

```
//
// FILENAME: eTAppUI.m
// SUMMARY: Implementation of eTAppUI, which works with eTApp and Appkit
// SUPERCLASS: Object:eTAppUI
// PROTOCOLS: None
// INTERFACE: eText.nib
// AUTHOR: Rohit Khare
// COPYRIGHT: ©1993,94 California Institute of Technology, eText Project
//
// Implementation Comments
// The nib has a lot of the intelligence hidden in its connections
// The main challenge is managing the UI, which is constantly changing.
//
// HISTORY
// 02/04/94: Added remote printing support.
// 11/20/94: Added History Panel UI.
// 10/29/94: Added Line Range hooks for modified MiscFindPanel.
// 10/22/94: Added Find Panel UI; from MiscFindPanelClass.
// 09/21/94: Revamped for eText5; removed savePanel, clean interface to eTApp
// 07/06/94: Added mailDevelopers:
// 05/21/94: Added savePanel code for multiple-format input.
// 05/05/94: Added "Accessory" support for future expansion
// 08/22/93: Added Tools UI.
// 08/16/93: Created. Implicitly supports an <eTextServices> protocol
//
// Imported Interfaces
//
// #import "eTAppUI.h"
//
// Hacked NXApp mainWindow accessor
//
@implementation Application (realMainWindow)
// the mainWindow method returns nil when the app is not focused.
- realMainWindow {return mainWindow;}
@end

@implementation eTAppUI
//
// AppKit Delegate
//
- appWillInit:sender {
    static BOOL hasInitted = NO;
    if (hasInitted) return self;

    infoController = nil;
    annotationTable = [[HashTable alloc] initWithKeyDesc:@"%"];
    agentTable = [[HashTable alloc] initWithKeyDesc:@"%"];
}
```

[illegible]

[illegible]

```

i = 0;
while (files && files[i]){
    strcpy(path, directory);
    strcat(path, "/");
    strcat(path, files[i]);
    [self openFromPath:NXUniqueString(path)
     type:NXUniqueString(rindex(path, '.')+1)];
    i++;
}
return self;
}

- registerAnnotation:(NXAtom) menuLabel
    key:(char) key
    name:(NXAtom) name
    icon:(NXImage *) icon {
static id _submenuTable=nil;

if ([annotationTable isKey:menuLabel]) return self;
if (!_submenuTable)
    _submenuTable = [[HashTable alloc] initWithKeyDesc:@"%"];

if (index(menuLabel, '/') && (*menuLabel != '/')) { // is there a slash?
    char tmp[256]; // C Programmer's Disease
    id _submenu;
    NXAtom tmpA;

    strncpy(tmp, menuLabel, 256);
    if (index(tmp, '/')) *index(tmp, '/') = 0; // cut out the prefix
    tmpA = NXUniqueString(tmp);
    _submenu = [_submenuTable valueForKey:tmpA];
    if (!_submenu) {
        _submenu = [[Menu alloc] initWithTitle:tmpA];
        [annotationMenu setSubmenu:_submenu forItem:
         [annotationMenu addItem:tmpA
          action:@selector(submenuAction:)
          keyEquivalent:'\0']];
        [_submenuTable insertKey:tmpA value:_submenu];
    }
    [[_submenu addItem: (index(menuLabel, '/')+1)
     action: @selector(annotationAction:)
     keyEquivalent: key]
     setUpdateAction:@selector(menuItemUpdate:) forMenu:_submenu];
    [annotationTable insertKey:(index(menuLabel, '/')+1) value:(STR)name];
} else {
    [[annotationMenu addItem: menuLabel
     action: @selector(annotationAction:)]

```

[illegible]

```

char          path[MAXPATHLEN];
static BOOL firstTime=YES;

id openpanel = [[OpenPanel new] allowMultipleFiles:YES];
[openpanel setTreatsFilePackagesAsDirectories:NO];
[openpanel chooseDirectories:NO];

if (firstTime) {
    docdir = [userModel stringQuery:ETFDIRECTORY];
    if (docdir && *docdir) {
        [openpanel setDirectory:docdir];
    }
    firstTime = NO;
}
i = [openpanel runModalForTypes:etfdType];
if (!i) return self;
files = [openpanel filenames];
directory = [openpanel directory];
i = 0;
while (files && files[i]){
    strcpy(path, directory);
    strcat(path, "/");
    strcat(path, files[i]);
    [self openFromPath:NXUniqueString(path)
     type:rindex(path, '.') ? NXUniqueString(rindex(path, '.')+1) : ""];
    i++;
}
return self;
}

```

```

- openAsText:sender {
    const char *directory;
    const char *const *files;
    const char *textType[1] = {NULL};
    int          i;
    char          path[MAXPATHLEN];

    id openpanel = [[OpenPanel new] allowMultipleFiles:YES];
    [openpanel setTreatsFilePackagesAsDirectories:YES];
    [openpanel chooseDirectories:NO];
    i = [openpanel runModalForTypes:textType];
    if (!i) return self;
    files = [openpanel filenames];
    directory = [openpanel directory];
    i = 0;
    while (files && files[i]){
        strcpy(path, directory);

```

```

        strcat(path, "/");
        strcat(path, files[i]);
        [self openFromPath:NXUniqueString(path)
         type:NXUniqueString("")];
        i++;
    }
    return self;
}

- openFromPath:(const char *)path type:(const char *)aType {
    id theDocInfo;

    if(!strcmp(aType, ETFD_EXT)) {
        [etApp openID:[[[[eTDocInfo alloc] init]
                        readComponentFromPath:NXUniqueString(path)]
                        docID]];

    } else if(!strcmp(aType, LINK_EXT)) {
        char tmp[64];
        long dID, aID;
        id theTable = [[[NXStringTable alloc] init] readFromFile:path];

        strncpy(tmp, [theTable valueForKey:DOCID], 63);
        sscanf(tmp, "%x", &dID);
        [etApp openID:dID];

        strncpy(tmp, [theTable valueForKey:ANCHORID], 63);
        theTable = [[theTable empty] free];
        sscanf(tmp, "%x", &aID);
        // This is a really severe kernel violation!
        [[[objc_lookUpClass("eTBookmarkBinder") new]
         bookmarkForID:aID] highlight:self];

    } else if((!strcmp(aType, HTMD_EXT)) || (!strcmp(aType, OLD_HTMD_EXT))) {
        char bounce[MAXPATHLEN];

        sprintf(bounce, "%s/"HTML_INDEX, path);
        [[Application workspace] openFile:bounce];

    } else if(!strcmp(aType, URI_EXT)) {
        NXStream *s;
        char *data, *url;
        Pasteboard *thePB;
        int len, maxlen;

        s = NXMapFile(path, NX_READONLY);
        NXGetMemoryBuffer(s, &data, &len, &maxlen);
    }
}

```

```

if (!strcmp(data, "URL:",4)) url = data+4;
    else if (!strcmp(data, "<URL:",5)) url = data+5;
    else url = data;

thePB = [Pasteboard newUnique];
[thePB declareTypes:&NXAsciiPboardType num:1 owner:nil];
[thePB writeType:NXAsciiPboardType data:(url) length:strlen(url)];

if (!NXPerformService("Open URL", thePB) &&
    !NXPerformService("OmniWeb/Open URL", thePB) &&
    !NXPerformService("SpiderWoman/Open URL", thePB))
    NXRunAlertPanel("eTApp", "Could not open a "URI_EXT" file (using
services): %s.", "OK", NULL, NULL, url);

NXCloseMemory(s, NX_FREEBUFFER);

} else if (!strcmp(aType, TeXD_EXT)) {
    [navigator find:"\\.tex$" inDir:(STR)path
        target:[Application workspace] action:@selector(openFile:)];

} else if ((!strcmp(aType, DOCI_EXT)) || (!strcmp(aType, NAVI_EXT))) {
    // Create an eTDocInfo, without opening the document
    [etApp loadDocInfoFromPath:path];
    // Navigator will redisplay on the next autoupdate

} else if ((!strcmp(aType, AGNT_EXT)) || (!strcmp(aType, TOOL_EXT)) ||
    (!strcmp(aType, BNDL_EXT)) || (!strcmp(aType, ANNT_EXT)) ||
    (!strcmp(aType, DSBD_EXT)) || (!strcmp(aType, ACCS_EXT))) {
    [etApp loadToolFromPath:path];

} else { // Pass along a virgin docInfo with the path (RTFD, RTF, or text).
    char tmp[MAXPATHLEN], *ext;

    theDocInfo = [[eTDocInfo alloc] init];
    // set the doc's title to be the old name (foo)
    strcpy(tmp, rindex(path, '/')+1);
    ext = rindex(tmp, '.');
    if (ext && ext+1 && !strcmp(ext+1, RTFD_EXT) && !strcmp(ext+1, RTF_EXT))
        *ext = 0;
    [theDocInfo setDocTitle:tmp];
    [theDocInfo setDocPath:path];
    [etApp openID:[theDocInfo docID]];

    // now that everything is hunky-dory, set a NEW pathname and touch it
    // set the path to be /directory/foo/was/in/foo.id.etfd
    strcpy(tmp, path);
    ext = rindex(tmp, '.');

```



```

    if (ext && ext+1 && !strcmp(ext+1, RTFD_EXT) && !strcmp(ext+1, RTF_EXT))
        *ext = 0;
    strcat(tmp, ".ETFD_EXT");
    while (!access(tmp, F_OK)) { // hey, the odds are 1 in 2^32, but...
        sprintf(rindex(tmp, '.'), ".%x.ETFD_EXT", [NXApp uniqueID]);
    }
    [theDocInfo setDocPath:tmp];
    [[theDocInfo etDoc] touch];
}
return self;
}

```

```

- import:sender {
    const char *directory;
    const char *const *files;
    NXAtom      *importTypes;
    int          i;
    char         path[MAXPATHLEN];
    static id    thePboard=nil;
    id           openpanel;
    NXAtom       filterHack;

    // Transparency violation
    if (![[[NXApp mainWindow] delegate] etDoc] acceptsAnnotation]){
        NXBeep(); return nil;
    }

    openpanel = [[OpenPanel new] allowMultipleFiles:YES];
    [openpanel setTreatsFilePackagesAsDirectories:NO]; // tough call!
    [openpanel chooseDirectories:NO];

    // see save panel docs. Assumes there is always an ANY_TYPE handler
    filterHack = NULL;
    importTypes = &filterHack;

    i = [openpanel runModalForTypes:importTypes];
    if (!i) return self;
    files = [openpanel filenames];
    directory = [openpanel directory];
    i = 0;
    if (!thePboard) thePboard = [Pasteboard newUnique];
    while (files && files[i]){
        id it;

        strcpy(path, directory);
        strcat(path, "/");
        strcat(path, files[i]);
    }
}

```


[illegible]


```

    } else {
        return NO;
    }
}

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

    if (action == @selector(saveAll:)) {
        enable = [etApp validateSaveAll];
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    }
    if ((action == @selector(annotationAction:)) ||
        (action == @selector(import:))) {
        id foo = [[[NXApp mainWindow] delegate] etDoc];
        enable = foo && [foo acceptsAnnotation];
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    }
    if (action == @selector(agentAction:)) {
        enable = [[[NXApp mainWindow] delegate] etDoc]? YES : NO;
        if ([menuCell isEnabled] != enable) {
            [menuCell setEnabled:enable];
            return YES;
        } else {
            return NO;
        }
    }
    return NO;
}

```

```

void initMenu(Menu *menu)

```

```

/*

```

```

 * Sets the updateAction for every menu item which sends to the
 * First Responder (i.e. their target is nil). When autoupdate is on,
 * every event will be followed by an update of each of the menu items

```

```

* which is visible. This keep all unavailable menu items dimmed out
* so that the user knows what options are available at any given time.
* Returns the activate menu if is found in this menu.
*/
{
    int count;
    Matrix *matrix;
    MenuCell *cell;
    id matrixTarget, cellTarget;

    matrix = [menu itemList];
    [menu setAutoupdate:YES];
    matrixTarget = [matrix target];

    count = [matrix cellCount];
    while (count--) {
        cell = [matrix cellAt:count :0];
        cellTarget = [cell target];
        if (!matrixTarget && !cellTarget) {
            [cell setUpdateAction:@selector(menuItemUpdate:) forMenu:menu];
        } else if ([cell hasSubmenu]) {
            initMenu(cellTarget);
        }
    }
}

- annotationAction:sender
{
    if (![NXApp mainWindow delegate] etDoc])
        return nil;
    if ([NXApp mainWindow delegate] etDoc] acceptsAnnotation]) {
        id it;

        it = [[etApp annotationByName:
                [annotationTable valueForKey:
                 NXUniqueString([sender selectedCell title])]
                ]
              alloc];

        if ([it respondsTo:@selector(initFromPboard:inDoc:linked:)]) {
            [it initFromPboard:nil
             inDoc:[NXApp mainWindow delegate] etDoc]
            linked:NO];
            [[NXApp mainWindow delegate] etDoc] insertAnnotation:it];
        } else {
            it = [it free];
            NXLogError("Annotation did not respond to initFromPboard::");
        }
    }
}

```

```

        NXBeep();
    }
}
return self;
}

- agentAction:sender
{
    if (![NXApp mainWindow delegate] etDoc)
        return nil;
    [[[NXApp mainWindow] delegate] etDoc]
        attachAgent:
        [[[etApp agentByName:
            [agentTable valueForKey:NXUniqueString([sender selectedCell title])]]
            alloc] init:[[[NXApp mainWindow] delegate] etDoc] docInfo]]];
    return self;
}

@end

```