

```
//#####
// FILENAME:   eTDoc.m
// SUMMARY:    Implementation of the abstract document controller.
// SUPERCLASS: Object:eTDoc
// PROTOCOLS: Uses <DocNotification>
// INTERFACE: None
// AUTHOR:     Rohit Khare
// COPYRIGHT: ©1993,94 California Institute of Technology, eText Project
//#####
// Implementation Comments
//      Keep a careful eye on the construction/destruction process!
//      Also watch for the inanities of how file-reading is covered between etDoc
//      and eTAppUI.
//#####
// HISTORY
// 10/06/94:   Revamped for eText5.
// 07/20/94:   Added LaTeX as per Tim Berners-Lee's sed script; cleanup.
// 06/17/94:   Added writeHTMD isBold,isItalic,isFixedPitch. By RK and TRZ.
// 01/14/94:   Completely revised for version 4.0
// 09/11/93:   Created (un)registerForDocNotification. <DocNotification>
// 09/09/93:   Added support for the docID member.
// 08/22/93:   Moved "Accessor Methods" to eTDoc
// 08/21/93:   Created.
//#####
// Imported Interfaces
//
//import "eTDoc.h"
//import "eTDocInfoUI.h"
//import "eTDocUI.h"

@implementation eTDoc

//#####
// Class Management
//
- initWithDocInfo:(eTDocInfo *) newDocInfo {
    Panel *oPanel;
    NXSelPt b,e;

    [super init];

    // PHASE I: Initialize the internal model
    theDocInfo = newDocInfo;
    notificationList = [[List alloc] init];
    [self registerNotification:navigator];
    componentTable = [[HashTable alloc] initWithKeyDesc:@"%"];
    undoManager = [[UndoManager alloc] init];
}
```

```

[undoManager setLevelsOfUndo:50]; // this should be an NXDefault
[theDocInfo setDoc:self];
selectedObj = theDocInfo;
if (*[theDocInfo agent])
    [self attachAgent:[[[etApp agentByName:[theDocInfo agent]]
                        alloc] init:theDocInfo]];
else theAgent = nil;
[undoManager disableUndoRegistration];

// PHASE II: Construct the UI
oPanel = NXGetAlertPanel("eTDoc",
    "Opening the document titled %s.",
    NULL, NULL, NULL, [theDocInfo docTitle]);
[oPanel setFloatingPanel:NO];
[oPanel setHideOnDeactivate:YES];
[oPanel orderFront:self];
NXPing();
docUI = [[[eTDocUI alloc] init] setDoc:self];

// PHASE III: Opening The Files
[self openFrom:[theDocInfo docPath]];

// PHASE IV: Notifications
// There's a little fudging between eTDoc and eTDocUI here because the
// selection can get screwed up by the broadcast.
[[docUI window] disableDisplay];
[[docUI eTextObj] getSel:&b :&e];
{
    int count;
    count = [notificationList count];
    while (count--)
        if([[notificationList objectAtIndex:count]
            respondsTo:@selector(docWillOpen:)])
            [[notificationList objectAtIndex:count] docWillOpen:self];
}
[[docUI eTextObj] setSel:b.cp :e.cp];
[[docUI eTextObj] scrollSelToVisible];
[[docUI window] setDocEdited:NO];
[[docUI window] reenableView];
[undoManager reenableView];
if (oPanel) {
    [oPanel orderOut:self];
    NXFreeAlertPanel(oPanel);
}
return self;
}
- free
{

```

[illegible]

```

// PHASE I: Notifications
{
    int count;
    count = [notificationList count];
    while (count--)
        if([[notificationList objectAtIndex:count]
            respondsTo:@selector(docWillWrite:)])
            [[notificationList objectAtIndex:count] docWillWrite:self];
}
sPanel = NXGetAlertPanel("eTDoc",
    "Saving document to \"%s\" in %s Format.",
    NULL, NULL, NULL, thePath, desc[fmt]);
[sPanel setFloatingPanel:NO];
[sPanel setHideOnDeactivate:YES];
[sPanel orderFront:self];
NXPing();
// PHASE II: Writing the files
switch(fmt) {
    case ETFD_FMT:      [self saveETFD:thePath changePath:changeIt]; break;
    case ASCII_FMT:     [self saveASCII:thePath]; break;
    case C_FMT:         [self saveC:thePath]; break;
    case TeXD_FMT:      [self saveLaTeX:thePath]; break;
    case HTMD_FMT:      [self saveHTMD:thePath]; break;
    case RTF_FMT:       [self saveRTF:thePath]; break;
}
if (sPanel) {
    [sPanel orderOut:self];
    NXFreeAlertPanel(sPanel);
}
// PHASE III: Notifications
{
    int count;
    count = [notificationList count];
    while (count--)
        if([[notificationList objectAtIndex:count]
            respondsTo:@selector(docDidWrite:)])
            [[notificationList objectAtIndex:count] docDidWrite:self];
}
[inspector thaw];
[[[docUI window] reenableView display] display];
return self;
}
- close:sender allowCancel:(BOOL)cancellable {
    return [docUI close:sender allowCancel:cancellable];}
- save:sender {
    return [docUI save:sender];}
- saveASCII:(const char *) thePath {
    [[docUI eTextObj] writeASCIItoPath:thePath];
}

```

```

    return self;}
- saveC:(const char *) thePath {
    [[docUI eTextObj] writeCtoPath:thePath];
    return self;}
- saveRTF:(const char *) thePath {
    [[docUI eTextObj] writeRTFtoPath:thePath];
    return self;}
- saveLaTeX:(const char *) thePath
{
    char                LaTeXfile[MAXPATHLEN];
    int                 i;
    struct stat         st;
    DIR                 *dirp;
    struct direct        *dp;
    NXStream            *s;
    taggingInfo        tags[8];

    // PHASE I: Prepare thePath
    if (stat(thePath, &st) || (((st.st_mode & S_IFMT) != S_IFDIR))) {
        removeFile(thePath);
        if (mkdir(thePath, 0777)) {
            NXLogError("Could not mkdir(\"%s\")", thePath);
            perror("eText5");
            NXBeep();
            return nil;
        }
    }
    [componentTable empty];

    // PHASE II: Prepare taggingInfo
    i=0;    // is incremented to compact out undefined tagging fonts
    tags[i].font = [theDocInfo tagFont:H1];
    tags[i].start = NXUniqueString("\\part{"); // use {book} with \chapter
    tags[i].end = NXUniqueString("}n");
    if (tags[i].font) i++;

    tags[i].font = [theDocInfo tagFont:H2];
    tags[i].start = NXUniqueString("\\section{");
    tags[i].end = NXUniqueString("}n");
    if (tags[i].font) i++;

    tags[i].font = [theDocInfo tagFont:H3];
    tags[i].start = NXUniqueString("\\subsection{");
    tags[i].end = NXUniqueString("}n");
    if (tags[i].font) i++;

    tags[i].font = [theDocInfo tagFont:H4];

```

```

tags[i].start = NXUniqueString("\\subsubsection{");
tags[i].end = NXUniqueString("}\n");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:H5];
tags[i].start = NXUniqueString("\\paragraph{");
tags[i].end = NXUniqueString("}\n");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:H6];
tags[i].start = NXUniqueString("\\subparagraph{");
tags[i].end = NXUniqueString("}\n");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:QT];
tags[i].start = NXUniqueString("\\begin{quote}\n");
tags[i].end = NXUniqueString("\\end{quote}\n");
if (tags[i].font) i++;

tags[i].font = NULL; tags[i].start = tags[i].end = NULL;

// PHASE III: Collect Annotations
sprintf(LaTeXfile, "%s/%s.tex", thePath, [theDocInfo docTitle]);
s = NXOpenMemory(NULL, 0, NX_WRITEONLY);
NXPrintf(s, "\\documentstyle[times,epsf]{article}\n\\begin{document}\n");
[theDocInfo writeLaTeXHeader:s];
[[docUI eTextObj] writeLaTeX:s withTags:tags];
NXPrintf(s, "\n\\end{document}\n");
NXSaveToFile(s, LaTeXfile);
NXCloseMemory(s, NX_FREEBUFFER);

// PHASE IV: Notifications
{
    int count;
    count = [notificationList count];
    while (count--)
        if([[notificationList objectAtIndex:count]
            respondsTo:@selector(writeComponentToPath:inFormat:)])
            [[notificationList objectAtIndex:count]
             writeComponentToPath:thePath inFormat:TeXD_FMT];
}

// PHASE V: Garbage Collection
// Iterate through the directory, deleting bastard objects
[self registerComponent:NXUniqueString(rindex(LaTeXfile, '/')+1)];
[self registerComponent:NXUniqueString(".")];
[self registerComponent:NXUniqueString("..")];
dirp = opendir(thePath);

```

```

dp = readdir(dirp);
while (dp != NULL) {
    if (![componentTable isKey:NXUniqueString(dp->d_name)]) {
        sprintf(LaTeXfile, "%s/%s", thePath, dp->d_name);
        removeFile(LaTeXfile);
    }
    dp = readdir(dirp);
}
closedir(dirp);
[componentTable empty];
return self;
}

- saveHTMD:(const char *) thePath
{
    char                HTMLfile[MAXPATHLEN];
    int                 i;
    struct stat         st;
    DIR                 *dirp;
    struct direct       *dp;
    NXStream            *s;
    taggingInfo         tags[8];

    // PHASE I: Prepare thePath
    if (stat(thePath, &st) || (((st.st_mode & S_IFMT) != S_IFDIR))) {
        removeFile(thePath);
        if (mkdir(thePath, 0777)) {
            NXLogError("Could not mkdir(\"%s\")", thePath);
            perror("eText5");
            NXBeep();
            return nil;
        }
    }
    [componentTable empty];

    // PHASE II: Prepare taggingInfo
    i=0; // is incremented to compact out undefined tagging fonts
    tags[i].font = [theDocInfo tagFont:H1];
    tags[i].start = NXUniqueString("<H1>"); // use {book} with \chapter
    tags[i].end = NXUniqueString("</H1>");
    if (tags[i].font) i++;

    tags[i].font = [theDocInfo tagFont:H2];
    tags[i].start = NXUniqueString("<H2>");
    tags[i].end = NXUniqueString("</H2>");
    if (tags[i].font) i++;

    tags[i].font = [theDocInfo tagFont:H3];

```

```

tags[i].start = NXUniqueString("<H3>");
tags[i].end = NXUniqueString("</H3>");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:H4];
tags[i].start = NXUniqueString("<H4>");
tags[i].end = NXUniqueString("</H4>");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:H5];
tags[i].start = NXUniqueString("<H5>");
tags[i].end = NXUniqueString("</H5>");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:H6];
tags[i].start = NXUniqueString("<H6>");
tags[i].end = NXUniqueString("</H6>");
if (tags[i].font) i++;

tags[i].font = [theDocInfo tagFont:QT];
tags[i].start = NXUniqueString("<BLOCKQUOTE>");
tags[i].end = NXUniqueString("</BLOCKQUOTE>");
if (tags[i].font) i++;

tags[i].font = NULL; tags[i].start = tags[i].end = NULL;

// PHASE III: Collect Annotations
sprintf(HTMLfile, "%s/"HTML_INDEX, thePath);
s = NXOpenMemory(NULL, 0, NX_WRITEONLY);
NXPrintf(s, "<HTML>\n");
[theDocInfo writeHTMLHeader:s];
NXPrintf(s, "\n<BODY>\n");
[[docUI eTextObj] writeHTML:s withTags:tags];
[theDocInfo writeHTMLTrailer:s];
NXPrintf(s, "\n</BODY>\n</HTML>\n");
NXSaveToFile(s, HTMLfile);
NXCloseMemory(s, NX_FREEBUFFER);

// PHASE IV: Notifications
{
    int count;
    count = [notificationList count];
    while (count--)
        if ([[notificationList objectAtIndex:count]
            respondsTo:@selector(writeComponentToPath:inFormat:)])
            [[notificationList objectAtIndex:count]
                writeComponentToPath:thePath inFormat:HTMD_FMT];
}

```



```

// PHASE V: Garbage Collection
// Iterate through the directory, deleting bastard objects
[self registerComponent:NXUniqueString(HTML_INDEX)];
[self registerComponent:NXUniqueString(".")] ;
[self registerComponent:NXUniqueString("..")] ;
dirp = opendir(thePath);
dp = readdir(dirp);
while (dp != NULL) {
    if (![componentTable isKey:NXUniqueString(dp->d_name)]) {
        sprintf(HTMLfile, "%s/%s", thePath, dp->d_name);
        removeFile(HTMLfile);
    }
    dp = readdir(dirp);
}
closedir(dirp);
[componentTable empty];
return self;
}

- saveETFD:(const char *) thePath changePath:(BOOL) changeIt
{
    NXAtom oldPath;
    char ETFFile[MAXPATHLEN];
    struct stat st;
    DIR *dirp;
    struct direct *dp;

    // PHASE I: Prepare thePath
    if (stat(thePath, &st) || (((st.st_mode & S_IFMT) != S_IFDIR))) {
        removeFile(thePath);
        if (mkdir(thePath, 0777)) {
            NXLogError("Could not mkdir(\"%s\")", thePath);
            perror("eText5");
            NXBeep();
            return nil;
        }
    }
    [componentTable empty];

    oldPath=[theDocInfo docPath];
    [theDocInfo setDocPath:thePath]; // this confuses saveTo:

    // PHASE II: Collect Annotations
    sprintf(ETFFile, "%s/TXT.rtf", thePath);
    [[docUI eTextObj] writeETFtoPath:ETFFile];

    // PHASE III: Notifications

```



```

- (eTDocUI *)docUI {                                     // Private
    return docUI;}

- touch {
    return [docUI touch];}

- (BOOL) needsSaving {
    return [docUI needsSaving];}

- (BOOL) acceptsAnnotation {
    NXSelPt a,b;

    [[docUI eTextObj] getSel:&a :&b];
    return ((a.cp >= 0) && [[docUI eTextObj] isEditable]);}

- insertAnnotation:theAnnotation {
    return [[docUI eTextObj] insertAnnotation:theAnnotation];}

- (BOOL) acceptsAgent {
    return theAgent ? NO : YES;}

- attachAgent:newAgent {
    if (![newAgent conformsTo:@protocol(Agent)]) return nil;
    theAgent = newAgent;
    [theDocInfo setAgent:NXUniqueString([theAgent name])];
    return [docUI attachAgent:theAgent];}

- detachAgent {
    [theDocInfo setAgent:NXUniqueString(")];
    [docUI detachAgent];
    theAgent = nil;
    return self;
}

- selectedObj {
    return selectedObj;}

- setSelectedObj:it {
    selectedObj = it;
    return self;}

- makeVisible {
    [[docUI window] makeKeyAndOrderFront:self];
    return self;}

- inspect {
    selectedObj = theDocInfo;
    return [inspector inspect:selectedObj];}

```

@end