Strings Programming Guide for Cocoa

イントロダクション

このドキュメントでは，CocoaにおけるStringオブジェクトの作りかた，探しかた，つなぎかた，そして描画のしかたを解説します。加えてStringに関連の深い CharacterSetオブジェクト，それからScannerオブジェクトについての説明もしています。

注意！
NSStringオブジェクトはObjective-Cだけにあるクラスです。JavaでCocoaをお使いの皆さんは，java.lang.string を使ってください。
NSScannerオブジェクトもObjective-Cだけにあるクラスです。同じくJavaでは，NSFormatterを使ってください。NSFormatterについては「Data Formatting Programming Guide for Cocoa」に解説があります。

Stringオブジェクト

StringオブジェクトはCocoaフレームワークにおける文字列処理のために用意されています。文字列をオブジェクトの形にしておくことで，他のオブジェクトと同じように文字列を扱えますし，カプセル化による隠蔽効果も期待できます。Stringオブジェクトにはどんなエンコーディングも使えますし，格納も単なる文字列の配列よりも効率的です。

Stringオブジェクトはユニコードキャラクタの配列（言い換えれば「テキストストリング」ですが）として実装されています。変更不能なStringは作成後に変更することができません。これはNSStringクラスのインスタンスとして生成，使用します。変更可能なStringオブジェクトが必要な場合は，NSMutableStringクラスのインスタンスを生成してください。

NSStringおよびNSMutableStringのインスタンスを通常Stringオブジェクト（時には単にString）と呼びます。標準的なchar*型で定義される文字列は「C string」という用語で表します。

Stringオブジェクトはユニコードキャラクタ（unichar型）の配列です。プログラマは `lengh` メソッドを使ってStringオブジェクトが保持するキャラクタ数を，また `characterAtIndex:` メソッドを使って任意の位置のキャラクタが何かを調べることができます。これら2つのメソッドがStringオブジェクトに対する基本的なアクセスメソッドです。

とはいえ，実際にプログラム中でStringオブジェクトを使う場合ーString同士を比較したり，一方の中にもう一方を探したり，2つのStringを結合して新しいStringオブジェクトを作るといった処理を行うにはもっと高いレベルのメソッドが必要です。また，もしStringオブジェクトを構成する個々のキャラクタにアクセスする必要があるのなら，プログラマはユニコードキャラクタ・エンコーディングについての理解が必要となります。詳しくは，以下を参照してください。

- The Unicode Standard, Version 4.0. The Unicode Consortium. Boston: Addison-Wesley, 2003. ISBN0-321-18578-1.
- The Unicode Consortium web site: http://www.unicode.org/.

Character Setオブジェクト

NSCharacterSetオブジェクトは，一連のユニコードキャラクタをセットとして扱うために用意されています。NSString，およびNSScannerオブジェクトは，検索処理のためにキャラクタを分類するのにNSCharacterオブジェクトを使用します。これを使えば，特定の性質をもつキャラクタを容易に見つけ出すことができるからです。
Character SetにもStringと同じように，生成後は変更のできないNSCharacterSetと変更可能なNSMutableCharacterSetがあり，必要に応じて使い分けることができます。

Character Setオブジェクトは，文字列処理に際してフィルタになるなど，単に一連のキャラクラタを保持しているだけのオブジェクトです。それ自体がなにかの処理を行うことはありません。NSString，およびNSScannerクラスは，ある種のキャラクタを発見するためのメソッドで，このオブジェクトを引数として受け取ります。

例を挙げましょう。以下のコードは，myStringというStringオブジェクトにおいて最初の大文字からなるブロックを検出します。

```
NSString *myString = @"some text in an NSString...";
NSRange letterRange;

letterRange = [myString
       rangeOfCharacterFromSet:[NSCharacterSet uppercaseLetterCharacterSet]];
```

このコードの実行後，letterRange.locationは「NSString」の最初の「N」の位置を指します。上の例で，もしmyStringの最初の文字「s」が大文字だったら，letterRange.locationは0になったわけです。

Character Setオブジェクトを使う処理は，時として実行速度に予想外の影響を与えることがあります。使用に際しては常にそのことに留意してください。また，これは他の類似のオブジェクトにも言えることですが，変更可能なCharacter Setの使用は，そうでないSetに比べてはるかにコストがかかります。メモリの消費も大きいし，逆転（Character Setの逆転はStringをスキャンする場合にしばしば発生します）するのにも時間とメモリを要します。

従って，プログラマは以下のことに留意するべきです。
- できるだけ，変更可能なCharacter Setオブジェクトは作らない。
- 同じSetを繰り返し使うのであれば，グローバル変数にして使い回す。
- 変更可能なSetに変更を加えて用途に適したSetを作る場合も，一度出来上がったらそれを変更不能なSetにコピーし，元のオブジェクトは速やかに破棄する。もしセッションを超えてそれが使用されるなら，アプリケーションのメイン・バンドルにそれを保存して再利用する。
- Character Setオブジェクトの保存にあたってはアーカイブは使わずにファイルとして保存する。アーカイブはしばしば重複し，ディスクやメモリを無駄に浪費します。

Scannerオブジェクト

Scannerオブジェクトは、NSStringオブジェクトの構成要素である文字群を、数値と文字列も集合として解釈するために使用されます。プログラマはオブジェクトの生成時に解釈したいStringオブジェクトをScannerに渡します。そうして作られたScannerは、保持するStringオブジェクトの内容を必要に応じて解釈できるのです。このクラスタに属するのは唯一NSScannerクラスだけです。
最も一般的なScannerオブジェクトの作り方は `scannerWithString:` あるいは `localizedScannerWithString:` を使うものです。どちらも解釈対象となるStringオブジェクトをパラメータとして指定します。新たに作成されたScannerはパラメータのStringの先頭をポイントしており、呼び出されるScanメソッドはここから実行されます。また、`setScanLocation:` を使用してScanの開始位置を変更することもできます。
Scanオペレーションは基本的に直前のオペレーションで最後にScanした文字の次の文字から開始されます。例えば「137 small cases of bananas」という文字列から整数をScanした後では、Scannerの位置は3となり、文字「7」の次の空白文字をポイントしており、次回のオペレーションはこの文字から始まります。
Scannerオブジェクトは、オペレーションの要求に合致しない文字をスキップします。つまり整数を要求されている場合、0〜9の数字を見つけるまですべての文字が無視されるということです。また、Scannerはすべての空白文字と改行文字をデフォルトでスキップします。`setCharactersToBeSkipped:` メソッドを使用して、ある種の文字（群）をあらかじめスキップするよう設定することも可能です。
上のサンプルのScanを続け、`scanString:intoString` メソッドで「of」の前までを文字列として抜き出したとしましょう。その場合Scannerは「small」の直前の空白は無視しますが、「of」の直前の空白文字は結果の文字列に含んでいます。
このケースでの探索文字列と結果の関係を以下の表に示します。

検索文字列	結果文字列
“of”（直前に空白なし）	“small cases ”（末尾の空白を含んでいる）
“ of”（直前に空白あり）	“small cases”（末尾に空白を含まない）

また、`setCaseSensitive:` メソッドを使用して、Scanに当たってアルファベットの大文字／小文字の区別を行うかどうかも指定することが可能です。デフォルトではScannerは大文字／小文字を区別しません。ただしこの指定は検索に関するもので、スキップに関しては適用されないので注意してください。もし英語の母音をスキップしたい場合には `setCharactersToBeSkipped:` で「AEIOUaeiou」と指定する必要があります。
Scannerの動作は、値の表現に関して言語や慣習を特定するロケールの影響を受けます。例えばNSScannerは十進数の区切り（英語では1,000倍ごとに入れるカンマのこと）に関してロケールの定義を使用します。プログラマはメソッド、localizedScannerWithString: を使ってロケール指定付きのScannerオブジェクトを生成することができます。また、 setLocale: メソッドを使い、既存のScannerオブジェクトに対して後からロケールを指定することも可能です。ロケールの指定がない場合、Scannerはデフォルトのロケール定義を使用します。

書式付きStringオブジェクト

NSStringオブジェクトで使用される書式の構文は、他のFormatterオブジェクトで使用されるものと同じです。NSStringクラスは、ANSI C の printf() 関数で定義されている変換指定記号（フォーマットキャラクタ）に加えて、オブジェクト全般に使用できる「%@」という表現をサポートしています（詳しくは後で出てくる表1を参照してください）。
オブジェクトがメッセージ、 descriptionWithLocale: に応じる場合、NSStringはこれを送って表示するテキストを検索します。それ以外の場合には、 description メッセージを送ります。ローカライズされた文字列内の変数の並び替えなどについては、ドキュメント「String Files」に詳しい記述があります。
プログラマはNSStringのクラスメソッド、stringWithFormat: を使用して、printf() 形式の書式を伴ったStringオブジェクトを生成することができます。これに関して詳しくは「Stringオブジェクトの生成と変換」で説明します。
以下のサンプルでは、さまざまな変換指定記号を使って文字列を作成しています。

```
NSString *string1 = [NSString stringWithFormat:@"A string: %@, a float: %1.2f",
                     @"string", 31415.9265];
// string1 is "A string: string, a float: 31415.93"

NSNumber *number = [NSNumber numberWithInt:1234];
NSDictionary *dictionary = [NSDictionary dictionaryWithObject:[NSDate date]
                                                       forKey:@"date"];
NSString *baseString = @"Base string.";
NSString *string2 = [baseString stringByAppendingFormat:
                     @" A number: %@, a dictionary: %@", number, dictionary];
// string2 is "Base string. A number: 1234, a dictionary: {date = 2005-10-17 09:02:01
-0700; }"
```

多くのコンパイラが、ptrintf() へのパラメータに対してタイプキャスティングを実行しているため、整数に対して「%f」という変換指定記号を指定しても正しい値が出力されます。が、このタイプキャスティングはNSStringでは行われません。変数型と変換指定記号は一致していなければなりません。

Stringオブジェクトと非ASCII文字

Ｓtringオブジェクトに非ASCII文字（ユニコードを含む）を入れたい場合は、stringWithFormat: あるいは stringWithUTF8String: を使用します。

```
NSString *s = [NSString stringWithFormat:@"Long %C dash", 0x2014];
```

また、「\xe2\x80\x94」 は「0x2014」に相当するUTF-8Stringですから、以下の方法も可能です。

```
NSString *s = [NSString stringWithUTF8String:"Long \xe2\x80\x94 dash"];
```

high-bitキャラクタ、すなわち8ビット目がオンになる文字をソースコードに含めるのは「安全」ではありません。

```
NSString *s = [NSString stringWithUTF8String:"Long – dash"];
NSString *s = @"Long – dash"; // Not allowed
```

NSLog と NSLogv

ユーティリティ関数、NSLog() と NSLogv() は、エラーメッセージを記録するためにNSStringの書式サービスを利用しています。これらの関数にパラメータを渡す場合、そのことに留意する必要があります。最もあちがちなミスは、以下のように変換指定文字を含む書式指定つきのStringを渡してしまうことです。

```
NSString *string = @"A contrived string %@";
NSLog(string);
// The application will crash here due to signal 10 (SIGBUS)
```

次の例のように変換指定文字の対象があればまだましです（結果は期待通りではないでしょうが、少なくともクラッシュは避けられます）。

```
NSString *string = @"A contrived string %@";
NSLog(@"%@", string);
// Output: A contrived string %@
```

変換指定文字一覧

以下の表はNSStringのフォーマッティング・メソッドがサポートしている変換指定文字の一覧です。

変換指定文字	説明
%%	% 文字
%d, %D, %i	符号付き32ビット整数（long）
%u, %U	符号なし32ビット整数（unsigned long）
%hi	符号付き16ビット整数（short）
%hu	符号なし16ビット整数（unsigned short）
%qi	符号付き64ビット整数（long long）
%qu	符号なし64ビット整数（unsigned long long）
%x	符号なし32ビット整数（unsigned long）を16進表現（0-9,a-f）で出力
%X	符号なし32ビット整数（unsigned long）を16進表現（0-9,A-F）で出力
%qx	符号なし64ビット整数（unsigned long long）を16進表現（0-9,a-f）で出力
%qX	符号なし64ビット整数（unsigned long long）を16進表現（0-9,A-F）で出力
%o, %O	符号なし32ビット整数（unsigned long）を8進表現で出力
%f	64ビット浮動小数点数（double）
%e	64ビット浮動小数点数（double）を指数形式（指数部記号は「e」）で出力
%E	64ビット浮動小数点数（double）を指数形式（指数部記号は「E」）で出力
%g	64ビット浮動小数点数（double）を、通常は%fで、指数部が-4より小さいか、または精度より大きい場合には%eを使って出力
%G	64ビット浮動小数点数（double）を、通常は%fで、指数部が-4より小さいか、または精度より大きい場合には%Eを使って出力

変換指定文字	説明
%c	符号なし8ビット整数（unsigned char）をNSLog()で、ASCII文字として出力、もしASCII文字でなかった場合は、オクタル（\\ddd）あるいはユニコードヘクサデシマル（\\udddd）で出力
%C	16ビットユニコードキャラクタ（unichar）をNSLog()で、ASCII文字として出力、もしASCII文字でなかった場合は、 オクタル（\\ddd）あるいはユニコードヘクサデシマル（\\udddd）で出力
%s	ヌルターミネートしている8ビットキャラクタの配列
%S	ヌルターミネートしている16ビットユニコードキャラクタの配列
%p	void型へのポインタ、0xに続く0-9, a-fのヘクサデシマルで出力
%@	Objective-Cのオブジェクト、出力文字列はオブジェクトの descriptionWithLocale: メソッド、あるいは description メソッドの戻り値。

Stringオブジェクトの生成と変換

NSString、およびそのサブクラスであるNSMutableStringでは、サポートしているさまざまなエンコーディングに基づく文字列からStringオブジェクトを生成する手段を提供しています。Stringオブジェクトは、その内容をいつでもユニコードキャラクタで保持しますが、同時にそれ以外のさまざまなエンコーディング、例を挙げれば7-bit ASCII、ISO Latin、EUCやShift-JISなど、にその内容を変換することができるのです。クラスメソッド availableStringEncodings によって、使用可能なエンコーディングを一覧することが可能です。また、C StringからStringオブジェクトを生成する際には、使用するエンコーディングを指定することができます。指定がない場合には、 defaultCStringEncoding で返されるデフォルトのエンコーディングが使われます。
プログラム中でStringオブジェクトを生成する最も簡単な方法はクラスメソッド stringWithCString: あるいはインスタンスメソッド initWithCString: を使用することです。どちらもヌルターミネートした通常のC StringをパラメータにしてStringオブジェクトを生成します。また、Objective-Cは7-bit ASCIIエンコーディングによるStringオブジェクトの定数を定義する書式として @”...” という書式をサポートしています。

```
NSString *temp = @"/tmp/scratch";
```

上のようなオブジェクトは、コンパイル時に生成されてプログラムの実行中メモリ上に常駐します。コンパイラはこうしたオブジェクトをモジュールベースでユニークに生成し、けして解放されることはありません（もちろんプログラム中でこれらに対し retain、あるいは release をコールすることはできます）。

StringオブジェクトからC Stringを取り出す場合、UTF8String を使うことをお勧めします。このメソッドは、UTF8エンコーディングによる文字列定数を変えします。
また、lossyCString メソッドを使用して、StringオブジェクトからC Stringを生成することも可能です。ただし、このメソッドはデフォルトのCエンコーディングを使用するので変換にあたってなんらかの情報が失われる可能性があります。
どちらの場合も、戻り値のStringオブジェクトはガベージコレクションの対象となるテンポラリなオブジェクトであることに留意してください。これらを恒久的なオブジェクトとして利用したい場合には、これらの戻り値のコピーを作らなければなりません。
なお、cString、cStringLength、getCString、getCString:maxLength: そして getCString:maxLength:range:remainingRange: の各メソッドは、近いうちに改訂される予定ですのでできるだけ使用は避けるようにしてください。

文字群からユニコードエンコーディングのStringオブジェクトを生成する、逆にStringオブジェクトから所定のエンコーディングでデータをとり出すのに使用するメソッドには上の他、以下のものがあります。initWithData:encoding: と、dataUsingEncoding: はNSDataからStringオブジェクトへ、またStringオブジェクトからNSDataへの変換を行います。また、NSStringのクラスメソッド stringWithContentsOfFile: を使用すればファイルから直接内容を読み込んでStringオブジェクトを生成することができ、インスタンスメソッド、writeToFile:atomically: を使えば、オブジェクトの内容をファイルに書き込むことが可能です。

最後に、既存のStringオブジェクトから新たなStringオブジェクトを生成するメソッドには2つのタイプがあります。localizedStringWithFormat: とその仲間は、変換指定文字を含む書式文字列と、その変換指定文字に対応する変数を使ってStringを生成します。stringByAppendingString: および、stringByAppendingFormat: は、既存のStringオブジェクトに新たなStringを追加することで新たなStringを生成します。

以下の表はStringオブジェクトの生成／抽出の関係を要約したものです。

ソース	生成メソッド	抽出メソッド
デフォルト C String エンコーディング	`stringWithCString:`	推奨されていません
UTF8エンコーディング	`stringWithUTF8String:`	`UTF8String`
ソースコード中で...	`@"..."`（コンパイラによる生成）	
ユニコードエンコーディング	`stringWithCharacters:length:`	`getCharacters:length:`
エンコーディング指定	`initWithData:encoding:`	`detaUsingEncoding:`
ファイルの内容	`stringWithContentsOfFile:`	`writeToFile:atomically:`
書式文字列による指定	`localizedStringWithFormat:`	~~`initWithFormat:locale`~~`:`
既存のStringオブジェクト	`stringByAppendingString:`	~~`stringByAppendingFormat:`~~

訳注：表中線で消してある項目は原文では消されていないが、明らかに「抽出メソッド」ではない。

Stringオブジェクトの検索と比較

Stringのクラスには、Stringオブジェクトから文字または文字列を検索するメソッド、およびStringオブジェクト同士の比較を行うメソッドが定義されています。これらのメソッドは、ユニコード規格に基づいて2つのキャラクタシーケンスの同一性を判定します。この比較メソッドは 汎用的かつ正確ですが、プログラムやデータの性格に合わせて、より効率的なメソッドで代替することも可能です。
検索メソッド、比較メソッドにはそれぞれ3種類のバリエーションがあります。それぞれの最もシンプルなバージョンはString全体を検索、あるいは比較するもの、その他は検索、比較の対象となる範囲を限定するものです。検索、比較に際しては以下のオプションが指定できます（すべてのメソッドで以下のすべてのオプションが指定可能なわけではありません）。

オプション	効果
NSCaseInsensitiveSearch	アルファベットの大文字／小文字を区別しない。
NSLiteralSearch	バイト・フォー・バイト比較。 キャラクタシーケンスにより「同じ」と見なされるような要素を排除してバイトごとの比較を行う。
NSBackwardsSearch	指定された範囲の最後から先頭に向けて検索を行う。
NSAnchoredSearch	指定された範囲の端の文字（先頭と末尾）のみを検索し、このどちらかに合致しなければ、もしその範囲に合致する要素があっても無視される。
NSNumericSearch	数字を数値として扱って比較する。例えば、FileName9.txt < FileName20.txt < FileName100.txt といった具合。

現在のところ、検索、比較処理はデフォルトでNSLiteralSearchオプションが指定されているものと見なします。ユニコードエンコーディングがより一般的になり、よりフレキシブルな比較方法が必要とされるにしたがって、このデフォルトの振る舞いは変更される予定です。

なお、検索にあたっては、検索文字列が指定された範囲に完全に含まれる場合のみ「発見」されます。また、もし範囲を指定して検索を行う場合に NSLiteralSearchオプションを指定しないのであれば、指定範囲の先頭、および末尾がキャラクタシーケンスと合致していなければなりません（詳しくは、`rangeOfComposedCharacterSequenceAtIndex:` の説明を参照してください）。

基本的な検索、比較メソッドの一覧

検索メソッド	比較メソッド
rangeOfString:	compare:
rangeOfString:options:	compare:options:
rangeOfString:options:range:	compare:options:range:
rangeOfCharacterFromSet:	
rangeOfCharacterFromSet:options:	
rangeOfCharacterFromSet:options:range:	

rangeOfString: メソッドは渡されたStrngがレシーバーに含まれるかどうか探します。
rangeOfCharacterFromSet: メソッドは渡されたCharacterSetのメンバーがレシーバーに含まれるがどうか探します。compare: メソッドは渡されたStringのレシーバーに対する辞書的な順番（アルファベット順）での大小を返します。これら３つの基本形に、オプションの指定、オプションと範囲の指定を組み合わせたのがその他のメソッドです。

なお、NSScannerクラスを使えばStringオブジェクト中に存在する数値を抽出することができます。
NSString、NSScannerのコンビは、検索、比較にあたってNSCharacterSetクラスを利用しています。

Stringの結合と抽出

Stringを結合したり抽出したりするにはさまざまな方法があります。最もシンプルな結合方法は、一方をもう一方に追加するやり方です。`stringByAppendingString:` メソッドは渡されたStringをレシーバーの末尾に結合したStringオブジェクトを返します。`stringByAppendingFormat:` あるいは `initWithFormat:` を使用すれば、いくつかのStringオブジェクトを一度に結合することも可能です。

また、Stringオブジェクトの一部を抽出することも可能です。レシーバーの最初からあるインデックスまでを取り出すには `substringToIndex:` 、あるインデックスから最後までなら `substringFromIndex:` 途中のある範囲を取り出したい場合には `substringWithRange:` を使用します。また、`componentsSeparatedByString:` メソッドを使えば、レシーバーを一度にいくつかのStringに分割することも可能です。

Stringオブジェクトの描画

Application Kitはフォーカスしているビューに対して描画するために、、NSStringクラスに `drawAtPoint:withAttributes:`、`drawInRect:withAttributes:`、`sizeWithAttributes:` という3種類のメソッドを用意しています（これはObjective-Cにのみ当てはまります。JavaではNSStringクラスではなく java.lang.string を使うことになるからです）。また、同様なメソッドはNSAttributedStringクラスに対しても用意されています。
最初の2つのメソッドは文字列全体を指定された属性で描画するもの、残りの１つはレシーバーを描画するために必要とされる描画領域のサイズを計算します。
数種類のフォントなど、複数の属性を指定したStringを描画する必要がある場合には、NSStringをパラメータにして新しいNSAttributedStringオブジェクトを生成して描画を行います。

`drawAtPoint:withAttributes:`、`drawInRect:withAttributes:` という２つのメソッドは、少量のテキストか，あるいはめったに再描画が発生しないようなテキストに対して使われるよう設計されているので，繰り返し文字列の描画を行う必要がある場合には NSLayoutManager を使う方が効率的です。

Parh-Stringの操作

NSStringクラスはファイルシステムで使用される「Path-String」を操作するために豊富な機能を提供します。それらは Path-Stringを保持するStringオブジェクトから、ディレクトリやファイル名、拡張子を抽出できますし、ユーザーのホーム・ディレクトリを表すチルダ表現（~username みたいなもの）を展開したり、逆に生成したりすることもできます。Pathに含まれるシンボリック・リンクを解決することも可能ですし、カレントディレクトリや親ディレクトリを表す記号も正しく扱えます。

NSStringクラスでは、基本的に「/」（スラッシュ）をPathセパレーター、「.」（ピリオド）を拡張子セパレーターと解釈します。ファイルやディレクトリを指定するParh-Stringをパラメータとして受け取る各種メソッドは、これらの一般的表現を内部で自動的にシステムに適合する形に変換します。ルートディレクトリがあるシステム（Mac OS Xはこれに当たります。Windowsは違いますね）では、絶対PathはPathセパレーター「/...」あるいはチルダ表現「~username/...」から始まります。デバイスの指定が必要な場所では、システムの仕様にしたがってそれを行えますし、Stringオブジェクトにデフォルト・デバイスを追加させることもできます。

CharacterSetの生成

NSCharacterSetクラスには、例えば「letters（アルファベットの大文字と小文字）」、「10進数字」、「空白文字」など、使用頻度の高いCharacterSetを返すクラスメソッドが用意されています。これらの「標準的な」CharacterSet（その内容については本章後半に詳しく述べます）は、たとえNSMutableCharacterSetクラスにメッセージを送って生成したものであっても変更不能です。が、これらのオブジェクトに `mutableCopy` メッセージを送って変更可能なコピーを作成することは可能であり、それをカスタマイズしてオリジナルのCharacterSetを作成することができます。例えば以下のコードは文字と数字に一般的な括弧を含むCharacterSetを生成しています。

```
NSMutableCharacterSet *workingSet;
NSCharacterSet *finalCharSet;

workingSet = [[NSCharacterSet alphanumericCharacterSet] mutableCopy];
[workingSet addCharactersInString:@";:,."];
finalCharSet = [workingSet copy];
[workingSet release];
```

ユニコードを使ってカスタムのCharacterSetを定義する場合は以下のコード（これは改ページ文字と改行文字のCharacterSetを作成しています）を参考にしてください。

```
UniChar chars[] = {0x000C, 0x2028};
NSString *string = [[NSString alloc] initWithCharacters:chars length:sizeof(chars)
/ sizeof(UniChar)];
NSCharacterSet *chset = [NSCharacterSet
characterSetWithCharactersInString:string];
[string release];
```

CharacterSetの項で説明したパフォーマンス的な理由から、カスタマイズを終えた変更可能なCharacterSetは必ず変更不能なオブジェクトに変換して使用することを推奨します。ただし、作成したそのCharacterSetの内容を頻繁に変更する必要があるのであれば別ですが。
また、この手法でカスタマイズしたCharacterSetを多用するようなアプリケーションでは、当然ながら一度作成したCharacterSetをリソースなどに保存して必要に応じてロードするようにする方が効率的です。以下のようにすれば、CharacterSetをNSDataの形式でファイルに保存することができます。

```
NSString *filename; /* Assume this exists. */
NSString *absolutePath;
NSData *charSetRep;
BOOL result;

absolutePath = [filename stringByStandardizingPath];
charSetRep = [finalCharSet bitmapRepresentation];
result = [charSetRep writeToFile:absolutePath atomically:YES];
```

CharacterSetを保存したファイルの拡張子は慣習として「.bitmap」とされています。このファイルを他のプログラマが利用できるように、できればこの慣習に従ってください。「.bitmap」ファイルは、`characterSetWithContentsOfFile:` メソッドで簡単にロードすることができます。

標準的CharacterSetとユニコード定義

`letterCharacterSet` メソッドなどで返される標準的CharacterSetは、例えばUppercase Letter、Combining Mark などの、ユニコード規格に定められたカテゴリに基づいて定義されており、多くの場合、１つあるいは複数のカテゴリの組み合わせたからできています。例えば `lowercaseLetterCharacterSet` によって返されるCharacterSetは、Lowercase Letters というカテゴリに含まれる文字のすべてを含んでいるし、 `letterCharacterSet` の戻り値は Letter カテゴリに含まれるすべての文字の集合です。

ユニコード規格の変更によってカテゴリの内容も変更される可能性があることに留意してください。ユニコード規格のカテゴリ定義は、http://www.unicode.org/ からダウンロードすることができます。

ドキュメント更新履歴

オリジナル・ドキュメントは「Strings Programming Guide for Cocoa」2006/01/10版。

要約は2006/01/25。本文中にも記したように要約にあたって一部の内容を訂正。