.v) DIV 3) + 20
               ELSE
                   top := ((screenSize.v - contentSize.v
```

With:

```
               IF forDialog THEN
                   { Put it in the top third of the screen }
                   top := ((screenSize.v - windowsize.v) DIV 3)
                       { calculate spare area }
                         + gMBarHeight
                         { add menu bar }
                       { calculate the right offset of
                       content inside the window }
                         + ((windowsize.v - contentsize.v
                         + fContRgnInset.v) DIV 2)
               ELSE
                   top := ((screenSize.v - windowsize.v) DIV 2)
                           { calculate spare area }
                         + gMBarHeight
                             { add menu bar }
                               { calculate the right offset of content
                               inside the window }
                         + ((windowsize.v - contentsize.v
                         + fContRgnInset.v) DIV 2);
```

Assorted Problems Due to a New TView.Focus Definition

The next items address a class of problems related to the fact that TView.Focus is defined to return TRUE if a drawing environment can be obtained (e.g., a GrafPort). Thus it now returns TRUE even if the view is invisible. The various problems are: 1) invisible controls in dialog boxes accepting mouse-down events and doing things; 2) children of invisible controls being asked to draw or handle a mouse-down event; 3) scroll bars of hidden scrollers appearing; 4) hidden scroll bars of scrollers not appearing; and 5) calls to IsShown for an arbitrary view returning incorrect results.

1. TView.IsShown contains the following line:

It turns out that the answer to this question is yes. There are many problems that occur in MacApp that are caused by views who are themselves not hidden, but whose superviews are. For instance, it is possible for a click to be registered on a view whose superview is hidden. This can cause the previously hidden control to appear.

Fix:

In TView.IsShown (file UMacApp.TView.p), replace the line above with the following:

```
    IF fSuperView <> NIL THEN
        IsShown := fShown &
        fSuperView.IsShown { By definition, a view cannot be
                           shown if its superview isn't.}
    ELSE IsShown := fShown;
```

2. Having TView.IsShown reflect the willingness of all its superviews to be shown causes one problem in MacApp. When a TScroller creates its scroll bars, it sets the fShown field of the TSScrollBar to the result of TScroller.IsShown. However, at the time a scroller creates its scroll bars, the window they are in is invisible. Its IsShown method returns FALSE, which is propagated down to the TScroller, causing TScroller.CreateTemplateScrollBar to initialize TSScrollBar.fShown to FALSE.

Fix:

Cause the TSScrollBar to inherit the fShown field of its TScroller **only**. In TScroller.CreateTemplateScrollBar (file UMacApp.TScroller.p), replace:

```
    anSScrollBar.fShown := IsShown;
```

With:

```
    anSScrollBar.fShown := fShown;
```

3. There is no `TCtlMgr.Show` to control the setting of `fCMgrControl^^.contrlVis`. Neglecting to do so results in certain silly things happening, like an activate event triggering the drawing of your invisible scroll bars.

Fix:

Override `TView.Show` with the following version of `TCtlMgr.Show` (file UMacApp.TControls.p). Don't forget to also update the declaration of `TCtlMgr` in UMacApp.p:

```
PROCEDURE TCtlMgr.Show(state, redraw: BOOLEAN);

    BEGIN
    SetCMgrVisibility(state);
    INHERITED Show(state, redraw);
```

Additionally, `TScrollBar` needs to override `Show` to implement its special appearance when shown in an inactive window. Add the following method to UMacApp.TControls, and add the appropriate declaration to UMacApp.p:

```
PROCEDURE TScrollBar.Show(state, redraw: BOOLEAN);

    VAR
        itsWindow:  TWindow;

    BEGIN
    INHERITED Show(state, redraw);
    itsWindow := GetWindow;
    SetCMgrVisibility(state & (itsWindow <> NIL)
         & itsWindow.fIsActive);
```

4. `TControl.ContainsMouse` needs to call `TCtlMgr.IsShown`. Otherwise, it's possible for those controls to receive mouse clicks.

Fix:

Use the following version of `TControl.ContainsMouse` (file UMacApp.TControls.p):

```
FUNCTION TControl.ContainsMouse(theMouse: VPoint): BOOLEAN; OVERRIDE;
    VAR
                aRect: Rect;
    BEGIN
    IF IsShown THEN
        BEGIN
        ControlArea(aRect);
        ContainsMouse := PtInRect(VPtToPt(theMouse), aRect);
        END
    ELSE
        ContainsMouse := FALSE;
```

5. `TView.Focus` used to return `FALSE` if the view was invisible. It no longer does this, and many routines in MacApp relying on this behavior now need to check this explicitly:

Fix:

The following routines should be modified to check IsShown before calling Focus. Note that the changes to TView.InvalidVRect, TView.InvalidVRect, and TView.ValidRect need not be made if the modifications to the third bug in the TView section have been made.

```
TView.IsViewEnabled  (file UMacApp.TView.p)
     IsViewEnabled := fViewEnabled & IsShown;

TGridView.HighlightCells (file UGridView.inc1.p)
     IF (fromHL <> toHL) & IsShown & Focus THEN

TCtlMgr.WhileFocused (file UMacApp.TControls.p)
TTEView.SynchView (file UTEView.TTEView.p)
     IF redraw & IsShown & Focus THEN

TView.InvalidRect ( see above comment ) (file UMacApp.TView.p )
TView.InvalidVRect ( see above comment ) (file UMacApp.TView.p )
TView.ValidVRect ( see above comment ) (file UMacApp.TView.p )
TGridView.InvalidateSelection (file UGridView.inc1.p)
TScroller.ScrollDraw (file UMacApp.TScroller.p)
     IF IsShown & Focus THEN

TSScrollBar.Activate (file UMacApp.TControls.p)
     add this check before WhileFocused:
```

6. With the changes from bug five in place, a problem appears when a TScroller is resized. The scroller hides its scroll bars, resizes itself, adjusts its scroll bars, and shows them again. AdjustScrollbars potentially asks a scroll bar to invalidate itself. However, at that time, the scroll bar is invisible, thus its contents cannot possibly be wrong, as they have yet to be drawn. It is the scroll bar itself that is wrong, and therefore the contents of its superview (in that rectangle) that must be invalidated.

Fix:

To patch the bug, modify the final few lines of TScroller.Resize (file UMacApp.TScroller.p):

```
FOR vhs := v TO h DO
    IF sBarWasVisible[vhs] THEN
        BEGIN
        fScrollBars[vhs].SetCMgrVisibility(TRUE);
        fScrollBars[vhs].ForceRedraw; { this is new }
        END;
```

This is not a real fix, this is only a patch. The final fix probably requires modification to TView.Locate and TControl.Resize.

7. Fixed in MacApp 2.0.1.
8. Assorted TView.Focus fixes #1 and #5 together have ramifications on TDialogTEView.InstallEditText. Because a view is now considered invisible if any of its superviews are invisible, and a view is now considered disabled if it is invisible, all views are effectively disabled in an invisible window. The effect of this is that InstallEditText disables the floating TDialogTEView and its scroller if called before the window is opened.

Fix:

You can most likely experience the problem when calling TDialogView.SelectEditText before the window is opened. Thus, modify SelectEditText to check if the window is shown or not. If so, call

TDialogView.DoSelectEditText as normal (which eventually calls InstallEditText). If the window is not open, simply set the specified view as the window's target, to be selected when the window is eventually opened. Thus, in TDialogView.SelectEditText (file UDialog.inc1.p), add the following local variable:

Then, replace:

With:

```
IF IsShown THEN
    DoSelectEditText(TEditText(aSubView), selectChars)
ELSE
    BEGIN
    itsWindow := GetWindow;
    IF itsWindow <> NIL THEN
        itsWindow.SetTarget(aSubView)
    ELSE
        ProgramBreak('found no way to select the edit text');
```

Global Routines and Interfaces

1. Fixed in MacApp 2.0.1.
2. Fixed in MacApp 2.0.1.
3. Fixed in MacApp 2.0.1.
4. Fixed in MacApp 2.0.1.
5. Fixed in MacApp 2.0.1. WithApplicationResFileDo needs a failure handler. Since the method's

   normal

   behavior is to preserve the current resource file, in case of a failure it should do the same thing. The problem is that if WithApplicationResFileDo contains a failure handler, it must be moved to another unit; UMacAppUtilities cannot access UFailure without introducing a circular reference.

Fix:

Move WithApplicationResFileDo to the file UMenuSetup.inc1.p and change it to the following:

```
PROCEDURE WithApplicationResFileDo(PROCEDURE DoWithResFile);

    VAR
        fi:         FailInfo;
        oldResFile: INTEGER;

    PROCEDURE HdlFailure(error: OSErr; message: LONGINT);

        BEGIN
        UseResFile(oldResFile);
        END;

    BEGIN
    oldResFile := CurResFile;
    CatchFailures(fi, HdlFailure);
    UseResFile(gApplicationRefNum);
    DoWithResFile;
    Success(fi);
    UseResFile(oldResFile);
```

7. VisibleRect returns the intersection of the specified rectangle along with the bounding boxes of the visRgn and clipRgn. When called during a window update, however, the visRgn can be smaller than expected. This difference can cause VisibleRect to return different sized rectangles when called inside or outside of an update

event.

Fix:

The final fix has not yet been determined; however, you may be able to kludge things by modifying `TWindow.Update` in the file UMacApp.TWindow.p. Just before `_BeginUpdate`, add the following line:

Next, add the following line immediately after both calls to `_EndUpdate`:

Then, in `VisibleRect` (file UMacApp.Globals.p), change:

```
               IF NOT gPrinting THEN
```

To:

```
               IF NOT (gPrinting | gUpdating) THEN
```

Finally, add `gUpdating` to the file UMacApp.p, and initialize it to `FALSE` in `InitUMacApp`. Or you can just live dangerously and take out the `_SectRgn` call altogether.

8. Patching a trap with the routines in `UPatch` can cause a crash under the Finder (when MultiFinder is not present) if that trap is already patched by MacApp, because the `CleanUpMacApp` routine incorrectly restores that trap to point at the MacApp patch, rather than at the original routine.

Solution:

Do not patch traps that MacApp patches (currently: _ExitToShell, _InitCursor, _SetCursor, _SetCCursor, _GetNextEvent, _EventAvail, _StillDown, and _WaitMouseUp).

Fix:

Rewrite `UnpatchTrap` (file UPatch.inc1.p) as follows, so it does the right thing when unpatching traps that have "newer" patches:

```
               PROCEDURE UnpatchTrap(VAR thePatch: TrapPatch);

                  VAR
                      aPatchPtr:          TrapPatchPtr;
                      newerPatchPtr:      TrapPatchPtr;

                  FUNCTION GetPreviousPatchPtr(thePatchPtr: TrapPatchPtr):
                                                  TrapPatchPtr;
                  { Walks the patch list backwards to return the patch record
                    just prior to thePatchPtr^ in the patch list }

                      VAR
                          tempPatchPtr:   TrapPatchPtr;

                      BEGIN
                      tempPatchPtr := pPatchList;
                      WHILE (tempPatchPtr <> NIL)
                      & (tempPatchPtr^.nextPatch <>
                        thePatchPtr) DO
                          tempPatchPtr := tempPatchPtr^.nextPatch;
                      GetPreviousPatchPtr := tempPatchPtr;
                      END;

                  FUNCTION GetNewerPatchPtr: TrapPatchPtr;
                  { returns a newer patch record in the patch list which has
                    the same trapNum as thePatch }
```

```
                        BEGIN
                        aPatchPtr := GetPreviousPatchPtr(@thePatch);
                        WHILE (aPatchPtr <> NIL) & (aPatchPtr^.trapNum <>
                          thePatch.trapNum) DO
                            aPatchPtr := GetPreviousPatchPtr(aPatchPtr);
                        GetNewerPatchPtr := aPatchPtr;
                        END;

                    BEGIN
                    { If this trap has a newer patch than the patch we're removing,
                      then we have to take some extra special precautions. We have
                      to muck with that patch's oldTrapAddr to point to this patch
                      record's oldTrapAddr (for both the patch record and the
                      jumpPtr code). We can pretty well ignore the case of an
                      older patch on this same trap since the trap address in our
                      patch record will be correct. }

                    newerPatchPtr := GetNewerPatchPtr;
                    IF (newerPatchPtr = NIL) THEN
                        WITH thePatch DO
                            NSetTrapAddress(OldTrapAddr, trapNum,
                                            GetTrapType(trapNum))
                    ELSE
                        BEGIN
                        { set up newerPatchPtr patch record so that it points to
                          thePatch's OldTrapAddr }
                        newerPatchPtr^.oldTrapAddr := thePatch.oldTrapAddr;

                        { set up newerPatchPtr^.jmpPtr so that it jumps to where
                          thePatch's code jumps to }
                        IF (newerPatchPtr^.jmpPtr <> NIL) THEN
                            BEGIN
                            IF LongIntPtr(newerPatchPtr^.jmpPtr)^ = $2F2F0004 THEN
                                T1PBlockPtr(newerPatchPtr^.jmpPtr)^.OldTrapAddr :=
                                                              thePatch.oldTrapAddr
                            ELSE IF IntegerPtr(newerPatchPtr^.jmpPtr)^ = $2F3C THEN
                                TPBlockPtr(newerPatchPtr^.jmpPtr)^.OldTrapAddr :=
                                                              thePatch.oldTrapAddr
                            ELSE
                                BEGIN
                                {$IFC qDebug}
                                Writeln('###In UnpatchTrap: can''t figure out ',
                                        'what kind of patch ', ORD(newerPatchPtr),
                                        ' is!');
                                DebugStr('Can''t unpatch trap.');
                                {$ENDC}
                                END;
                            END;
                        END;

                    { Unlink the patch from the linked list of patches }
                    IF @thePatch = pPatchList THEN
                        pPatchList := thePatch.nextPatch
                    ELSE
                        BEGIN
                        aPatchPtr := pPatchList;
                        WHILE (aPatchPtr <> NIL) & (aPatchPtr^.nextPatch <>
                          @thePatch) DO
                            aPatchPtr := aPatchPtr^.nextPatch;
                        { Couldn't find thePatch, so don't try to unpatch it. }
                        IF aPatchPtr = NIL THEN
                            EXIT(UnpatchTrap);
                        aPatchPtr^.nextPatch := thePatch.nextPatch;
                    END;

                { If the patch allocated a block in the system heap,
                  deallocate it }
                WITH thePatch DO
                    jmpPtr := DisposeIfPtr(jmpPtr);
```

9. Fixed in MacApp 2.0.1.
10. `IsClassIDMemberClass` does not range check for negative class IDs. This could result in some extremely rare cases where a handle appears to be an object when it really is not.

Solution:

In the file UObject.a, replace:

```
                Cmp.W       (A0),D0   ; make sure class ID is in range
                     Bge.S       isFALSE
                     Cmp.W       (A0),D1   ; make sure class ID is in range
```

With:

```
                Cmp.W       (A0),D0  ; make sure class ID is in range
                Bge.S       isFALSE

                Tst.W       D0       ; make sure class ID is non-negative
                Blt.S       isFALSE

                Move.W      D0,D2    ; make sure class ID is even
                And         #1,D2
                Tst.W       D2
                Bnz.S       isFALSE

                Cmp.W       (A0),D1  ; make sure class ID is in range
                Bge.S       isFALSE

                Tst.W       D1       ; make sure class ID is non-negative
                Blt.S       isFALSE

                Move.W      D1,D2    ; make sure class ID is even
                And         #1,D2
                Tst.W       D2
```

11. Discipline signals a problem on a `_Get1NamedResource` call when it tries to load `CODE("GMain")`. This segment is listed in `'seg!'` and `'res!'`, but it does not exist.

Fix:

This bug is ultra-benign, but can be fixed by removing the reference to `GMain` in the file MacApp.r

12. The number of calls to `RegisterStdType` has increased from 17 to 25 since the MacApp 2.0b9 release; however, the limit (`kMaxSignatures`, defined in the file UMacApp.p) remains at 32. This difference means your application can only register seven additional types instead of the 15 previously allowed.

Fix:

Recompiling MacApp with a limit of 40 should suffice for now. Future versions of MacApp will implement a dynamic list so that no limits would be imposed.

13. Fixed in MacApp 2.0.1.
14. Fixed in MacApp 2.0.1.
15. Fixed in MacApp 2.0.1.
16. MATextBox may have problems if you are drawing with a wide font into a small box in a right-justified script system.

Fix:

In UMacAppUtilities.inc1.p, add a local Integer variable called minWidth. Then, replace:

```
                WITH destRect DO
```

With:

```
                WITH destRect DO
                    BEGIN
                    minWidth := Max(Max(right - left, widMax), 20);
                    CASE GetActualJustification(itsJust) OF
                        teJustLeft,
                        teForceLeft: right := left + minWidth;
                        teJustRight: left := right - minWidth;
                        teJustCenter:
                            BEGIN
                            left := (right+left-minWidth) DIV 2;
                            right := left + minwidth;
                            END;
                        END;
```

Back to top

MABuild Bugs

13. MABuild does not support both AppName.r and AppName.rsrc files as part of a MacApp project. Actually, the problem is a more general one: the file Build Rules and Dependencies defines the default dependency ".rsrc [[florin]] .r". Therefore, if AnyFile.rsrc is mentioned either in the file Basic Definitions or your own .MAMake file, Make produces a command that compiles AnyFile.r into AnyFile.rsrc, or complains if AnyFile.r does not exist.

Solution:

Avoid the .rsrc suffix for files that are not compiled from .r files.

Fix:

Globally replace ".rsrc" with ".r.o" in the files {MATools}Basic Definitions and {MATools}Build Rules and Dependencies. This change causes Make to create Anyfile.r.o files instead of AnyFile.rsrc files, removing the conflict and preserving any .rsrc files that you may have created with ResEdit or ViewEdit. Be sure to update your .MAMake file similarly.

2. MABuild doesn't support spaces or multiple files in the OtherViewTypesSrc Make variable, because of the following line in the file Build Rules and Dependencies:

Assuming OtherViewTypesSrc is set to something like "My Hard Disk:My Folder:My File.r", that line gets expanded to:

The double quotes on either end cancel each other out, and any pathname with spaces is treated as separate items. Compounding the problem is the fact that "OtherViewTypesSrc" is the name of both a Make variable and a Shell variable.

Fix:

Support for spaces in OtherViewTypesSrc can be easily added. In {MATools}Basic Definitions, replace:

With:

In {MATools}Build Rules and Dependencies, replace:

```
IF "{OtherViewTypesSrc}" != ""
    SET OtherViewTypesSrc "{OtherViewTypesSrc}"
    SET XIncludeOtherViewTypes 1
```

With:

```
IF {OtherViewTypesSrc} != ""
    SET XOtherViewTypesSrc {OtherViewTypesSrc}
    SET XIncludeOtherViewTypes 1
```

This stuff occurs three times, replace it in all three locations. Next, in {MARIncludes}ViewTypes.r, replace the line:

```
#Include $$Shell("OtherViewTypesSrc");
      // let end users extend the view
      //  type
```

With:

```
#Include $$Shell("XOtherViewTypesSrc");
      // let end users extend the view
      //  type
```

3. MABuild doesn't support more than one user library.

Solution:

Not yet determined. Requires a change to MABuildTool.p.

4. Creating an application with `qNeedsROM128K` set to `TRUE` and running it on a 512KE under System 3.2 causes it to bomb with an ID = 12 error, because the traps that MacApp needs are not present. However, the application runs properly under System 3.4, as the traps are implemented under that system.

Fix:

Tell MacApp to use the set of glue routines that check for the presence of the needed trap before it is called. In {MAPInterfaces}UPrinting.p, replace the following lines:

```
{$IFC NOT qNeedsROM128K}
{$IFC UNDEFINED UsingPrinting} {$I Printing.p} {$ENDC}
{$ELSEC}
{$IFC UNDEFINED UsingPrintTraps} {$I PrintTraps.p} {$ENDC}
```

With:

```
               {$IFC UNDEFINED UsingPrinting} {$I Printing.p} {$ENDC}

     In {MALibraries}PrivateInterfaces:UPrinting.p, replace:

               {$IFC NOT qNeedsROM128K}
               Printing,
               {$ELSEC}
               PrintTraps,
```

With:

5. At the top of the file UMacAppUtilities.inc1.p are the following compiler options:

```
        {$W+}
        {$R-}
        {$Init-}
        {$OV-}
        {$IFC qNames}
        {$D+}
```

The intent here is that these routines should not have debugger probes (`%_BP`, `%_EP`, `%_EX`) inserted into them, allowing them to run at full speed. Unfortunately, if you compile with something like `MABuild -NoDebug -Trace`, the debugger probes are inserted.

Fix:

Add {$D-} before {$IFC qNames}

6. The Commando dialog box for `MABuild` is out of date. For example, `-NeedsSystem6` and `-NoDebug` are now the `MABuild` default and cannot be turned off through the Commando dialog box.
7. The help button in the debug options dialog box in the `MABuild` Commando interface is partially obscured.
8. The Commando dialog has a three-state button "Show Times", that sets the flag "-T". The help text for this is "Have all tools show elapsed time." Actually, "-T" tells only `MABuildTool` to show elapsed time; to have all tools do this, you need the "-TT" flag.
9. There is a small problem in the file {MAPInterfaces}UTEView.p that causes your compiles to be imperceptibly slower than you would expect. Several references to `__TEView__` at the top of the file should really be `__UTEView__`, thus:

```
         {$IFC UNDEFINED __UTEView__}
         {$SETC __UTEView__ := FALSE}
         {$ENDC}

         {$IFC NOT __UTEView__}
```

10. In the file UViewCoords.h, #ifndef __UVIEWCOORDS__ should be #ifndef __UViewCoords__.

Fix:

Change the header file.

11. MacApp uses CPlusLib instead of CPlusLib881 when compiling for C++ and FPU support.

Fix:

In the file Basic Definitions, remove `"{CLibraries}CPlusLib.o"` from the definition of `31CPlusSupport`, add it to `31CPlusNonFPUSANELib`, and add `"{CLibraries}CPlusLib881.o"` to `31CPlusFPUSANELib`. Thus, replace:

```
#############
# For MPW 3.0, 3.1
#############
31CPlusSupport = [[partialdiff]]
    "{CLibraries}CRuntime.o" [[partialdiff]]
    "{CLibraries}CInterface.o" [[partialdiff]]
    "{CLibraries}CPlusLib.o" [[partialdiff]]
    "{CLibraries}StdCLib.o" [[partialdiff]]
    "{PLibraries}PasLib.o"

31CPlusNonFPUSANELib = [[partialdiff]]
    "{CLibraries}CSANELib.o" [[partialdiff]]
    "{PLibraries}SANElib.o" [[partialdiff]]
    "{CLibraries}Math.o" [[partialdiff]]
    "{CLibraries}Complex.o"

31CPlusFPUSANELib = [[partialdiff]]
    "{CLibraries}CLib881.o" [[partialdiff]]
    "{CLibraries}CSANELib881.o" [[partialdiff]]
    "{PLibraries}SANELib881.o" [[partialdiff]]
    "{CLibraries}Math881.o" [[partialdiff]]
```

With:

```
#############
# For MPW 3.0, 3.1
#############
31CPlusSupport = [[partialdiff]]
    "{CLibraries}CRuntime.o" [[partialdiff]]
    "{CLibraries}CInterface.o" [[partialdiff]]
    "{CLibraries}StdCLib.o" [[partialdiff]]
    "{PLibraries}PasLib.o"

31CPlusNonFPUSANELib = [[partialdiff]]
    "{CLibraries}CPlusLib.o" [[partialdiff]]
    "{CLibraries}CSANELib.o" [[partialdiff]]
    "{PLibraries}SANElib.o" [[partialdiff]]
    "{CLibraries}Math.o" [[partialdiff]]
    "{CLibraries}Complex.o"

31CPlusFPUSANELib = [[partialdiff]]
    "{CLibraries}CPlusLib881.o" [[partialdiff]]
    "{CLibraries}CLib881.o" [[partialdiff]]
    "{CLibraries}CSANELib881.o" [[partialdiff]]
    "{PLibraries}SANELib881.o" [[partialdiff]]
    "{CLibraries}Math881.o" [[partialdiff]]
```

12. "MABuild's mechanism for handling C++ Load/Dump is sort of lame. Why not support FPU and Load/Dump simultaneously? It's not that hard to get working."

Fix:

Yeah, but it used to be. So there. MABuild is trying to work around a problem that exists in CFront 3.1b3 and earlier. If you are using a later version, you can remove the safety check. Go into the file MABuildTool.p, remove the following lines, then rebuild `MABuildTool`.

```
               { C++ external symbol table files support }
               IF fCPlusLoad & fNeedsFPU THEN
                   BEGIN
                   Echo('''###'' MABuild: Warning: CPlusLoad and NeedsFPU
                        are incompatible.  Using NoCPlusLoad.');
                   fCPlusLoad := FALSE;
```

13. This is not a bug with MABuild, but this change belongs in the MABuild section. With all the changes and fixes suggested here, one of MacApp's segments--GRes--becomes uncomfortably close to 32K.

Fix:

Move the routines originally mapped to MAControlRes and MADocumentRes into GRes2 by opening the file {MATools}Basic Definitions. Change the occurrence of MAControlRes=GRes to MAControlRes=GRes2 and MADocumentRes=GRes to MADocumentRes=GRes2.

Back to top

Bugs Only In Debug Mode

These bugs occur only in debug versions of your program, and do not affect the final production version.

13. DisposeIfHandle fails if called with a valid, but purged, handle:

```
        h := NewHandle(20);
        IF h <> NIL THEN
            BEGIN
            EmptyHandle(h);
            DisposIfHandle(h); {<--PBreak: 'handle is so
                                    bad, couldn't get handle bits'}
```

Fix:

In DisposeIfHandle (file UMacAppUtilities), add:

```
               IF IsHandlePurged(aHandle) THEN
                         { h might have been purged }
                   BEGIN
                   DisposHandle(aHandle);
                   EXIT(DisposeIfHandle);
```

Just before: This fix is not the cleanest, but it is the easiest.

2. Doctor, doctor. My application hangs if Print... is chosen while stopped in the debugger.

Solution:

Don't do that.

3. With a desk accessory open in the application heap (e.g., Option-Alarm Clock), you can enter the MacApp debugger, but it does not accept any keystrokes.

Solution:

Click in the Debug Transcript window to jumpstart it.

4. If the performance tools are on, you must turn them off with "T"oggle before "E"nding. Failure to do so leaves the

performance tools active, although their data has been disposed.

Solution:

Always "T"oggle the performance tools off before "E"nding.

Fix:

Modify `PerfCmd` to turn off the performance tools when "E"nding.

5. `TTranscriptView` does not initialize `fFontInfo` in `CommonInit`.

Fix:

Before the {$Pop} statement in `TTranscriptView.CommonInit` (file UTranscriptView.inc1.p), add:

6. `TList.GetSameItemNo` fails in debug if looking for `NIL`. With previous versions of MacApp, it was perfectly acceptable to check for a `NIL` object in a list. `GetSameItemNo` would return zero, as expected. With MacApp 2.0, there is an explicit check in debug mode that the object is valid, so passing `NIL` does not work.

Solution:

Call `GetSameItemNo` with the following wrapper:

```
IF obj = NIL then
    index := 0
ELSE
```

Fix:

Modify `TList.GetSameItemNo` (file TList.inc1.p) to make the same check.

7. If a failure occurs in `IApplication`, the debugger incorrectly issues the following warning:

"You're leaving a routine without calling Success for a handler that will be destroyed."

This message occurs because the routine `MADebuggerMainEntry` checks `gTopHandler` to see if the `FailInfo` record it points to is below the stack. However, this test doesn't work properly if `gTopHandler` is `NIL`, as it is in `IApplication`.

Fix:

Add a check for `(gTopHandler = NIL)` in `MADebuggerMainEntry` (file UDebug.inc1.p). Replace the line:

```
forgotSuccess := ((which = tEnd) | (which = tExit))
                 & (StripLong(LongIntPtr(pLink)^) >=
```

With:

```
forgotSuccess := ((which = tEnd) | (which = tExit))
                  & (gTopHandler <> nil)
                 & (StripLong(LongIntPtr(pLink)^) >=
```

8. Fixed in MacApp 2.0.1.
9. Fixed in MacApp 2.0.1.
10. There are two problems with `DebugGetActiveWindow`. These affect you only if you try to inspect the labels

GetActiveWindow or GetActiveDocument.

Fix:

In UDebug.inc1.p, replace the following lines in DebugGetActiveWindow:

```
                pDebugWindow.fFloats := FALSE;
                  { so the debugger window doesn't get reported }
                DebugGetActiveWindow := gApplication.GetActiveWindow;
```

With:

```
                pDebugWindow.fFloats := TRUE;
                  { so the debugger window doesn't get reported }
                DebugGetActiveWindow := pSavedState.gApplication.GetActiveWindow;
```

Back to top

MPW 3.2 Compatibility

This section describes problems that occur when trying to build MacApp 2.0 under MPW 3.2. MacApp 2.0 was developed under MPW 3.0 and 3.1 and could not take into account changes made to MPW 3.2.

Note:

Even at the time of this writing, it is unclear which of the following items will be compatibility problems. For example, item four is a problem with MPW 3.1a1, but not with MPW 3.2b1. On the other hand, item three is a problem with MPW 3.2b1, but not with MPW 3.2a1. Apple will update the status of these items when MPW 3.2 is final.

8. The file {MALibraries}PrivateInterfaces:UDebug.p needs symbol information from the file Packages.p. Under MPW 3.1, this file was automatically included when the file UDebug.p included the file Script.p in its USES statement. Under MPW 3.2, this is no longer the case, and UDebug does not compile.

Fix:

Add a reference to Packages before Script in the file UDebug.p:

```
                USES
                    <etc.>
                    Desk, DiskInit, ToolUtils, Retrace,
                    Memory, Resources, FixMath, Packages,
                    Script, PasLibIntf, OSEvents, Traps,
```

2. The file UDebug.inc1.p contains the definition for the following procedure:

```
        PROCEDURE JTOffProc(A5JTOffset: UNIV INTEGER;
```

DisAsmStr80 is declared in the file {PInterfaces}DisAsmLookup.p under MPW 3.1. It is no longer used under MPW 3.2.

Fix:

Change DisAsmStr80 to Str255.

3. In the NMRec record defined in the files Notification.c and Notification.p, nmSIcon has been changed to the

infinitely clearer `nmIcon`.

Fix:

In UDebug.inc1.p, change the occurrence of `nmSIcon` to `nmIcon`.

4. At the bottom of the file UDebug.a, there is a line that looks like the following:

`TIOFLUSH` is not supported under MPW 3.2a1, and the Assembler aborts with an error when it gets to this line.

Fix:

Comment out or remove the reference to `TIOFLUSH`:

Back to top

SADE Compatibility

3. In the SADEScripts folder (part of the SADE product) is a file called StepMethod. This file contains the definition of a procedure called `stepIntoMethod`, which includes the following lines:

```
        break %_NEWMETHOD020.CacheOut
        break %_NEWMETHOD020.TableOut
        go
        unbreak %_NEWMETHOD020.CacheOut
```

MacApp 2.0 no longer defines the symbol `%_NEWMETHOD020` and SADE is not able to find it when you attempt to step into an overridden method.

Fix:

Replace those lines with the following:

```
                break %_NEWMETHOD.CacheOut
                break %_NEWMETHOD.TableOut
                go
                unbreak %_NEWMETHOD.CacheOut
```

Back to top

THINK Pascal Compatibility

3. Fixed in MacApp 2.0.1.
4. This isn't really a bug, but you might incorporate the following: in the file UMacAppUtilities.p, place a {$PUSH} {$D-} in front the BlockSet routine and a {$Pop} after it. This change speeds up the execution of programs which are compiled with the MacApp debugger when running under the THINK Pascal environment. (Doing this may not be necessary if you incorporate the fix to problem #5 in the MABuild section.)

Back to top

MacApp Samples Bugs

3. In the C++ version of DemoText, strings which normally appear in the About box show up in the color picker, because `kPromptStringsID` is declared differently between the Rez file and the C++ file.
4. In the file UIconEdit.inc1.p, the procedure `TIconBitMap.Free` does not call `INHERITED Free`. It should call `INHERITED Free` or the space in the heap used for the object never gets freed.
5. Instead of referring to `@fShowInvisibles`, `TTabTEView.Fields` actually refers to `@ShowInvisibles`.

Back to top

Other

3. The script {MATools}CleanupDeRezzedViews misses a situation where it needs to quote a Shell variable. This problem causes the script to abort if the file you are processing contains a space in it.

Fix:

Replace the second line of the script:

With:

2. Fixed in MacApp 2.0.1.

THINK Pascal is a trademark of Symantec Corporation.

Back to top

Downloadables

Acrobat version of this Note (K). Download

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code