```
// Controller.m

#import "Controller.h"
#import "regex.h"
#import <strings.h>
#import <stdlib.h>

@implementation Controller

- init
{
	[super init];
	itemList = [[List alloc] init];
	return self;
}

- free
{
	[itemList free];
	return [super free];
}

- appDidInit:sender
{
	static NXDefaultsVector RTFSyntaxDefaults = {
		{"configuration", ""},
		{NULL}
	};

	// make this object the services delegate
	[[NXApp appListener] setServicesDelegate:self];
```

```
    // setup text objects
    [regularExpression setMonoFont:YES];
    [matchedSampleText setMonoFont:YES];
    [unmatchedSampleText setMonoFont:YES];

    // register defaults
    NXRegisterDefaults("RTFSyntax", RTFSyntaxDefaults);

    // read defaults
    [defaultConfiguration setStringValue:NXGetDefaultValue("RTFSyntax", "configuration")];

    // open the default configuration (or a new one if there is no default)
    if (!fileOpened && ![self openDefaultConfiguration])
        [self newConfiguration:self];

    return self;
}

- (int)app:sender openFile:(const char *)path type:(const char *)type
{
    if (strcmp(type, "syntax"))
        return NO;

    [self openNamedConfiguration:path];
    [configurationWindow makeKeyAndOrderFront:self];
    fileOpened = YES;

    return YES;
}

- (BOOL)appAcceptsAnotherFile:(Application *)sender
{
    return YES;
```

```
}

- previousItem:sender
{
    int i = [itemNumber intValue];
    if (i <= 1)
        return self;
    [self saveCurrentItem];
    [self displayItem:i-1];
    return self;
}

- nextItem:sender
{
    int i = [itemNumber intValue];

    if (i >= [numberOfItems intValue])
        return self;
    [self saveCurrentItem];
    [self displayItem:i+1];
    return self;
}

- deleteItem:sender
{
    int i = [itemNumber intValue];

    if ([numberOfItems intValue] == 1)
        return self;    // can't delete last item (need at least one)
    [[itemList removeObjectAt:i-1] free];
    [numberOfItems setIntValue:[numberOfItems intValue]-1];
    if (i == 1)
```

```
        [self displayItem:1];
    else
        [self displayItem:i-1];
    return self;
}

- insertItem:sender
{
    int i = [itemNumber intValue];

    [self saveCurrentItem];
    [itemList insertObject:[[SyntaxItem alloc] init] at:i-1];
    [numberOfItems setIntValue:[numberOfItems intValue]+1];
    [self displayItem:i];
    return self;
}

- (void)saveCurrentItem
{
    SyntaxItem *item = [itemList objectAt:[itemNumber intValue]-1];

    [item setRegularExpression:regularExpression];
    [item setMatchedSampleText:matchedSampleText];
}

- (void)displayItem:(int)i
{
    SyntaxItem *item = [itemList objectAt:i-1];

    [itemNumber setIntValue:i];

    if ([item getRegularExpression]) {
```

```
        [regularExpression readText:[[item getRegularExpression] stream]];
        [regularExpression setFont:[[item getRegularExpression] font]];
        [regularExpression display];
    } else
        [regularExpression setText:""];

    if ([item getMatchedSampleText]) {
        [matchedSampleText readText:[[item getMatchedSampleText] stream]];
        [matchedSampleText setFont:[[item getMatchedSampleText] font]];
        [matchedSampleText setTextColor:[[item getMatchedSampleText] textColor]];
        [matchedSampleText display];
        [matchedColorWell setColor:[matchedSampleText textColor]];
    } else
        [matchedSampleText setText:""];
}

- (BOOL)openDefaultConfiguration
{
    if (![defaultConfiguration stringValue] || !strlen([defaultConfiguration stringValue])) {
        [configurationWindow setTitle:""];
        return NO;
    }
    // load default configuration if not already loaded
    if (strcmp([defaultConfiguration stringValue], [configurationWindow title]))
        [self openNamedConfiguration:[defaultConfiguration stringValue]];
    return YES;
}

- openConfiguration:sender
{
    const char *const types[2] = {"syntax", NULL};
    id openPanel = [OpenPanel new];
```

```
    [openPanel allowMultipleFiles:NO];
    if (![openPanel runModalForTypes:types])
        return self;
    [self openNamedConfiguration:[openPanel filename]];
    [configurationWindow makeKeyAndOrderFront:self];
    return self;
}

- (void)openNamedConfiguration:(const char *)name
{
    NXTypedStream *stream;
    Text *text;
    NXColor color;

    [configurationWindow setTitle:name];
    if (!(stream = NXOpenTypedStreamForFile(name, NX_READONLY))) {
        NXRunAlertPanel("Error", "Can't open %s.", NULL, NULL, NULL, name);
        [self newConfiguration:self];
        return;
    }

    text = NXReadObject(stream);
    [unmatchedSampleText readText:[text stream]];
    [unmatchedSampleText setFont:[text font]];
    color = NXReadColor(stream);
    [unmatchedSampleText setTextColor:color];
    [unmatchedSampleText display];
    [unmatchedColorWell setColor:color];
    [text free];

    [itemList free];
    itemList = NXReadObject(stream);
    NXCloseTypedStream(stream);
```

```
    [numberOfItems setIntValue:[itemList count]];
    if ([itemList count] < 1) {
        NXRunAlertPanel("Error", "No items in configuration file", NULL, NULL, NULL);
        return;
    }
    [self displayItem:1];
    return;
}

- newConfiguration:sender
{
    [configurationWindow makeKeyAndOrderFront:self];
    [itemList freeObjects];
    [itemList addObject:[[SyntaxItem alloc] init]];
    [numberOfItems setIntValue:1];
    [configurationWindow setTitle:"Untitled"];
    [self displayItem:1];
    [unmatchedSampleText setText:"unmatched text"];
    return self;
}

- saveConfiguration:sender
{
    NXTypedStream *stream;

    [configurationWindow makeKeyAndOrderFront:self];
    if (!strcmp([configurationWindow title], "Untitled"))
        [self getConfigurationName];
    if (!(stream = NXOpenTypedStreamForFile([configurationWindow title], NX_WRITEONLY))) {
        NXRunAlertPanel("Error", "Can't open %s.", NULL, NULL, NULL, [configurationWindow title]);
        return self;
    }
    [self saveCurrentItem];
```

```
    NXWriteRootObject(stream, unmatchedSampleText);
    NXWriteColor(stream, [unmatchedSampleText textColor]);
    NXWriteObject(stream, itemList);
    NXCloseTypedStream(stream);

    return self;
}

- revertToSavedConfiguration:sender
{
    [configurationWindow makeKeyAndOrderFront:self];
    [self openNamedConfiguration:[configurationWindow title]];
    return self;
}

- (BOOL)getConfigurationName
{
    id savePanel = [SavePanel new];

    [savePanel setRequiredFileType:"syntax"];
    if (![savePanel runModalForDirectory:NULL file:NULL])
        return NO;
    [configurationWindow setTitle:[savePanel filename]];
    return YES;
}

- saveAsConfiguration:sender
{
    if ([self getConfigurationName])
        [self saveConfiguration:self];
    return self;
}
```

```
- setDefaultConfigurationName:sender
{
    const char *const types[2] = {"syntax", NULL};
    id openPanel = [OpenPanel new];

    [openPanel allowMultipleFiles:NO];
    if (![openPanel runModalForTypes:types])
        return NO;
    [defaultConfiguration setStringValue:[openPanel filename]];
    NXWriteDefault("RTFSyntax", "configuration", [openPanel filename]);
    return self;
}

- setMatchedColor:sender
{
    [matchedSampleText setTextColor:[matchedColorWell color]];
    [matchedSampleText display];
    return self;
}

- setUnmatchedColor:sender;
{
    [unmatchedSampleText setTextColor:[unmatchedColorWell color]];
    [unmatchedSampleText display];
    return self;
}

#define MAIL_ADDRESS "rgc@cs.umd.edu"
#define SUBJECT "Suggestion/Bug for RTFSyntax"

- suggestion:sender
```

```
{
    id s = [NXApp appSpeaker];

    NXPortFromName("Mail", NULL); // make sure app is launched
    [[NXApp appSpeaker] setSendPort:NXPortFromName("MailSendDemo", NULL)];
    [s performRemoteMethod:"setTo:" with:MAIL_ADDRESS length:strlen(MAIL_ADDRESS)+1];
    [s performRemoteMethod:"setSubject:" with:SUBJECT length:strlen(SUBJECT)+1];
    return self;
}

#define ERRBUF 100

- colorSyntax:pasteboard userData:(const char *)userData error:(char **)message
{
    NXAtom typelist[1];
    const NXAtom *types;
    NXStream *stream;
    Text *text;

    for (types = [pasteboard types]; *types; types++)
        if (*types == NXRTFPboardType)
            break;
    if (*types == 0) {
        *message = "Could not find RTF pasteboard type";
        NXRunAlertPanel("Error", *message, 0, 0, 0);
        return nil;
    }

    if (![self openDefaultConfiguration]) {
        *message = "No default configuration.";
        NXRunAlertPanel("Error", *message, 0, 0, 0);
        return nil;
    }
```

```
    // read the rich text from the pasteboard stream and save it in an internal text object
    if (!(stream = [pasteboard readTypeToStream:NXRTFPboardType])) {
        *message = "Could not read RTF data";
        NXRunAlertPanel("Error", *message, 0, 0, 0);
        return nil;
    }
    text = [[Text alloc] init];
    [text setMonoFont:NO];
    NXSeek(stream, 0L, NX_FROMSTART);
    [text readRichText:stream];
    NXCloseMemory(stream, NX_FREEBUFFER);

    [self colorRTF:text];

    // copy results to the pasteboard
    typelist[0] = NXRTFPboardType;
    [pasteboard declareTypes:typelist num:1 owner:self];
    if (!(stream = NXOpenMemory(NULL, 0, NX_READWRITE))) {
        NXRunAlertPanel("Error", "Can't open memory file.", 0, 0, 0);
        [text free];
        return self;
    }
    [text writeRichText:stream];
    if (![pasteboard writeType:NXRTFPboardType fromStream:stream])
        NXRunAlertPanel("Error", "Can't write results to pasteboard.", 0, 0, 0);
    NXCloseMemory(stream, NX_FREEBUFFER);
    [text free];

    return self;
}
```

- (void)colorRTF:(Text *)text

```
/*
Description:
    Color the given text object using the default configuration.
    Uses GNU's regex library (POSIX syntax).
*/
{
    char errbuf[ERRBUF];
    typedef struct {
        regex_t re;
        int begin, end;         // offsets
        BOOL compute;           // YES => compute match
        BOOL done;              // YES => no more matches
    } Match;
    Match *match;
    char *s;
    int length;
    int i;
    int skip = 0;
    int errcode;

    // first set text to be like the unmatched sample text
    length = [text textLength];
    [text setSel:0    :length];
    [text setSelColor:[unmatchedSampleText textColor]];
    [text setSelFont:[unmatchedSampleText font]];
    [text selectNull];

    // allocate the text string and get it
    if (!(s = malloc(length + 1))) {
        NXRunAlertPanel("Error", "malloc() failed.", 0, 0, 0);
        return;
    }
    if ([text getSubstring:s start:0 length:length] <= 0) {
```

```
        NXRunAlertPanel("Error", "Can't get substring.", 0, 0, 0);
        free(s);
        return;
    }
    s[length] = 0;

    // allocate the match array
    if (!(match = malloc(sizeof(Match) * [itemList count]))) {
        NXRunAlertPanel("Error", "malloc() failed.", 0, 0, 0);
        free(s);
        return;
    }

    // compile all of the regular expressions
    for (i = 0; i < [itemList count]; i++) {
        id item = [itemList objectAt:i];
        char *reText = [item getRegularExpressionText];
        int reLength = strlen(reText);
        int j, k;

        // remove newlines
        for (j = 0, k = 0; k < reLength; k++)
            if (reText[k] != '\n')
                reText[j++] = reText[k];
        reText[j] = '\0';
        reLength = j;

        // convert "\n"'s to newlines
        for (j = 0, k = 0; k < reLength; k++)
            if (reText[k] != '\\')
                reText[j++] = reText[k];
            else if (k+1 == reLength)
                reText[j++] = '\\';
```

```
            else if (reText[k+1] == 'n') {
                reText[j++] = '\n';
                k++;
            } else {
                reText[j++] = '\\';
                reText[j++] = reText[k+1];
                k++;
            }
        reText[j] = '\0';
        reLength = j;

        // make sure regular expression is not empty
        if (!reLength) {
            NXRunAlertPanel("Error", "Regular expression %d is empty.", 0, 0, 0, i+1);
            goto cleanup;
        }

        // compile regular expression
        if ((errcode = regcomp(&match[i].re, reText, REG_EXTENDED | REG_NEWLINE))) {
            regerror(errcode, &match[i].re, errbuf, ERRBUF);
            NXRunAlertPanel("Error", "regcomp() failed -- %s", 0, 0, 0, errbuf);
            goto cleanup;
        }
        match[i].compute = YES;
        match[i].done = NO;
    }

    for (;;) {
        int j;              // index of match[] to use

        // compute all regular expressions that need it
        for (i = 0; i < [itemList count]; i++)
            if (!match[i].done && match[i].compute) {
```

```
            regmatch_t rm;
            if (regexec(&match[i].re, s + skip, 1, &rm, 0))
                match[i].done = YES;
            else {
                match[i].begin = rm.rm_so + skip;
                match[i].end = rm.rm_eo + skip;
                match[i].compute = NO;
            }
        }

// find the first (earliest) match using all regular expressions
// if two RE's find match at the same starting offset, choose the longer match
// store the result in j
for (i = 0, j = 0; i < [itemList count]; i++)
        if (!match[i].done) {
            j = i;
            break;
        }
if (i == [itemList count])              // no matches
        break;
for (++i; i < [itemList count]; i++)
        if (!match[i].done && (match[j].begin > match[i].begin ||
            (match[j].begin == match[i].begin && match[j].end < match[i].end)))
            j = i;

// reset all matches that started before match[j].end
for (i = 0; i < [itemList count]; i++)
        if (!match[i].done && match[i].begin < match[j].end)
            match[i].compute = YES;

// color the matching text
// note that these 4 lines take up about 70% of the program's CPU time
[text setSel:match[j].begin :match[j].end];
```

```
            [text setSelColor:[[[itemList objectAt:j] getMatchedSampleText] textColor]];
            [text setSelFont:[[[itemList objectAt:j] getMatchedSampleText] font]];
            [text selectNull];

            skip = match[j].end;
        }

cleanup:
        // free the compiled regular expressions and other stuff
        for (i = 0; i < [itemList count]; i++)
            regfree(&match[i].re);
        free(match);
}

- showInfo:sender
{
    if (!infoPanel)
        infoPanel = [NXApp loadNibSection:"InfoPanel.nib" owner:self];
    [infoPanel makeKeyAndOrderFront:self];
    return self;
}

@end
```