

[illegible]

```

- init {
    char          buf[MAXPATHLEN];
    NXBundle      *bundle;

    [super init];
    theAgent = nil;
    printInfo = nil;
    etDoc = nil;
    bundle = [NXBundle bundleForClass:[eTDocUI class]];
    if ([bundle getPath:buf forResource:"eTDocUI" ofType:"nib"] ) {
        [NXApp loadNibFile:buf owner:self withNames:NO];
    } else {
        NXLogError("NIB not found: eTDocUI");
    }
    return self;
}

- free {
    //Commented out by RK on 8/22; put back in eText5
    if ([NXApp printInfo] == printInfo) [NXApp setPrintInfo:nil];
    // Taken back out on 12/1 -- can't see why, but it fails.
    // [NXApp delayedFree:printInfo];

    [etDoc setSelectedObj:nil];
    //[etDoc unregisterNotification:self]; Why Bother?
    theAgent = [theAgent free];
    theContainer = [theContainer free];
    [theWindow setDelegate:nil];
    eTextObj = [eTextObj free];
    theWindow = [theWindow close];
    // possible fix, 8/22 since TE will do the next freewhencosed.
    // eTextObj will be freed automagically. eTDoc is still alive here
    // Paranoia!
    NXPing();
    return self = [super free];
}

- awakeFromNib {
    static int i=0;
    NXRect      r;

    // User Interface Initialization: load in the eText + ScrollView
    [theWindow disableDisplay];
    [theSplitview addSubview:theScroller];
    [theScroller getContentSize:&(r.size)];
    [eTextObj setVertResizable:YES];
    [eTextObj setHorizResizable:NO];
    [eTextObj sizeTo:r.size.width :r.size.height];
    [eTextObj setMinSize:&(r.size)];

```

```

r.size.width = MAXFLOAT; r.size.height = MAXFLOAT;
[eTextObj setMaxSize:&(r.size)];
// Lay out the window at an offset
[theWindow setFrame:&r];
[theWindow moveTo:r.origin.x + (i%10) * 22 :r.origin.y - (i%10) * 22];
i++;
// Display the window
[[theWindow reenableView] display];
return self;
}
- setDoc:theDoc {
    char            RTFfile[MAXPATHLEN];

    etDoc = theDoc;
    [etDoc registerNotification:self];
    [eTextObj setDoc:etDoc];
    [theWindow disableDisplay];
    [[etDoc undoManager] disableUndoRegistration];

    // Read in .etUInfo parameters
    if (!access([[etDoc docInfo] docPath], R_OK)) {
        NXStream      *s;
        float          dh, h, w;

        sprintf(RTFfile, "%s/"UIINFOFILE, [[etDoc docInfo] docPath]);
        s = NXOpenTypedStreamForFile(RTFfile, NX_READONLY);
        if (!s) { // eText4 Compatibility
            sprintf(RTFfile, "%s/"UIINFOFILE2, [[etDoc docInfo] docPath]);
            s = NXOpenTypedStreamForFile(RTFfile, NX_READONLY);
        }
        if (s) {
            NXRect r;

            printInfo = NXReadObject(s);
            NXReadTypes(s, "fffii", &dh, &h, &w, &begin, &end);
            [theWindow setFrame:&r];
            r.origin.y += r.size.height - h;
            r.size.height = MIN(h, r.origin.y + r.size.height); // don't fall off the
bottom
            r.size.width = w;
            [theWindow placeWindow:&r];
            // we can't invert this operation: [theSplitview dh];
            if ([eTextObj isEditable]) [eTextObj setSel:begin :end];
            NXCloseTypedStream(s);
        }
    } else {
        [eTextObj setEditable:YES];
    }
}

```

[illegible]

```

- saveTo:sender {
    if ([self needsSaving]) {
        [self saveTo:sender changePath:YES forceETFD:YES];
    }
    return self;
}

- formatAction:sender { // Private Callback hook!
    static BOOL _warned=NO;
    const char *docdir;
    id sp = [SavePanel new];

    int fmt = [[sender selectedCell] tag];
    switch (fmt) {
        case ETFD_FMT:
            [sp setRequiredFileType:ETFD_EXT];
            docdir = [userModel stringQuery:ETFDIRECTORY];
            if (docdir && *docdir) {
                [sp setDirectory:docdir];
            }
            break;
        case HTMD_FMT:
            [sp setRequiredFileType:HTMD_EXT];
            if (!_warned) _warned = [userModel boolQuery:HTMD_WARNED];
            if (!_warned) {
                NXRunAlertPanel("Save as HTMD","Remember to 1) keep them in the same
directory and 2) no spaces in filenames.", "OK", NULL, NULL);
                _warned = YES;
            }
            docdir = [userModel stringQuery:HTMDIRECTORY];
            if (docdir && *docdir) {
                [sp setDirectory:docdir];
            }
            break;
        case ASCII_FMT:
            [sp setRequiredFileType:ASCII_EXT];
            break;
        case C_FMT:
            [sp setRequiredFileType:""];
            break;
        case TeXD_FMT:
            [sp setRequiredFileType:TeXD_EXT];
            break;
        case RTF_FMT:
            [sp setRequiredFileType:RTF_EXT];
            break;
    }
    return self;
}

```

```

}
- saveTo:sender changePath:(BOOL) changeIt forceETFD:(BOOL) forceIt
{
    char                p[MAXPATHLEN], *f=NULL;
    int                 fmt;
    id                  savepanel = [SavePanel new];
    static id           formatPopup = nil, formatButton = nil;

    // PHASE 0: Creating a formatPicker
    if (!formatPopup) {
        char item[MAXPATHLEN];

        formatPopup = [[PopUpList alloc] init];
        [formatPopup changeButtonTitle:YES];
        [formatPopup setAction:@selector(formatAction:)];
        sprintf(item,"%s (.%s)", ETFD_DESC, ETFD_EXT);
        [[[formatPopup addItem:item] setTag:ETFD_FMT] setKeyEquivalent:'1'];
        sprintf(item,"%s (.%s)", HTMD_DESC, HTMD_EXT);
        [[[formatPopup addItem:item] setTag:HTMD_FMT] setKeyEquivalent:'2'];
        sprintf(item,"%s (.%s)", TeXD_DESC, TeXD_EXT);
        [[[formatPopup addItem:item] setTag:TeXD_FMT] setKeyEquivalent:'3'];
        sprintf(item,"%s (.%s)", RTF_DESC, RTF_EXT);
        [[[formatPopup addItem:item] setTag:RTF_FMT] setKeyEquivalent:'4'];
        sprintf(item,"%s (.%s)", ASCII_DESC, ASCII_EXT);
        [[[formatPopup addItem:item] setTag:ASCII_FMT] setKeyEquivalent:'5'];
        sprintf(item,"%s (.%s)", C_DESC, C_EXT);
        [[[formatPopup addItem:item] setTag:C_FMT] setKeyEquivalent:'6'];
        formatButton = NXCreatePopUpListButton(formatPopup);
    }

    // PHASE I: Initializing & Running the savePanel

    strcpy(p, [[etDoc docInfo] docPath]);
    if (rindex(p, '.') *rindex(p, '.')= '\0';
    if (rindex(p, '/')) {
        f = rindex(p, '/')+1;
        *rindex(p, '/')= '\0';
    }

    [savepanel setDirectory:p];          // First crack; htmd or etfd may chdir
    [formatPopup setTarget:self];
    if (forceIt) {
        [savepanel setTitle:"Save/eText Format"];
        [savepanel setAccessoryView:nil];
        [savepanel setRequiredFileType:ETFD_EXT];
    } else {
        [savepanel setTitle:"Save/Any Format"];
    }
}

```

```

        [savepanel setAccessoryView:formatButton];
        [self formatAction:[formatPopup itemList]];
    }
    if (![savepanel runModalForDirectory:[savepanel directory] file:f])
        return nil;

    // PHASE II: Consistency Checks, write datafiles
    fmt = (forceIt) ? ETFD_FMT : [[[formatPopup itemList] selectedCell] tag];
    strcpy(p, [savepanel filename]);

    if (fmt==HTMD_FMT) {
        sprintf(p, "%s/%s", [savepanel directory], rindex([[etDoc docInfo]
docPath], '/')+1);
        *(rindex(p, '.')+1) = '\0';
        strcat(p, HTMD_EXT);
        if (strcmp(p, [savepanel filename])) {
            // i.e. is the basename != the original pathname (for link anchors)
            int choice = NXRunAlertPanel( "Save As HTMD",
                "You must use the filename %s to make links to this document.",
                "OK", "Shut Up And Do What I Said", NULL, p);
            if(choice==NX_ALERTALTERNATE)
                strcpy(p, [savepanel filename]); // do what the bonehead sez...
        }
    }

    [etDoc saveTo:p inFormat:fmt changePath:changeIt];
    if (fmt == ETFD_FMT) {
        [theWindow setDocEdited:NO];
        [theWindow setTitleAsFilename:[etDoc docInfo] docPath]];
    }
    return self;
}

- close:sender {
    return [self close:sender allowCancel:YES];}
- close:sender allowCancel:(BOOL)cancellable
{
    // Run a possible alert panel
    if ([self needsSaving]) {
        const char *name;
        int choice;

        if (*[[etDoc docInfo] docPath])
            name = rindex([[etDoc docInfo] docPath], '/')+1;
        else
            name = NXUniqueString("Untitled");
    }
}

```

```

choice = NXRunAlertPanel("Close","Save changes to %s?",
    "Save", "Don't Save", cancellable ? "Cancel" : NULL, name);
switch (choice) {
    case NX_ALERTDEFAULT: [self save:self];
                        break;
    case NX_ALERTOTHER:  return nil;
}
}
// The actual closing begins from eTApp
[etApp perform:@selector(closeID:) with:[[etDoc docInfo] docID]
    afterDelay:0 cancelPrevious:NO];
return self;
}

```

```

- revert:sender
{
    int         choice;
    const char *name;
    NXSelPt     bPt, ePt;

    // PHASE I: Run a possible alert panel
    if (*[[etDoc docInfo] docPath])
        name = rindex([[etDoc docInfo] docPath], '/') + 1;
    else {
        NXBeep(); return nil;
    }

    if ([self needsSaving] && name) {
        choice = NXRunAlertPanel("Revert","Discard changes to the document %s?",
            "Revert", "Cancel", NULL, name);
        if (choice == NX_ALERTALTERNATE) {return nil;}
    }

    // PHASE II: Reload
    [theWindow disableDisplay];
    [eTextObj getSel:&bPt :&ePt];
    begin = bPt.cp; end = ePt.cp;
    [[etDoc undoManager] disableUndoRegistration];
    [[eTextObj setSel:0 :[eTextObj textLength]] delete:self];
    [[etDoc undoManager] reenableViewUndoRegistration];
    [[etDoc undoManager] emptyUndoManager];
    if (theAgent) {
        [theContainer removeFromSuperview];
        [theContainer free];
        theContainer = theAgent = nil;
    }
    [eTextObj setSel:begin :end];
    [theWindow setDocEdited:NO];
}

```



```

[etDoc setSelectedObj:nil];
[inspector inspect:nil];
[[theWindow reenableDisplay] display];
return self;
}
- print:sender
{
    [eTextObj printPSCode:sender];
    return self;
}

- (BOOL)validateCommand:(MenuCell *)menuCell
{
    SEL action = [menuCell action];
    BOOL enable;
    static NXAtom removeAgent = NULL;

    if (!removeAgent) removeAgent = NXUniqueString("Detach Agent");

    if (NXUniqueString([menuCell title])==removeAgent) {
        enable = (theAgent?YES:NO);
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    } else if ((action == @selector(revert:)) ||
                (action == @selector(save:))) {
        enable = [self needsSaving];
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    } else if ((action == @selector(saveAs:)) ||
                (action == @selector(saveTo:)) ||
                (action == @selector(print:)) ||
                (action == @selector(close:))) {
        enable = YES;
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    }
}

```

```

} else if (action == @selector(undo:)) {
    char buf[256];

    if (![etDoc undoManager] lastUndoName){
        if ([menuCell isEnabled]) {
            [menuCell setTitle:"Undo"];
            [menuCell setEnabled:NO];
            return YES;
        }
    } else {
        sprintf(buf, "Undo %s", [[etDoc undoManager] lastUndoName]);
        if (strcmp(buf, [menuCell title])) {
            [menuCell setTitle:buf];
            [menuCell setEnabled:YES];
            return YES;
        }
    }
} else if(action == @selector(redo:)){
    char buf[256];

    if (![etDoc undoManager] lastRedoName){
        if ([menuCell isEnabled]) {
            [menuCell setTitle:"Redo"];
            [menuCell setEnabled:NO];
            return YES;
        }
    } else {
        sprintf(buf, "Redo %s", [[etDoc undoManager] lastRedoName]);
        if (strcmp(buf, [menuCell title])) {
            [menuCell setTitle:buf];
            [menuCell setEnabled:YES];
            return YES;
        }
    }
} else {
    return NO;
}
return NO;

```

[illegible]

```

        if (choice == NX_ALERTALTERNATE) {return nil;}
    }
    [theWindow disableDisplay];
    if (theAgent) {
        // [[[undoManager setUndoTarget:self] freeUndoArgs] attachAgent:theAgent];
        // [undoManager setUndoName:"Replace Agent"];
        // [undoManager setRedoName:"Replace Agent"];
        [theContainer removeFromSuperview];
        theContainer = [theContainer free];
        [theSplitview setDelegate:self];
        theContainer = theAgent = nil;
    } else {
        // [[[undoManager setUndoTarget:self] detachAgent];
        // [undoManager setUndoName:"Attach Agent"];
        // [undoManager setRedoName:"Attach Agent"];
    }
    // PHASE II: Find the new agent -- either in sender or tables
    if ([newAgent conformsTo:@protocol(Agent)])
        theAgent = newAgent;
    [[theAgent controlView] getFrame:&frame];
    theContainer = [[eTContainerView alloc] initWithFrame:&frame];
    [theContainer attachAgent:theAgent];
    [theSplitview setDelegate:theContainer];
    [theSplitview addSubview:theContainer];
    [theContainer getMinSize:&sz];
    sz.width += 30.0;
    sz.height += 230.0;
    [theWindow setMinSize:&sz];
    [[theWindow reenableDisplay] display];
    return self;
}

```

```

- detachAgent {
    [theWindow disableDisplay];
    if (theAgent) {
        // [[[undoManager setUndoTarget:self] freeUndoArgs] attachAgent:theAgent];
        // [undoManager setUndoName:"Reattach Agent"];
        // [undoManager setRedoName:"Reattach Agent"];
        [theContainer removeFromSuperview];
        [theContainer free];
        [theSplitview setDelegate:self];
        theContainer = theAgent = nil;
    }
    [[theWindow reenableDisplay] display];
    return self;
}

```

