


```

        NXSoundPboardType,
        NXPostScriptPboardType,
        NXTIFFPboardType,
        NX3DRIBPboardType,
        NXColorPboardType,
        NXFontPboardType,
        NXRulerPboardType,
        NXRTFPboardType,
        NXAsciiPboardType,
        NXTabularTextPboardType,
        NULL};
NXAtom      docdir;
char        thePath[MAXPATHLEN];

// PHASE I: Internal initialization
[super init];
docTable = [[HashTable alloc] initWithKeyDesc:"i"];
agentTable = [[HashTable alloc] initWithKeyDesc:@"%"];
annotationTable = [[HashTable alloc] initWithKeyDesc:@"%"];
typesTable = [[HashTable alloc] initWithKeyDesc:@"%"];
[[NXApp printInfo] setHorizPagination:NX_FITPAGINATION];
[[NXApp printInfo] setMarginLeft:36.0 right:36.0 top:36.0 bottom:36.0];

// PHASE II: Global initialization
etApp = self;
etAppUI = [NXApp delegate]; // relies on Nib setting.
inspector = [[Inspector alloc] init];
userModel = [[UserModel alloc] init];
navigator = [[Navigator alloc] init];
[NXApp registerServicesMenuSendTypes:supportedTypes
                    andReturnTypes:supportedTypes];

// PHASE III: Locate and load tools and document info
// First, activate the classes compiled into the Kernel
{
    const char *internalTools[12] = {"eTImage", "eTAudio", "eTLink",
        "eTBookmark", "Divider", "eTFileLink", "eTLiteral", "eTNote",
        "eTURLink", "HotLinks", "MailTo", NULL};
    int i;

    for(i=0; internalTools[i]; i++)
        [objc_lookUpClass(internalTools[i]) toolAwake:self];
}

// Create a Library directory, public_html and set the defaults.

sprintf(thePath, "%s/Library/eText/", NXHomeDirectory());

```

[illegible]

[illegible]

```

- openID:(long) docID {
    id theDoc;
    id theDocInfo;

    if ([docTable isKey:(void *)docID]) {
        [(id) [docTable valueForKey:(void *)docID] makeVisible];
        return nil;
    }
    theDocInfo = [eTDocInfo findDocInfo:docID];
    if (!theDocInfo) {
        theDocInfo = [[eTDocInfo alloc] init];
        docID = [theDocInfo docID];
    } else if (![theDocInfo isVirgin] &&
        access([theDocInfo docPath], R_OK|X_OK)) {
        int choice=NXRunAlertPanel("Invalid docPath",
            "Did not find %s at the path %s.",
            "Create New Document", "Cancel", NULL,
            [theDocInfo docTitle], [theDocInfo docPath]);
        if (choice == NX_ALERTALTERNATE) return nil;
    }
    theDoc = [[eTDoc alloc] initWithDocInfo:theDocInfo];
    if (!theDoc) return nil;
    [docTable insertKey:(void *)docID value:theDoc];
    return theDoc;
}

- closeID: (long) docID {
    id theDoc;

    if (![docTable isKey:(void *)docID])
        return nil;
    theDoc = [docTable valueForKey:(void *)docID];
    [theDoc free];
    [[eTDocInfo findDocInfo:docID] free];
    [docTable removeKey:(void *)docID];
    return self;
}

- (long) createID {
    char newPath[MAXPATHLEN];
    eTDocInfo *theDocInfo;
    static int count=0;
    const char *docdir=[userModel stringQuery:ETFDIRECTORY];

    theDocInfo = [[eTDocInfo alloc] init];
    do { sprintf(newPath, "%s/Untitled-%d.%s",
        (docdir && *docdir) ? docdir : NXHomeDirectory(),
        count++, ETFD_EXT);
    } while (!access(newPath, F_OK));
}

```

```
[theDocInfo setDocPath: NXUniqueString(newPath)];  
[self openID:[theDocInfo docID]];  
return [theDocInfo docID];  
}  
  
- documentByID: (long) docID {return [docTable valueForKey:(void *)docID];}  
  
//DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD  
// Global Operations  
//  
  
- saveAll {  
    long    key;  
    id      doc;  
    NXHashState state = [docTable initState];  
  
    while ([docTable nextState: &state key:(void *)&key value:(void *)&doc])  
        [doc save:self];  
    return self;  
}  
  
- shutdown:(BOOL) cancellable {  
    BOOL     dirtyDocs;  
    long     key;  
    id       theDoc;  
    int      choice;  
    NXHashState state = [docTable initState];  
  
    recheck: // label. Cycle until all docs handled.  
    dirtyDocs = NO;  
    state = [docTable initState];  
    while (!dirtyDocs &&  
        [docTable nextState: &state key:(void *)&key value:(void *)&theDoc]  
        dirtyDocs != [theDoc needsSaving];  
  
if (dirtyDocs) {  
    if (cancellable)  
        choice=NXRunAlertPanel("Quit",  
                               "There are unsaved files",  
                               "Review Unsaved", "Quit Anyway", "Cancel");  
    else choice=NXRunAlertPanel("Logout/Power-Off",  
                                "There are unsaved files",  
                                "Review Unsaved", "Quit Anyway", NULL);  
    if (choice == NX_ALERTDEFAULT){  
        state = [docTable initState];  
        while([docTable nextState:&state  
            key:(void *)&key value:(void *)&theDoc])  
            [theDoc close:self allowCancel:cancellable];  
    } else if (choice == NX_ALERTOTHER)  
        return nil;  

```

[illegible]

```

        dirtyDocs |= [theDoc needsSaving];
    return dirtyDocs;
}
- loadToolFromPath:(const char*)path {
    id theBundle = [[NXBundle alloc] initWithDirectory:NXUniqueString(path)];
    if ([[theBundle principalClass] conformsTo:@protocol(Tool)])
        [[theBundle principalClass] toolAwake:self];
    return self;
}
- loadDocInfoFromPath:(const char*)path {
    [[eTDocInfo alloc] initWithComponentFromPath:path];
    return self;
}
- docTable {
    return docTable;
}
@end

```