NSExpression Class Reference

Cocoa > Data Management

2008-10-15

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

Simultaneously published in the United States and Canada.

Contents

NSExpression Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Predicate Programming Guide
Declared in	NSExpression.h
Related sample code	iSpend

Overview

`NSExpression` is used to represent expressions in a predicate.

Comparison operations in an `NSPredicate` are based on two expressions, as represented by instances of the `NSExpression` class. Expressions are created for constant values, key paths, and so on.

Generally, anywhere in the `NSExpression` class hierarchy where there is composite API and subtypes that may only reasonably respond to a subset of that API, invoking a method that does not make sense for that subtype will cause an exception to be thrown.

Expression Types

In Mac OS X v10.5, `NSExpression` introduces several new expression types: `NSSubqueryExpressionType`, `NSAggregateExpressionType`, `NSUnionSetExpressionType`, `NSIntersectSetExpressionType`, and `NSMinusSetExpressionType`.

Aggregate Expressions

The aggregate expression allows you to create predicates containing expressions that evaluate to collections that contain further expressions. The collection may be an `NSArray`, `NSSet`, or `NSDictionary` object.

For example, consider the BETWEEN operator (`NSBetweenPredicateOperatorType`); its right hand side is a collection containing two elements. Using just the Mac OS X v10.4 API, these elements must be constants, as there is no way to populate them using variable expressions. On Mac OS X v10.4, it is not possible to create a predicate template to the effect of `date between {$YESTERDAY, $TOMORROW}`; instead you must create a new predicate each time.

Aggregate expressions are not supported by Core Data.

Subquery Expressions

The `NSSubqueryExpressionType` (page 23) creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

Set Expressions

The set expressions (`NSUnionSetExpressionType` (page 23), `NSIntersectSetExpressionType` (page 23), and `NSMinusSetExpressionType` (page 23)) combine results in a manner similar to the `NSSet` methods.

Both sides of these expressions must evaluate to a collection; the left-hand side must evaluate to an `NSSet` object, the right-hand side can be any other collection type.

```
(expression UNION expression)
(expression INTERSECT expression)
(expression MINUS expression)
```

Set expressions are not supported by Core Data.

Function Expressions

On Mac OS X v10.4, `NSExpression` only supported a predefined set of functions: `sum`, `count`, `min`, `max`, and `average`. These predefined functions were accessed in the predicate syntax using custom keywords (for example, `MAX(1, 5, 10)`).

In Mac OS X v10.5, function expressions have been extended to support arbitrary method invocations as well. To use this extended functionality, you can now use the syntax `FUNCTION(receiver, selectorName, arguments, ...)`, for example:

```
FUNCTION(@"/Developer/Tools/otest", @"lastPathComponent") => @"otest"
```

All methods must take 0 or more `id` arguments and return an `id` value, although you can use the `CAST` expression to convert datatypes with lossy string representations (for example, `CAST(####, "NSDate")`). The `CAST` expression is extended in Mac OS X v10.5 to provide support for casting to classes for use in creating receivers for function expressions.

Note that although Core Data supports evaluation of the predefined functions, it does not support the evaluation of custom predicate functions in the persistent stores (during a fetch).

Tasks

Initializing an Expression

`- initWithExpressionType:` (page 19)
Initializes the receiver with the specified expression type.

Creating an Expression for a Value

`+ expressionForConstantValue:` (page 9)
Returns a new expression that represents a given constant value.

`+ expressionForEvaluatedObject` (page 9)
Returns a new expression that represents the object being evaluated.

`+ expressionForKeyPath:` (page 14)
Returns a new expression that invokes `valueForKeyPath:` with a given key path.

`+ expressionForVariable:` (page 17)
Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

Creating a Collection Expression

`+ expressionForAggregate:` (page 8)
Returns a new aggregate expression for a given collection.

`+ expressionForUnionSet:with:` (page 16)
Returns a new `NSExpression` object that represent the union of a given set and collection.

`+ expressionForIntersectSet:with:` (page 14)
Returns a new `NSExpression` object that represent the intersection of a given set and collection.

`+ expressionForMinusSet:with:` (page 15)
Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

Creating a Subquery

`+ expressionForSubquery:usingIteratorVariable:predicate:` (page 15)
Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

Creating an Expression for a Function

`+ expressionForFunction:arguments:` (page 10)
Returns a new expression that will invoke one of the predefined functions.

+ `expressionForFunction:selectorName:arguments:` (page 13)
  Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

Getting Information About an Expression

- `arguments` (page 17)
  Returns the arguments for the receiver.
- `collection` (page 17)
  Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.
- `constantValue` (page 18)
  Returns the constant value of the receiver.
- `expressionType` (page 18)
  Returns the expression type for the receiver.
- `function` (page 19)
  Returns the function for the receiver.
- `keyPath` (page 20)
  Returns the key path for the receiver.
- `leftExpression` (page 20)
  Returns the left expression of an aggregate expression.
- `operand` (page 20)
  Returns the operand for the receiver.
- `predicate` (page 21)
  Return the predicate of a subquery expression.
- `rightExpression` (page 21)
  Returns the right expression of an aggregate expression.
- `variable` (page 21)
  Returns the variable for the receiver.

Evaluating an Expression

- `expressionValueWithObject:context:` (page 18)
  Evaluates an expression using a given object and context.

Class Methods

expressionForAggregate:

Returns a new aggregate expression for a given collection.

```
+ (NSExpression *)expressionForAggregate:(NSArray *)collection
```

Parameters

collection

A collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`) that contains further expressions.

Return Value
A new expression that contains the expressions in *collection*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

expressionForConstantValue:

Returns a new expression that represents a given constant value.

```
+ (NSExpression *)expressionForConstantValue:(id)obj
```

Parameters

obj

The constant value the new expression is to represent.

Return Value
A new expression that represents the constant value, *obj*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend

Declared In
`NSExpression.h`

expressionForEvaluatedObject

Returns a new expression that represents the object being evaluated.

```
+ (NSExpression *)expressionForEvaluatedObject
```

Return Value
A new expression that represents the object being evaluated.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionForFunction:arguments:

Returns a new expression that will invoke one of the predefined functions.

```
+ (NSExpression *)expressionForFunction:(NSString *)name arguments:(NSArray
    *)parameters
```

Parameters

name

The name of the function to invoke.

parameters

An array containing `NSExpression` objects that will be used as parameters during the invocation of selector.

For a selector taking no parameters, the array should be empty. For a selector taking one or more parameters, the array should contain one `NSExpression` object which will evaluate to an instance of the appropriate type for each parameter.

If there is a mismatch between the number of parameters expected and the number you provide during evaluation, an exception may be raised or missing parameters may simply be replaced by `nil` (which occurs depends on how many parameters are provided, and whether you have over- or underflow).

Return Value

A new expression that invokes the function *name* using the parameters in *parameters*.

Discussion

The *name* parameter can be one of the following predefined functions.

Function	Parameter	Returns	Availability
`average:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the average of values in the array)	Mac OS X v10.4 and later
`sum:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the sum of values in the array)	Mac OS X v10.4 and later
`count:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the number of elements in the array)	Mac OS X v10.4 and later
`min:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the minimum of the values in the array)	Mac OS X v10.4 and later
`max:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the maximum of the values in the array)	Mac OS X v10.4 and later
`median:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the median of the values in the array)	Mac OS X v10.5 and later

Function	Parameter	Returns	Availability
`mode:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSArray` object (the mode of the values in the array)	Mac OS X v10.5 and later
`stddev:`	An `NSArray` object containing `NSExpression` objects representing numbers	An `NSNumber` object (the standard deviation of the values in the array)	Mac OS X v10.5 and later
`add:to:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the sum of the values in the array)	Mac OS X v10.5 and later
`from:subtract:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of subtracting the second value in the array from the first value in the array)	Mac OS X v10.5 and later
`multiply:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of multiplying the values in the array)	Mac OS X v10.5 and later
`divide:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`modulus:by:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the remainder of dividing the first value in the array by the second value in the array)	Mac OS X v10.5 and later
`sqrt:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the square root of the value in the array)	Mac OS X v10.5 and later
`log:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the log of the value in the array)	Mac OS X v10.5 and later
`ln:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the natural log of the value in the array)	Mac OS X v10.5 and later
`raise:toPower:`	An `NSArray` object containing two `NSExpression` objects representing numbers	An `NSNumber` object (the result of raising the first value in the array to the power of the second value in the array)	Mac OS X v10.5 and later
`exp:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the base-e exponential of the value in the aray)	Mac OS X v10.5 and later

Function	Parameter	Returns	Availability
`ceiling:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the smallest integral value not less than the value in the array)	Mac OS X v10.5 and later
`abs:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the absolute value of the value in the array)	Mac OS X v10.5 and later
`trunc:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (the integral value nearest to but no greater than the value in the array)	Mac OS X v10.5 and later
`random`	`nil`	An `NSNumber` object (a random integer value)	Mac OS X v10.5 and later
`random:`	An `NSArray` object containing one `NSExpression` object representing a number	An `NSNumber` object (a random integer value between `0` and the value in the array (exclusive))	Mac OS X v10.5 and later
`now`	`nil`	An `[NSDate]` object (the current date and time)	Mac OS X v10.5 and later

This method raises an exception immediately if the selector is invalid; it raises an exception at runtime if the parameters are incorrect.

The *`parameters`* argument is a collection containing an expression which evaluates to a collection, as illustrated in the following examples:

```
NSNumber *number1 = [NSNumber numberWithInteger:20];
NSNumber *number2 = [NSNumber numberWithInteger:40];
NSArray *numberArray = [NSArray arrayWithObjects: number1, number2, nil];

NSExpression *arrayExpression = [NSExpression expressionForConstantValue:
numberArray];
NSArray *argumentArray = [NSArray arrayWithObject: arrayExpression];

NSExpression* expression =
    [NSExpression expressionForFunction:@"sum:" arguments:argumentArray];
id result = [expression expressionValueWithObject: nil context: nil];

BOOL ok = [result isEqual: [NSNumber numberWithInt: 60]]; // ok == YES

[NSExpression expressionForFunction:@"random" arguments:nil];

[NSExpression expressionForFunction:@"max:"
    arguments: [NSArray arrayWithObject:
        [NSExpression expressionForConstantValue:
            [NSArray arrayWithObjects:
                [NSNumber numberWithInt: 5], [NSNumber numberWithInt: 10],
nil]]]];
```

```
[NSExpression expressionForFunction:@"subtract:from:"
    arguments: [NSArray arrayWithObjects:
        [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 5]],
       [NSExpression expressionForConstantValue: [NSNumber numberWithInt: 10]],
 nil]];
```

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

Availability
Available in Mac OS X v10.4 and later.

See Also
`+ expressionForFunction:selectorName:arguments:` (page 13)

Declared In
`NSExpression.h`

expressionForFunction:selectorName:arguments:

Returns an expression which will return the result of invoking on a given target a selector with a given name using given arguments.

```
+ (NSExpression *)expressionForFunction:(NSExpression *)target selectorName:(NSString
     *)name arguments:(NSArray *)parameters
```

Parameters

target
: An `NSExpression` object which will evaluate an object on which the selector identified by *name* may be invoked.

name
: The name of the method to be invoked.

parameters
: An array containing `NSExpression` objects which can be evaluated to provide parameters for the method specified by *name*.

Return Value
An expression which will return the result of invoking the selector named *name* on the result of evaluating the target expression with the parameters specified by evaluating the elements of *parameters*.

Discussion
See the description of `expressionForFunction:arguments:` (page 10) for examples of how to construct the parameter array.

Special Considerations

This method throws an exception immediately if the selector is unknown; it throws at runtime if the parameters are incorrect.

This expression effectively allows your application to invoke any method on any object it can navigate to at runtime. You must consider the security implications of this type of evaluation.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ expressionForFunction:arguments:` (page 10)

Declared In
`NSExpression.h`

expressionForIntersectSet:with:

Returns a new `NSExpression` object that represent the intersection of a given set and collection.

```
+ (NSExpression *)expressionForIntersectSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left

An expression that evaluates to an `NSSet` object.

right

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value
A new `NSExpression` object that represents the intersection of *left* and *right*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

expressionForKeyPath:

Returns a new expression that invokes `valueForKeyPath:` with a given key path.

```
+ (NSExpression *)expressionForKeyPath:(NSString *)keyPath
```

Parameters

keyPath

The key path that the new expression should evaluate.

Return Value
A new expression that invokes `valueForKeyPath:` with *keyPath*.

Availability
Available in Mac OS X v10.4 and later.

Related Sample Code
iSpend

Declared In
`NSExpression.h`

expressionForMinusSet:with:

Returns a new `NSExpression` object that represent the subtraction of a given collection from a given set.

```
+ (NSExpression *)expressionForMinusSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

left

An expression that evaluates to an `NSSet` object.

right

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value

A new `NSExpression` object that represents the subtraction of *right* from *left*.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`NSExpression.h`

expressionForSubquery:usingIteratorVariable:predicate:

Returns an expression that filters a collection by storing elements in the collection in a given variable and keeping the elements for which qualifier returns true.

```
+ (NSExpression *)expressionForSubquery:(NSExpression *)expression
    usingIteratorVariable:(NSString *)variable predicate:(id)predicate
```

Parameters

expression

A predicate expression that evaluates to a collection.

variable

Used as a local variable, and will shadow any instances of variable in the bindings dictionary. The variable is removed or the old value replaced once evaluation completes.

predicate

The predicate used to determine whether the element belongs in the result collection.

Return Value

An expression that filters a collection by storing elements in the collection in the variable variable and keeping the elements for which qualifier returns true

Discussion

This method creates a sub-expression, evaluation of which returns a subset of a collection of objects. It allows you to create sophisticated queries across relationships, such as a search for multiple correlated values on the destination object of a relationship.

For example, suppose you have an Apartment entity that has a to-many relationship to a Resident entity, and that you want to create a query for all apartments inhabited by a resident whose first name is "Jane" and whose last name is "Doe". Using only API available for Mac OS X v 10.4, you could try the predicate:

```
resident.firstname == "Jane" && resident.lastname == "Doe"
```

but this will always return false since `resident.firstname` and `resident.lastname` both return collections. You could also try:

```
resident.firstname CONTAINS "Jane" && resident.lastname CONTAINS "Doe"
```

but this is also flawed—it returns true if there are two residents, one of whom is John Doe and one of whom is Jane Smith. The only way to find the desired apartments is to do two passes: one through residents to find "Jane Doe", and one through apartness to find the ones where our Jane Does reside.

Subquery expressions provide a way to encapsulate this type of qualification into a single query.

The string format for a subquery expression is:

```
SUBQUERY(collection_expression, variable_expression, predicate);
```

where `expression` is a predicate expression that evaluates to a collection, `variableExpression` is an expression which will be used to contain each individual element of `collection`, and `predicate` is the predicate used to determine whether the element belongs in the result collection.

Using subqueries, the apartment query could be reformulated as

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe").@count
 != 0)
```

or

```
(SUBQUERY(residents, $x, $x.firstname == "Jane" && $x.lastname == "Doe")[size]
 != 0)
```

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

expressionForUnionSet:with:

Returns a new `NSExpression` object that represent the union of a given set and collection.

```
+ (NSExpression *)expressionForUnionSet:(NSExpression *)left with:(NSExpression
    *)right
```

Parameters

`left`

An expression that evaluates to an `NSSet` object.

`right`

An expression that evaluates to a collection object (an instance of `NSArray`, `NSSet`, or `NSDictionary`).

Return Value
An new `NSExpression` object that represents the union of *`left`* and *`right`*.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSExpression.h

expressionForVariable:

Returns a new expression that extracts a value from the variable bindings dictionary for a given key.

```
+ (NSExpression *)expressionForVariable:(NSString *)string
```

Parameters

string

The key for the variable to extract from the variable bindings dictionary.

Return Value
A new expression that extracts from the variable bindings dictionary the value for the key *string*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

Instance Methods

arguments

Returns the arguments for the receiver.

```
- (NSArray *)arguments
```

Return Value
The arguments for the receiver—that is, the array of expressions that will be passed as parameters during invocation of the selector on the operand of a function expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

collection

Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

```
- (id)collection
```

Return Value
Returns the collection of expressions in an aggregate expression, or the collection element of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

constantValue

Returns the constant value of the receiver.

```
- (id)constantValue
```

Return Value
The constant value of the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionType

Returns the expression type for the receiver.

```
- (NSExpressionType)expressionType
```

Return Value
The expression type for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

expressionValueWithObject:context:

Evaluates an expression using a given object and context.

```
- (id)expressionValueWithObject:(id)object context:(NSMutableDictionary *)context
```

Parameters

object

The object against which the receiver is evaluated.

context

A dictionary that the expression can use to store temporary state for one predicate evaluation.

Note that *context* is mutable, and that it can only be accessed during the evaluation of the expression. You must not attempt to retain it for use elsewhere.]

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

function

Returns the function for the receiver.

```
- (NSString *)function
```

Return Value
The function for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

initWithExpressionType:

Initializes the receiver with the specified expression type.

```
- (id)initWithExpressionType:(NSExpressionType)type
```

Parameters

type

The type of the new expression, as defined by NSExpressionType (page 22).

Return Value
An initialized NSExpression object of the type *type*.

Special Considerations

This method is the designated initializer for NSExpression.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

keyPath

Returns the key path for the receiver.

```
- (NSString *)keyPath
```

Return Value
The key path for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
NSExpression.h

leftExpression

Returns the left expression of an aggregate expression.

```
- (NSExpression *)leftExpression
```

Return Value
The left expression of a set expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.5 and later.

Declared In
NSExpression.h

operand

Returns the operand for the receiver.

```
- (NSExpression *)operand
```

Return Value
The operand for the receiver—that is, the object on which the selector will be invoked.

Discussion
The object is the result of evaluating a key path or one of the defined functions. This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

predicate

Return the predicate of a subquery expression.

```
- (NSPredicate *)predicate
```

Return Value
The predicate of a subquery expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

rightExpression

Returns the right expression of an aggregate expression.

```
- (NSExpression *)rightExpression
```

Return Value
The right expression of a set expression.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`NSExpression.h`

variable

Returns the variable for the receiver.

```
- (NSString *)variable
```

Return Value
The variable for the receiver.

Discussion
This method raises an exception if it is not applicable to the receiver.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

Constants

NSExpressionType

Defines the possible types of `NSExpression`.

```
typedef enum {
    NSConstantValueExpressionType = 0,
    NSEvaluatedObjectExpressionType,
    NSVariableExpressionType,
    NSKeyPathExpressionType,
    NSFunctionExpressionType,
    NSAggregateExpressionType,
    NSSubqueryExpressionType,
    NSUnionSetExpressionType,
    NSIntersectSetExpressionType,
    NSMinusSetExpressionType
} NSExpressionType;
```

Constants

`NSConstantValueExpressionType`
An expression that always returns the same value.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSEvaluatedObjectExpressionType`
An expression that always returns the parameter object itself.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSVariableExpressionType`
An expression that always returns whatever value is associated with the key specified by ‘variable’ in the bindings dictionary.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSKeyPathExpressionType`
An expression that returns something that can be used as a key path.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSFunctionExpressionType`
An expression that returns the result of evaluating a function.
Available in Mac OS X v10.4 and later.
Declared in `NSExpression.h`.

`NSAggregateExpressionType`
: An expression that defines an aggregate of `NSExpression` objects.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSSubqueryExpressionType`
: An expression that filters a collection using a subpredicate.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSUnionSetExpressionType`
: An expression that creates a union of the results of two nested expressions.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSIntersectSetExpressionType`
: An expression that creates an intersection of the results of two nested expressions.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

`NSMinusSetExpressionType`
: An expression that combines two nested expression results by set subtraction.

  Available in Mac OS X v10.5 and later.

  Declared in `NSExpression.h`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`NSExpression.h`

Document Revision History

This table describes the changes to *NSExpression Class Reference*.

Date	Notes
2008-10-15	Corrected SUBQUERY expression example.
2008-07-11	Corrected examples shown in the expressionForFunction:arguments: method description.
2008-02-08	Corrected errors in the table describing the expressionForFunction:arguments: method.
2007-05-13	Updated for Mac OS X v10.5.
2006-06-28	Clarified the use of expressionForFunction:arguments:.
2006-05-23	First publication of this content as a separate document.

Document Revision History

Index

R

V