Predicate Programming Guide

Data Management: Data Types & Collections

2009-02-03

Apple Inc.
© 2005, 2009 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Cocoa, Logic, Mac, Mac OS, Objective-C, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Finder, iPhone, and Spotlight are trademarks of Apple Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction to Predicates Programming Guide 5

Who Should Read This Document? 5
Organization of This Document 5

Predicate Basics 7

Predicates 7
Expressions 7
Constraints and Limitations 8

Creating Predicates 9

Creating a Predicate Using a Format String 9
- String Constants, Variables, and Wildcards 9
- Boolean Values 10
- Dynamic Property Names 10

Creating Predicates Directly in Code 11
Creating Predicates Using Predicate Templates 12
Format String Summary 12

Using Predicates 15

Evaluating Predicates 15
Using Predicates with Arrays 15
Using Predicates with Relationships 16
Null Values 16
- Testing for Null 17

Using Predicates with Cocoa Bindings 17
Using Predicates with Core Data 18
- Fetch Requests 18
- Object Controllers 18

Regular Expressions 18
Performance 19
- Joins 19
- Structuring Your Data 20

Comparison of NSPredicate and Spotlight Query Strings 21

Spotlight and NSPredicate 21
Creating a Predicate Format String From a Spotlight Search in Finder 22

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Predicate Format String Syntax 23

Parser Basics 23
Basic Comparisons 23
Boolean Value Predicates 24
Basic Compound Predicates 25
String Comparisons 25
Aggregate Operations 25
Identifiers 26
Literals 26
Reserved Words 27

BNF Definition of Cocoa Predicates 29

NSPredicate 29
NSCompoundPredicate 29
NSComparisonPredicate 29
Operations 29
Aggregate Qualifier 30
Expression 30
Value Expression 30
Literal Value 30
String Value 30
Predicate Argument 30
Format Argument 31
Predicate Variable 31
Keypath Expression 31
Literal Aggregate 31
Index Expression 31
Aggregate Expression 31
Assignment Expression 31
Binary Expression 32
Binary Operator 32
Function Expression 32
Function Name 32
Array Expression 32
Dictionary Expression 32
Integer Expression 32
Numeric Value 33
Identifier 33

Document Revision History 35

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Introduction to Predicates Programming Guide

Predicates provide a general means of specifying queries in Cocoa. The predicate system is capable of handling a large number of domains, not just—for example—Core Data or Spotlight. This document describes predicates in general, their use, their syntax, and their limitations.

Who Should Read This Document?

You should read this document if your application needs to filter objects or to set up queries retrieving objects using Core Data.

Organization of This Document

The following articles explain the basics of predicates in Cocoa, explain how to create and use predicate objects, and define the predicate syntax:

- "Predicate Basics" (page 7) describes the fundamental aspects of the technology.
- "Creating Predicates" (page 9) describes how to correctly instantiate predicates programmatically and how to retrieve them from a managed object model.
- "Using Predicates" (page 15) explains how to use predicates and discusses some issues related to performance.
- "Comparison of NSPredicate and Spotlight Query Strings" (page 21) compares NSPredicate and Spotlight queries.
- "Predicate Format String Syntax" (page 23) describes the syntax of the predicate format string.
- "BNF Definition of Cocoa Predicates" (page 29) provides a definition of Cocoa predicates in Backus-Naur Form notation.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Predicate Basics

Cocoa predicates provide a means of encoding queries in a manner independent of the backing store used to hold the data being searched. You use predicates to represent logical conditions used for constraining the set of objects retrieved by Spotlight and Core Data, and for in-memory filtering of objects.

Predicates

There are two types of predicate, comparison and compound. Comparison predicates are used to compare two expressions. Comparison predicates have a left expression, a right expression, and an operator, and return the result of invoking the operator with the results of evaluating the expressions. Compound predicates join two or more predicates using a logical operator (AND or OR), or negate another predicate (a NOT operation).

Comparison predicates and compound predicates are represented in Cocoa by instances of `NSComparisonPredicate` and `NSCompoundPredicate` respectively. Both classes inherit from `NSPredicate`.

Cocoa supports a wide range of types of predicate, including the following:

- Simple comparisons, such as `grade == 7` or `firstName like 'Mark'`
- Case or diacritic insensitive lookups, such as `name contains[cd] 'citroen'`
- Logical operations, such as `(firstName beginswith 'M') AND (lastName like 'Adderley')`

You can also create predicates for relationships—such as `group.name matches 'work.*'`, `ALL children.age > 12`, and `ANY children.age > 12`—and for operations such as `@sum.items.price < 1000`.

You can use predicates with any class of object, but note that a class must be key-value coding compliant for the keys you want to use in a predicate (see *Key-Value Coding Programming Guide*).

Expressions

Predicate expressions in Cocoa are represented by instances of `NSExpression`. The simplest expression simply represents a constant value, as shown in the following example.

```
NSExpression *expression = [NSExpression
    expressionForConstantValue:[NSDecimalNumber numberWithInt:10]];
```

You frequently use expressions that retrieve the value for a key path of the object currently being evaluated in the predicate.

```
NSExpression *expression = [NSExpression
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

```
    expressionForKeyPath:@"lastName"];
```

You can also create an expression to represent the object currently being evaluated in the predicate, to serve as a placeholder for a variable to be extracted from a variable bindings dictionary, or to return the result of performing an operation on an array, as illustrated by the following example.

```
NSExpression *expression = [NSExpression
    expressionForFunction:@"max" arguments:anArray];
```

The expression system is currently not extensible.

Constraints and Limitations

If you use predicates with Core Data or Spotlight, you should take care that they will work with the data store. There is no specific implementation language for predicate queries—a predicate query may be translated into SQL, XML, or another format, depending on the requirements of the backing store (if indeed there is one).

The predicate system is intended to support a useful range of operators, so provides neither the set union nor the set intersection of all operators supported by all backing stores. Therefore, not all possible predicate queries are supported by all backing stores, and not all operations supported by all backing stores can be expressed with `NSPredicate` and `NSExpression`. The back end may downgrade the predicate (for example it may make a case-sensitive comparison case-insensitive) or raise an exception if you try to use an unsupported operator. For example:

- The `matches` operator uses `regex`, so is not supported by Core Data's SQL store— although it does work with in-memory filtering.
- The Core Data SQL store supports only one to-many operation per query; therefore in any predicate sent to the SQL store, there may be only one operator (and one instance of that operator) from `ALL`, `ANY`, and `IN`.
- You cannot necessarily translate "arbitrary" SQL queries into predicates.
- The `ANYKEY` operator can only be used with Spotlight.
- Spotlight does not support relationships.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Creating Predicates

There are three ways to create a predicate in Cocoa: using a format string, directly in code, and from a predicate template. Note, however, that not all possible predicate queries are supported by all backing stores.

Creating a Predicate Using a Format String

Creating predicates using the format string is quick and easy to code. It is typically best used for predefined query terms, although variable substitution allows for considerable flexibility. The disadvantage of this technique is that you must take care to avoid introducing errors into the string—you will not discover mistakes until runtime.

`NSPredicate` provides a number of class methods of the form `predicateWithFormat...` to create a predicate directly from a string, optionally using variable substitution. At runtime, variable substitution—if any—is performed, and the resulting string is parsed to create corresponding predicate and expression objects. Note that the predicate format string syntax is different from the regular expression syntax. The regular expression syntax is defined by the ICU—see http://www.icu-project.org/userguide/regexp.html.

The predicate string parser is whitespace insensitive, case insensitive with respect to keywords, and supports nested parenthetical expressions. It also supports `printf`-style format arguments (like `%x` and `%@`)—see Formatting String Objects. Variables are denoted with a `$` (for example `$VARIABLE_NAME`)—see "Creating Predicates Using Predicate Templates" (page 12) for more details. Note that the parser does not perform any semantic type checking. It will make a best-guess effort to create suitable expressions, but particularly in the case of substitution variables, it is possible that a runtime error will be generated. For a complete description of the string syntax and a list of all the operators available, see "Predicate Format String Syntax" (page 23). The following example creates a compound predicate with two comparison predicates.

```
NSPredicate *predicate = [NSPredicate
    predicateWithFormat:@"(lastName like[cd] %@) AND (birthday > %@)",
            lastNameSearchString, birthdaySearchDate];
```

In the example, `like[cd]` means "case- and diacritic-insensitive like."

String Constants, Variables, and Wildcards

String constants must be quoted within the expression—single and double quotes are both acceptable, but must be paired appropriately (that is, a double quote (`"`) does not match a single quote (`'`)). If you use variable substitution using `%@` (such as `firstName like %@`), the quotation marks are added for you automatically. If you use string constants within your format string, you must quote them yourself, as in the following example.

```
NSPredicate *predicate = [NSPredicate
    predicateWithFormat:@"lastName like[c] \"S*\""];
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

You must take automatic quotation into account when using wildcards—you must add wildcards to a variable prior to substitution, as shown in the following example.

```
NSString *prefix = @"prefix";
NSString *suffix = @"suffix";
NSPredicate *predicate = [NSPredicate
    predicateWithFormat:@"SELF like[c] %@",
    [[prefix stringByAppendingString:@"*"] stringByAppendingString:suffix]];
BOOL ok = [predicate evaluateWithObject:@"prefixxxxxxsuffix"];
```

In this example, variable substitution produces the predicate string `SELF LIKE[c] "prefix*suffix"`, and the value of `ok` is `YES`. The following fragment, by contrast, yields the predicate string `SELF LIKE[c] "prefix" * "suffix"`, and the predicate evaluation yields a runtime error:

```
predicate = [NSPredicate
    predicateWithFormat:@"SELF like[c] %@*%@", prefix, suffix];
ok = [predicate evaluateWithObject:@"prefixxxxxxsuffix"];
```

Finally, the following fragment results in a runtime parse error (`Unable to parse the format string "SELF like[c] %@*"`).

```
predicate = [NSPredicate
    predicateWithFormat:@"SELF like[c] %@*", prefix];
```

You should also note the difference between variable substitution in the format string and variable expressions. The following code fragment creates a predicate with a right-hand side that is a variable expression.

```
predicate = [NSPredicate
    predicateWithFormat:@"lastName like[c] $LAST_NAME"];
```

For more about variable expressions, see "Creating Predicates Using Predicate Templates" (page 12).

Boolean Values

You specify and test for equality of Boolean values as illustrated in the following examples:

```
NSPredicate *newPredicate =
    [NSPredicate predicateWithFormat:@"anAttribute == %@", [NSNumber
numberWithBool:aBool]];
NSPredicate *testForTrue =
    [NSPredicate predicateWithFormat:@"anAttribute == YES"];
```

Dynamic Property Names

Because string variables are surrounded by quotation marks when they are substituted into a format string using `%@`, you *cannot* use `%@` to specify a dynamic property name—as illustrated in the following example.

```
NSString *attributeName = @"firstName";
NSString *attributeValue = @"Adam";
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"%@ like %@",
        attributeName, attributeValue];
```

The predicate format string in this case evaluates to `"firstName" like "Adam"`.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

If you want to specify a dynamic property name, you use `%K` in the format string, as shown in the following fragment.

```
predicate = [NSPredicate predicateWithFormat:@"%K like %@",
        attributeName, attributeValue];
```

The predicate format string in this case evaluates to `firstName like "Adam"` (note that there are no quotation marks around `firstName`).

Creating Predicates Directly in Code

You can create predicate and expression instances directly in code. `NSComparisonPredicate` and `NSCompoundPredicate` provide convenience methods that allow you to easily create compound and comparison predicates respectively. `NSComparisonPredicate` provides a number of operators ranging from simple equality tests to custom functions.

The following example shows how to create a predicate to represent `(revenue >= 1000000) and (revenue < 100000000)`. Note that the same left-hand side expression is used for both comparison predicates.

```
NSExpression *lhs = [NSExpression expressionForKeyPath:@"revenue"];

NSExpression *greaterThanRhs = [NSExpression expressionForConstantValue:[NSNumber
 numberWithInt:1000000]];
NSPredicate *greaterThanPredicate = [NSComparisonPredicate
    predicateWithLeftExpression:lhs
    rightExpression:greaterThanRhs
    modifier:NSDirectPredicateModifier
    type:NSGreaterThanOrEqualToPredicateOperatorType
    options:0];

NSExpression *lessThanRhs = [NSExpression expressionForConstantValue:[NSNumber
 numberWithInt:100000000]];
NSPredicate *lessThanPredicate = [NSComparisonPredicate
    predicateWithLeftExpression:lhs
    rightExpression:lessThanRhs
    modifier:NSDirectPredicateModifier
    type:NSLessThanPredicateOperatorType
    options:0];

NSCompoundPredicate *predicate = [NSCompoundPredicate
andPredicateWithSubpredicates:
    [NSArray arrayWithObjects:greaterThanPredicate, lessThanPredicate, nil]];
```

The disadvantage of this technique should be immediately apparent—you may have to write a lot of code. The advantages are that it is less prone to spelling and other typographical errors that may only be discovered at runtime and it may be faster than depending on string parsing.

This technique is most likely to be useful when the creation of the predicate is itself dynamic, such as in a predicate builder.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Creating Predicates Using Predicate Templates

Predicate templates offer a good compromise between the easy-to-use but potentially error-prone format string-based technique and the code-intensive pure coding approach. A predicate template is simply a predicate that includes a variable expression. (If you are using the Core Data framework, you can use the Xcode design tools to add predicate templates for fetch requests to your model—see Managed Object Models.) The following example uses a format string to create a predicate with a right-hand side that is a variable expression.

```
NSPredicate *predicateTemplate = [NSPredicate
    predicateWithFormat:@"lastName like[c] $LAST_NAME"];
```

This is equivalent to creating the variable expression directly as shown in the following example.

```
NSExpression *lhs = [NSExpression expressionForKeyPath:@"lastName"];

NSExpression *rhs = [NSExpression expressionForVariable:@"LAST_NAME"];

NSPredicate *predicateTemplate = [NSComparisonPredicate
    predicateWithLeftExpression:lhs
    rightExpression:rhs
    modifier:NSDirectPredicateModifier
    type:NSLikePredicateOperatorType
    options:NSCaseInsensitivePredicateOption];
```

To create a valid predicate to evaluate against an object, you use the `NSPredicate` method `predicateWithSubstitutionVariables:` to pass in a dictionary that contains the variables to be substituted. (Note that the dictionary must contain key-value pairs for all the variables specified in the predicate.)

```
NSPredicate *predicate = [predicateTemplate predicateWithSubstitutionVariables:
    [NSDictionary dictionaryWithObject:@"Turner" forKey:@"LAST_NAME"]];
```

The new predicate returned by this example is `lastName LIKE[c] "Turner"`.

Because the substitution dictionary must contain key-value pairs for all the variables specified in the predicate, if you want to match a null value, you must provide a null value in the dictionary, as illustrated in the following example.

```
NSPredicate *predicate = [NSPredicate
    predicateWithFormat:@"date = $DATE"];
predicate = [predicate predicateWithSubstitutionVariables:
    [NSDictionary dictionaryWithObject:[NSNull null] forKey:@"DATE"]];
```

The predicate formed by this example is `date == <null>`.

Format String Summary

It is important to distinguish between the different types of value in a format string. Note also that single or double quoting variables (or substitution variable strings) will cause `%@`, `%K`, or `$variable` to be interpreted as a literal in the format string and so will prevent any substitution.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

`@"attributeName == %@"`

This predicate checks whether the value of the key `attributeName` is the same as the value of the object `%@` that is supplied at runtime as an argument to `predicateWithFormat:`. Note that `%@` can be a placeholder for any object whose description is valid in the predicate, such as an instance of `NSDate`, `NSNumber`, `NSDecimalNumber`, or `NSString`.

`@"%K == %@"`

This predicate checks whether the value of the key `%K` is the same as the value of the object `%@`. The variables are supplied at runtime as arguments to `predicateWithFormat:`.

`@"name IN $NAME_LIST"`

This is a template for a predicate that checks whether the value of the key `name` is in the variable `$NAME_LIST` (no quotes) that is supplied at runtime using `predicateWithSubstitutionVariables:`.

`@"'name' IN $NAME_LIST"`

This is a template for a predicate that checks whether the constant value `'name'` (note the quotes around the string) is in the variable `$NAME_LIST` that is supplied at runtime using `predicateWithSubstitutionVariables:`.

`@"$name IN $NAME_LIST"`

This is a template for a predicate that expects values to be substituted (using `predicateWithSubstitutionVariables:`) for both `$NAME` and `$NAME_LIST`.

`@"%K == '%@'"`

This predicate checks whether the value of the key `%K` is equal to the string literal "`%@`" (note the single quotes around `%@`). The key name `%K` is supplied at runtime as an argument to `predicateWithFormat:`.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Using Predicates

This document describes in general how you use predicates, and how the use of predicates may influence the structure of your application data.

Evaluating Predicates

To evaluate a predicate, you use the `NSPredicate` method `evaluateWithObject:` and pass in the object against which the predicate will be evaluated. The method returns a Boolean value—in the following example, the result is `YES`.

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"SELF IN %@",
    [NSArray arrayWithObjects:@"Stig", @"Shaffiq", @"Chris", nil]];
BOOL result = [predicate evaluateWithObject:@"Shaffiq"];
```

You can use predicates with any class of object, but note that a class must support key-value coding for the keys you want to use in a predicate.

Using Predicates with Arrays

`NSArray` and `NSMutableArray` provide methods to filter array contents. `NSArray` provides `filteredArrayUsingPredicate:` which returns a new array containing objects in the receiver that match the specified predicate. `NSMutableArray` provides `filterUsingPredicate:` which evaluates the receiver's content against the specified predicate and leaves only objects that match.

```
NSMutableArray *array =
    [NSMutableArray arrayWithObjects:@"Bill", @"Ben", @"Chris", @"Melissa",
nil];

NSPredicate *bPredicate = [NSPredicate predicateWithFormat:@"SELF beginswith[c]
 'b'"];
NSArray *beginWithB = [array filteredArrayUsingPredicate:bPredicate];
// beginWithB contains { @"Bill", @"Ben" }.

NSPredicate *sPredicate = [NSPredicate predicateWithFormat:@"SELF contains[c]
's'"];
[array filterUsingPredicate:sPredicate];
// array now contains { @"Chris", @"Melissa" }
```

If you use the Core Data framework, the array methods provide an efficient means of filtering an existing array of objects without—as a fetch does—requiring a round trip to a persistent data store.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Using Predicates with Relationships

Recall that you can follow relationships in a predicate using a key path. The following example illustrates the creation of a predicate to find employees that belong to a department with a given name (but see also "Performance" (page 19)).

```
NSString *departmentName = ... ;
NSPredicate *predicate = [NSPredicate predicateWithFormat:
        @"department.name like %@", departmentName];
```

If you use a to-many relationship, the construction of a predicate is slightly different. If you want to fetch Departments in which at least one of the employees has the first name "Matthew," for instance, you use an `ANY` operator as shown in the following example:

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:
    @"ANY employees.firstName like 'Matthew'"];
```

If you want to find Departments in which *all* the employees are paid more than a certain amount, you use an `ALL` operator as shown in the following example:

```
float salary = ... ;
NSPredicate *predicate = [NSPredicate predicateWithFormat:
    @"ALL employees.salary > %f", salary];
```

Null Values

A comparison predicate does not match any value with null except null (`nil`) or the `NSNull` null value (that is, (`$value == nil`) returns `YES` if `$value` is `nil`). Consider the following example.

```
NSString *firstName = @"Ben";

NSMutableArray *array = [NSMutableArray array];
[array addObject:
        [NSDictionary dictionaryWithObject:@"Turner" forKey:@"lastName"]];
[array addObject:
        [NSDictionary dictionaryWithObjectsAndKeys:@"Ben", @"firstName",
                @"Ballard", @"lastName",
               [NSDate dateWithString:@"1972-03-24 10:45:32 +0600"], @"birthday", nil]];

NSPredicate *predicate =
    [NSPredicate predicateWithFormat:@"firstName like %@", firstName];
NSArray *filteredArray = [array filteredArrayUsingPredicate:predicate];

NSLog(@"filteredArray: %@", filteredArray);
// Output:
// filteredArray ({birthday = 1972-03-24 10:45:32 +0600; \\
                       firstName = Ben; lastName = Ballard; })
```

The predicate does match the dictionary that contains a value `Ben` for the key `firstName`, but does not match the dictionary with no value for the key `firstName`. The following code fragment illustrates the same point using a date and a greater-than comparator.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

```
NSDate *referenceDate = [NSDate dateWithTimeIntervalSince1970:0];

predicate = [NSPredicate predicateWithFormat:@"birthday > %@", referenceDate];
filteredArray = [array filteredArrayUsingPredicate:predicate];

NSLog(@"filteredArray: %@", filteredArray);
// Output:
// filteredArray: ({birthday = 1972-03-24 10:45:32 +0600; \\
                       firstName = Ben; lastName = Ballard; })
```

Testing for Null

If you want to match null values, you must include a specific test in addition to other comparisons, as illustrated in the following fragment.

```
predicate = [NSPredicate predicateWithFormat:@"(firstName like %@) || (firstName = nil)",
 firstName];
filteredArray = [array filteredArrayUsingPredicate:predicate];
NSLog(@"filteredArray: %@", filteredArray);

// Output:
// filteredArray: ( { lastName = Turner; }, { birthday = 1972-03-23 20:45:32 -0800;
firstName = Ben; lastName = Ballard; }
```

By implication, a test for null that matches a null value returns true. In the following code fragment, `ok` is set to `YES` for both predicate evaluations.

```
BOOL ok;
predicate = [NSPredicate predicateWithFormat:@"firstName = nil"];
ok = [predicate evaluateWithObject:[NSDictionary dictionary]];

ok = [predicate evaluateWithObject:
    [NSDictionary dictionaryWithObject:[NSNull null] forKey:@"firstName"]];
```

Using Predicates with Cocoa Bindings

You can set a predicate for an array controller to filter the content array. You can set the predicate in code (using `setFilterPredicate:`). You can also bind the array controller's `filterPredicate` binding to a method that returns an `NSPredicate` object. The object that implements the method may be the File's Owner or another controller object. If you change the predicate, remember that you must do so in a key-value observing compliant way (see *Key-Value Observing Programming Guide*) so that the array controller updates itself accordingly.

You can also bind the `predicate` binding of an `NSSearchField` object to the `filterPredicate` of an array controller. A search field's `predicate` binding is a multivalue binding, described in Binding Types.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Using Predicates with Core Data

If you are using the Core Data framework, you can use predicates in the same way as you would if you were not using Core Data (for example, to filter arrays or with an array controller). In addition, however, you can also use predicates as constraints on a fetch request and you can store fetch request templates in the managed object model (see Managed Object Models).

Note the you cannot necessarily translate "arbitrary" SQL queries into predicates or fetch requests. There is no way, for example, to convert a SQL statement such as `SELECT name, sum(value) FROM ... GROUP BY name` into a fetch request. You must fetch the objects of interest, then either perform a calculation directly using the results, or use an array operator.

Fetch Requests

You create a predicate to match properties of the target entity (note that you can follow relationships using key paths) and associate the predicate with a fetch request. When the request is executed, an array is returned that contains the objects (if any) that match the criteria specified by the predicate. The following example illustrates the use of a predicate to find employees that earn more than a specified amount.

```
NSFetchRequest *request = [[[NSFetchRequest alloc] init] autorelease];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Employee"
        inManagedObjectContext:managedObjectContext];
[request setEntity:entity];
// assume salaryLimit defined as an NSNumber variable
NSPredicate *predicate = [NSPredicate predicateWithFormat:
        @"salary > %@", salaryLimit];
[request setPredicate:predicate];
NSError *error = nil;
NSArray *array = [managedObjectContext executeFetchRequest:request error:&error];
```

Object Controllers

If you are using Cocoa bindings, you can specify a fetch predicate for an object controller (such as an instance of `NSObjectController` or `NSArrayController`). You can type a predicate directly into the predicate editor text field in the inspector panel of Interface Builder or you can set it programmatically using `setFetchPredicate:`. The predicate is used to constrain the results returned when the controller executes a fetch. If you are using an `NSObjectController` object, you specify a fetch that uniquely identifies the object you want to be the controller's content—for example, if the controller's entity is Department, the predicate might be `name like "Engineering"`.

Regular Expressions

The `MATCHES` operator uses ICU's Regular Expressions package, as illustrated in the following example:

```
NSArray *array = [NSArray arrayWithObjects:
    @"TATACCATGGGCCATCATCATCATCATCATCATCATCATCACAG",
    @"CGGGATCCCTATCAAGGCACCTCTTCG", @"CATGCCATGGATACCAACGAGTCCGAAC",
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

```
    @"CAT", @"CATCATCATGTCT", @"DOG", nil];

// find strings that contain a repetition of at least 3 'CAT' sequences,
// but not followed by a further 'CA'
NSPredicate *catPredicate =
    [NSPredicate predicateWithFormat:@"SELF MATCHES '.*(CAT){3,}(?!CA).*'"];

NSArray *filteredArray = [array filteredArrayUsingPredicate:catPredicate];
// filteredArray contains just 'CATCATCATGTCT'
```

Note that, according to the ICU specification, regular expression metacharacters are not valid inside a pattern set. For example, the regular expression `\d{9}[\dxX]` does *not* match valid ISBN numbers (any ten digit number, or a string with nine digits and the letter 'X') since the pattern set (`[\dxX]`) contains a metacharacter (`\d`). Instead you could write an OR expression, as shown in the following code sample:

```
NSArray *isbnTestArray =
    [NSArray arrayWithObjects:@"123456789X", @"987654321x",
        @"1234567890", @"12345X", @"1234567890X", nil];
NSPredicate *isbnPredicate =
    [NSPredicate predicateWithFormat:@"SELF MATCHES '\\\\d{10}|\\\\d{9}[Xx]'"];

NSArray *isbnArray = [isbnTestArray filteredArrayUsingPredicate:isbnPredicate];
// isbnArray contains (123456789X, 987654321x, 1234567890)
```

Performance

You should structure compound predicates to minimize the amount of work done. Regular expression matching in particular is an expensive operation. In a compound predicate, you should therefore perform simple tests before a regular expression; thus instead of using a predicate shown in the following example:

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:
    @"( title matches .*mar[1-10] ) OR ( type = 1 )"];
```

you should write

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:
    @"( type = 1 ) OR ( title matches .*mar[1-10] )"];
```

In the second example, the regular expression is evaluated only if the first clause is false.

Joins

In general, joins (queries that cross relationships) are also expensive operations, and you should avoid them if you can. When testing to-one relationships, if you already have—or can easily retrieve—the relationship source object (or its object ID), it is more efficient to test for object equality than to test for a property of the source object. Instead of writing the following:

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:
        @"department.name like %@", [department name]];
```

it is more efficient to write:

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:
        @"department == %@", department];
```

If a predicate contains more than one expression, it is also typically more efficient to structure it to avoid joins. For example, `@"firstName startswith[cd] 'Matt' AND (ANY directreports.paygrade <= 7)"` is likely to be more efficient than `@"(ANY directreports.paygrade <= 7) AND (firstName startswith[cd] 'Matt')"` because the former avoids making a join unless the first test succeeds.

Structuring Your Data

In some situations, there may be tension between the representation of your data and the use of predicates. If you intend to use predicates in your application, the pattern of typical query operations may influence how you structure your data. In Core Data, although you specify entities and entity-class mapping, the levels that create the underlying structures in the persistent store are opaque. Nevertheless, you still have control over your entities and the properties they have.

In addition to tending to be expensive, joins may also restrict flexibility. It may be appropriate, therefore, to de-normalize your data. In general—assuming that queries are issued often—it may be a good trade-off to have larger objects, but for it to be easier to find the right ones (and so have fewer in memory).

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Comparison of NSPredicate and Spotlight Query Strings

Both Spotlight and `NSPredicate` implement a query string syntax, and though they are similar, they differ in several respects. One query string can be converted into the other string form, as long as the common subset of functionality is used.

Spotlight and NSPredicate

There is no special relationship between Spotlight and `NSPredicate` other than that `NSMetadataQuery` is the Cocoa interface to Spotlight, and it uses `NSPredicate` in its API. The Spotlight query string syntax is similar to, but different from, the `NSPredicate` query string syntax. You can convert one query string into the other string form, as long as you use the common subset of functionality—Spotlight's query syntax is generally more limited than `NSPredicate`'s. For a complete description of the Spotlight query expression syntax, see Query Expression Syntax, and for a complete description of the `NSPredicate` string syntax, see "Predicate Format String Syntax" (page 23).

Spotlight requires comparison clauses to take the form `"KEY operator VALUE"` and does not accept `"VALUE operator KEY"`. Moreover, the kinds of attribute Spotlight accepts for `VALUE` are more limited than those accepted by `NSPredicate`. As a partial consequence of this limitation, you do not always have to quote literal strings in Spotlight queries. You can omit the quotes when `VALUE` is a string and no special operators need to be applied to it. You cannot do this with an `NSPredicate` query string, as the result would be ambiguous.

The syntax for denoting case- and diacritic-insensitivity for queries in Spotlight is different from the `NSPredicate` version. In Spotlight, you append markers to the end of the comparison string (for example, `"myAttribute == 'foo'cd"`). In `NSPredicate` strings, you use the `like` operator and prefix the markers within "[]"s (for example, `"myAttribute like[cd] 'foo'"`). In both cases, `'cd'` means case-insensitive and diacritic-insensitive. Spotlight puts the modifiers on the value, `NSPredicate` puts the modifiers on the operator.

You cannot use an `MDQuery` operator as the `VALUE` of an `NSPredicate` object `"KEY operator VALUE"` string. For example, you write an "is-substring-of" expression in Spotlight like this: `"myAttribute = '*foo*'"`; in `NSPredicate` strings you use the `contains` operator, like this: `"myAttribute contains 'foo'"`. Spotlight takes glob-like expressions, `NSPredicate` uses a different operator.

If you use "*" as left-hand-side key in a comparison expression, in Spotlight it means "any key in the item" and can only be used with ==. You could only use this expression in an `NSPredicate` object in conjunction with an `NSMetadataQuery` object.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Creating a Predicate Format String From a Spotlight Search in Finder

You can create a predicate format string from a search in Finder. Perform a search, save it, then select the folder where you saved it and choose Show Info—the Info panel shows the query that is used by Spotlight. Note however, that there are slight differences between the `NSPredicate` format string and the one stored in Finder. The Finder string might look like the following example.

```
((* = "FooBar*"wcd)) || (kMDItemTextContent = "FooBar*"cd))
    && (kMDItemContentType != com.apple.mail.emlx)
    && (kMDItemContentType != public.vcard))
```

Typically all you have to do to convert the Spotlight query into a predicate format string is make sure the predicate does not start with `*` (this is not supported by `NSMetadataQuery` when parsing a predicate). In addition, when you want to use a wildcard, you should use `LIKE`, as shown in the following example.

```
((kMDItemTextContent LIKE[cd] "FooBar")
    && (kMDItemContentType != "com.apple.mail.emlx")
    && (kMDItemContentType != "public.vcard"))
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Predicate Format String Syntax

This article describes the syntax of the predicate string and some aspects of the predicate parser.

Parser Basics

The predicate string parser is whitespace insensitive, case insensitive with respect to keywords, and supports nested parenthetical expressions. Note that the parser string is different from a string expressions passed to the regex engine. The sections describe the parser text, not the syntax for the regex engine.

Variables are denoted with a `$` (for example `$VARIABLE_NAME`). The parser does not perform semantic type checking. `?` is not a valid parser token.

The format string supports `printf`-style format arguments such as `%x` (see Formatting String Objects)). Two important arguments are `%@` and `%K`.

- `%@` is a var arg substitution for an object value—often a string, number, or date.
- `%K` is a var arg substitution for a key path.

When string variables are substituted into a format string using `%@`, they are surrounded by quotation marks. If you want to specify a dynamic property name, use `%K` in the format string, as shown in the following example.

```
NSString *attributeName = @"firstName";
NSString *attributeValue = @"Adam";
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"%K like %@",
        attributeName, attributeValue];
```

The predicate format string in this case evaluates to `firstName like "Adam"`.

Single or double quoting variables (or substitution variable strings) will cause `%@`, `%K`, or `$variable` to be interpreted as a literal in the format string and so will prevent any substitution. In the following example, the predicate format string evaluates to `firstName like "%@"` (note the single quotes around `%@`).

```
NSString *attributeName = @"firstName";
NSString *attributeValue = @"Adam";
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"%K like '%@'",
        attributeName, attributeValue];
```

Basic Comparisons

`=`, `==`

The left-hand expression is equal to the right-hand expression.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

`>=`, `=>`

The left-hand expression is greater than or equal to the right-hand expression.

`<=`, `=<`

The left-hand expression is less than or equal to the right-hand expression.

`>`

The left-hand expression is greater than the right-hand expression.

`<`

The left-hand expression is less than the right-hand expression.

`!=`, `<>`

The left-hand expression is not equal to the right-hand expression.

`BETWEEN`

The left-hand expression is between the values specified in the right-hand side.

The right-hand side is a two value array (an array is required to specify order) giving upper and lower bounds. For example, `1 BETWEEN { 0 , 33 }`, or `$INPUT BETWEEN { $LOWER, $UPPER }`.

In Objective-C, you could create a BETWEEN predicate as shown in the following example:

```
NSPredicate *betweenPredicate = [NSPredicate predicateWithFormat: @"attributeName
 BETWEEN %@",
                                    [NSArray arrayWithObjects:one, ten, nil]];
```

This creates a predicate that matches `( ( 1 <= attributeValue ) && ( attributeValue <= 10 ) )`, as illustrated in the folowing example:

```
NSNumber *one = [NSNumber numberWithInteger:1];
NSNumber *ten = [NSNumber numberWithInteger:10];
NSPredicate *betweenPredicate = [NSPredicate predicateWithFormat: @"attributeName
 BETWEEN %@",
                                    [NSArray arrayWithObjects:one, ten, nil]];

NSNumber *five = [NSNumber numberWithInteger:5];
NSDictionary *dictionary = [NSDictionary dictionaryWithObject:five
forKey:@"attributeName"];

BOOL between = [betweenPredicate evaluateWithObject:dictionary];
if (between) {
    NSLog(@"between");
}
```

Boolean Value Predicates

`TRUEPREDICATE`

A predicate that always evaluates to `TRUE`.

`FALSEPREDICATE`

A predicate that always evaluates to `FALSE`.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Basic Compound Predicates

`AND`, `&&`
: Logical AND.

`OR`, `||`
: Logical OR.

`NOT`, `!`
: Logical NOT.

String Comparisons

String comparisons are by default case and diacritic sensitive. You can modify an operator using the key characters `c` and `d` within square braces to specify case and diacritic insensitivity respectively, for example `firstName BEGINSWITH[cd] $FIRST_NAME`.

`BEGINSWITH`
: The left-hand expression begins with the right-hand expression.

`CONTAINS`
: The left-hand expression contains the right-hand expression.

`ENDSWITH`
: The left-hand expression ends with the right-hand expression.

`LIKE`
: The left hand expression equals the right-hand expression: `?` and `*` are allowed as wildcard characters, where `?` matches `1` character and `*` matches `0` or more characters. In Mac OS X v10.4, wildcard characters do not match newline characters.

`MATCHES`
: The left hand expression equals the right hand expression using a `regex`-style comparison according to ICU v3 (for more details see the ICU User Guide for Regular Expressions).

Aggregate Operations

`ANY`, `SOME`
: Specifies any of the elements in the following expression. For example `ANY children.age < 18`.

`ALL`
: Specifies all of the elements in the following expression. For example `ALL children.age < 18`.

`NONE`
: Specifies none of the elements in the following expression. For example, `NONE children.age < 18`. This is logically equivalent to `NOT (ANY ...)`.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

`IN`

Equivalent to an SQL IN operation, the left-hand side must appear in the collection specified by the right-hand side.

For example, `name IN { 'Ben', 'Melissa', 'Matthew' }`. The collection may be an array, a set, or a dictionary—in the case of a dictionary, its values are used.

In Objective-C, you could create a IN predicate as shown in the following example:

```
NSPredicate *inPredicate =
            [NSPredicate predicateWithFormat: @"attribute IN %@", aCollection];
```

where `aCollection` may be an instance of `NSArray`, `NSSet`, `NSDictionary`, or of any of the corresponding mutable classes.

`array[index]`

Specifies the element at the specified index in the array `array`.

`array[FIRST]`

Specifies the first element in the array `array`.

`array[LAST]`

Specifies the last element in the array `array`.

`array[SIZE]`

Specifies the size of the array `array`.

Identifiers

C style identifier

Any C style identifier that is not a reserved word.

#symbol

Used to escape a reserved word into a user identifier.

[\]{octaldigit}{3}

Used to escape an octal number (`\` followed by 3 octal digits).

[\][xX]{hexdigit}{2}

Used to escape a hex number (`\x` or `\X` followed by 2 hex digits).

[\][uU]{hexdigit}{4}

Used to escape a Unicode number (`\u` or `\U` followed by 4 hex digits).

Literals

Single and double quotes produce the same result, but they do not terminate each other. For example, `"abc"` and `'abc'` are identical, whereas `"a'b'c"` is equivalent to a space-separated concatenation of `a`, `'b'`, `c`.

`FALSE`, `NO`

Logical false.

`TRUE`, `YES`

Logical true.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

`NULL`, `NIL`
: A null value.

`SELF`
: Represents the object being evaluated.

`"text"`
: A character string.

`'text'`
: A character string.

Comma-separated literal array
: For example, `{ 'comma', 'separated', 'literal', 'array' }`.

Standard integer and fixed-point notations
: For example, `1`, `27`, `2.71828`, `19.75`.

Floating-point notation with exponentiation
: For example, `9.2e-5`.

`0x`
: Prefix used to denote a hexadecimal digit sequence.

`0o`
: Prefix used to denote an octal digit sequence.

`0b`
: Prefix used to denote a binary digit sequence.

Reserved Words

The following words are reserved:

`AND`, `OR`, `IN`, `NOT`, `ALL`, `ANY`, `SOME`, `NONE`, `LIKE`, `CASEINSENSITIVE`, `CI`, `MATCHES`, `CONTAINS`, `BEGINSWITH`, `ENDSWITH`, `BETWEEN`, `NULL`, `NIL`, `SELF`, `TRUE`, `YES`, `FALSE`, `NO`, `FIRST`, `LAST`, `SIZE`, `ANYKEY`, `SUBQUERY`, `CAST`, `TRUEPREDICATE`, `FALSEPREDICATE`

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

BNF Definition of Cocoa Predicates

This article defines Cocoa predicates in Backus-Naur Form notation.

NSPredicate

```
NSPredicate ::= NSComparisonPredicate | NSCompoundPredicate
    | "(" NSPredicate ")" | TRUEPREDICATE | FALSEPREDICATE
```

NSCompoundPredicate

```
NSCompoundPredicate ::= NSPredicate "AND" NSPredicate
    | NSPredicate "OR" NSPredicate
    | "NOT" NSPredicate
```

NSComparisonPredicate

```
NSComparisonPredicate ::= expression operation expression
    | aggregate_qualifier NSComparisonPredicate
```

Operations

`CONTAINS` and `IN` serve both as aggregate operators and string operators, depending on the types of their arguments.

```
operation ::= "=" | "!=" | "<" | ">" | "<=" | ">="
    | BETWEEN
    | aggregate_operations [ "[" string_options "]" ]

aggregate_operations ::= CONTAINS | IN | string_operations

string_operations ::= BEGINSWITH | ENDSWITH | LIKE | MATCHES

string_options ::= c | d | cd
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Aggregate Qualifier

```
aggregate_qualifier ::= ANY | ALL | NONE | SOME
```

Expression

```
expression ::= "(" expression ")"
    | binary_expression
    | function_expression
    | assignment_expression
    | index_expression
    | keypath_expression
    | value_expression
```

Value Expression

```
value_expression ::= literal_value | literal_aggregate
```

Literal Value

```
literal_value ::= string_value
    | numeric_value
    | predicate_argument
    | predicate_variable
    | NULL
    | TRUE
    | FALSE
    | SELF
```

String Value

```
string_value ::= "text" | 'text'
```

Predicate Argument

```
predicate_argument ::= "%" format_argument
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Format Argument

```
format_argument ::= "@" | "%" | "K"
    printf style conversion character
```

Predicate Variable

```
predicate_variable ::= "$" identifier
```

Keypath Expression

```
keypath_expression ::= identifier | "@" identifier
    | expression "." expression
```

Literal Aggregate

```
literal_aggregate ::= "{" [ expression [ "," expression ... ] ] "}"
```

Index Expression

```
index_expression ::= array_expression "[" integer_expression "]"
    | dictionary_expression   "[" expression "]"
    | aggregate_expression "[" FIRST "]"
    | aggregate_expression "[" LAST "]"
    | aggregate_expression "[" SIZE "]"
```

Aggregate Expression

```
aggregate_expression ::= array_expression | dictionary_expression
```

Assignment Expression

```
assignment_expression ::= predicate_variable ":=" expression
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Binary Expression

```
binary_expression ::= expression binary_operator expression
    | "-" expression
```

Binary Operator

```
binary_operator ::= "+" | "-" | "*" | "/" | "**"
```

Function Expression

```
function_expression ::= function_name "(" [ expression [ "," expression ... ]
] ")"
```

Function Name

```
function_name ::= "sum" | "count" | "min" | "max"
    | "average" | "median" | "mode" | "stddev"
    | "sqrt" | "log" | "ln" | "exp"
    | "floor" | "ceiling" | "abs" | "trunc"
    | "random" | "randomn" | "now"
```

Array Expression

```
array_expression ::= any expression that evaluates to an NSArray object
```

Dictionary Expression

```
dictionary_expression ::= any expression that evaluates to an NSDictionary object
```

Integer Expression

```
integer_expression ::= any expression that evaluates to an integral value
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Numeric Value

```
numeric_value ::= C style numeric constant
```

Identifier

```
identifier ::= C style identifier | "#" reserved_word
```


2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Predicate Programming Guide*.

Date	Notes
2009-02-03	Updated for iPhone OS 3.0.
2008-10-15	Clarified example of BETWEEN operator.
2006-10-03	Clarified issues with variable substitution in Predicate Format String Syntax and added an example for Booleans.
2006-05-23	Clarified the use of wildcards in the LIKE operator.
2006-04-04	Moved discussion of Core Data model-based fetch requests to "Core Data Programming Guide."
2006-03-08	Clarified need for key-value coding compliance for keys to be used in a predicate.
2006-01-10	Corrected typographical errors.
2005-11-09	Corrected minor typographical errors.
2005-10-04	Clarified relationships examples; added section on regular expressions.
2005-08-11	Added information comparing NSPredicate with Spotlight queries, and described how to use predicates with Cocoa bindings and Core Data.
2005-07-07	Corrected minor typographical errors.
2005-04-29	New document that describes how to specify queries in Cocoa.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.

2009-02-03 | © 2005, 2009 Apple Inc. All Rights Reserved.