NSUserDefaults

Inherits From: NSObject

Conforms To: NSObject (NSObject)

Declared In: Foundation/NSUserDefaults.h

Class at a Glance

Purpose

The NSUserDefaults class provides an programmatic interface for interacting with the OPENSTEP *defaults system*.

Principal Attributes

- A dictionary of defaults.

Creation

+ standardUserDefaults	Returns a shared instance initialized with the current user's defaults.

Commonly Used Methods

– objectForKey:	Returns the default value for the specified key.
– setObject:forKey:	Sets the default value for the specified key.
– removeObjectForKey:	Removes the default entry identified by the specified key.
– registerDefaults:	Adds the specified defaults to the NSRegistrationDomain—a cache of application-provided defaults that are used unless a user overrides them.

Class Description

The NSUserDefaults class provides a programmatic interface for interacting with the OPENSTEP *defaults system*. The defaults system allows an application to customize its behavior to match a user's preferences. For example, you can allow users to determine what units of measurement your application displays or how often documents are automatically saved. Applications record such preferences by assigning values to a set of parameters in a user's defaults database. The parameters are referred to as *defaults* since they're commonly used to determine an application's default state at startup or the way it acts by default.

A defaults database is created automatically for each user. On Unix (Mach, Solaris, and HPUX) platforms, the database is made up of a collection of files located in the **.OpenStep** directory of a user's home directory. On Windows platforms, the defaults database is stored in the Windows registry.

At run time, you use an NSUserDefaults object to read the defaults that your application uses from a user's defaults database. NSUserDefaults caches the information to avoid having to open the user's defaults database each time you need a default value. The **synchronize** method, which is automatically invoked at periodic intervals, keeps the in-memory cache in sync with a user's defaults database.

Warning: User defaults are not thread safe.

Domains

Defaults are grouped in domains. For example, there's a domain for application-specific defaults and another for system-wide defaults that apply to all applications.

Note: All defaults are stored and accessed per user. OpenStep doesn't provide for defaults that affect all users.

Each domain has a name by which it's identified and stores defaults as key-value pairs in an NSDictionary object. Each default is made up of three components:

- The domain in which the default is stored
- The name by which the default is identified (an NSString)
- The default's value, which can be any property-list object (NSData, NSString, NSArray, or NSDictionary)

A domain is either persistent or volatile. Persistent domains are permanent and last past the life of the NSUserDefaults object. Persistent domains are stored in a user's defaults database. If you use NSUserDefaults to make a changes to a default in a persistent domain, the changes are saved in the user's defaults database automatically. On the other hand, volatile domains last only as long as the NSUserDefaults object exists; they aren't saved in the user's defaults database. The standard domains are:

Domain	State
NSArgumentDomain	volatile
Application (Identified by the application's name)	persistent
NSGlobalDomain	persistent
Languages (Identified by the language names)	volatile
NSRegistrationDomain	volatile

A search for the value of a given default proceeds through the domains in an NSUserDefaults object's *search list*. Only domains in the search list are searched. The standard search list contains the domains from the table above, in the order listed. A search ends when the default is found. Thus, if multiple domains contain the same default, only the domain nearest the beginning of the search list provides the default's

value. Using the **setSearchList:** method, you can reorder the default search list or set up one that is a subset of all the user's domains.

The following sections describe the purpose of each of the domains.

NSArgumentDomain

Default values can be set from command line arguments (if you start the application from the command line) as well as from a user's *defaults database*. Default values set from the command line go in the NSArgumentDomain. They are set on the command line by preceding the default name with a hyphen and following it with a value. For example, the following command launches Project Builder and sets Project Builder's IndexOnOpen default to NO:

```
localhost> ProjectBuilder.app/ProjectBuilder -IndexOnOpen NO
```

Defaults set from the command line temporarily override values from a user's defaults database. In the example above, Project Builder won't automatically index projects even if the user's IndexOnOpen preference is set to YES in the defaults database.

Application Domain

The application domain contains application-specific defaults that are read from a user's defaults database. The application domain is identified by the name of the application, as returned by NSProcessInfo's **processName** method:

```
NSString *applicationName = [[NSProcessInfo processInfo] processName];
```

NSGlobalDomain

The global domain contains defaults that are read from a user's defaults database and are applicable to all applications that a user runs. Many Application Kit and Foundation objects use default values from the NSGlobalDomain. For example, NSRulerView objects automatically use a user's preferred measurement units, as stored in the user's defaults database under the key "NSMeasurementUnit." Consequently, ruler views in all applications use the user's preferred measurement units—unless an application overrides the default by creating an NSMeasurementUnit default in its application domain. Another NSGlobalDomain default, NSLanguages, allows users to specify a preference of languages. For example, a user could specify English as the preferred language, followed by Spanish, French, German, Italian, and Swedish.

Languages

If a user has a value for the NSLanguages default, then NSUserDefaults records language-specific default values in domains identified by the language name. The language specific domains contain defaults for a *locale*. Certain classes from the Foundation Framework (NSCalendarDate, NSDate, NSTimeZone, NSString, and NSScanner, for example) use locale defaults to modify their behavior. For example, when you request an NSString representation of an NSCalendarDate, the NSCalendarDate looks at the locale to

determine what the months and the days of the week are named in your preferred language. For more information on locale defaults, see the document “Locales” in the *Foundation Reference*.

NSRegistrationDomain

The registration domain is a set of application-provided defaults that are used unless a user overrides them. For example, the first time you run Project Builder, there isn’t an IndexOnOpen value saved in your defaults database. Consequently, Project Builder registers a default value for IndexOnOpen in the NSRegistrationDomain as a “catch all” value. Project Builder can thereafter assume that an NSUserDefaults object always has a value to return for the default, simplifying the use of user defaults.

You set NSRegistrationDomain defaults programmatically with the method **registerDefaults:**.

Synchronizing an NSUserDefaults Object with the Defaults Database

Since other applications (and the user) can write to a defaults database, the database and an NSUserDefaults object might not agree on the value of a given default at all times. Using the **synchronize** method, you can update the defaults database with an NSUserDefaults object’s new values and update the NSUserDefaults object with any changes that have been made to the database. In applications in which a run-loop is present, **synchronize** is automatically invoked at periodic intervals. Consequently, you might synchronize before exiting a process, but otherwise you shouldn’t need to.

Using NSUserDefaults

Typically, you use this class by invoking the **standardUserDefaults** class method to get an NSUserDefaults object. This method returns a global NSUserDefaults object with a search list already initialized. Use the **objectForKey:** and **setObject:forKey:** methods to get and set default values.

For example, suppose that your application needs a default that specifies whether or not to delete backup files. You could use an NSUserDefaults object to manage your default, as follows:

Set a default in the NSRegistrationDomain.

An application can set values for all its defaults in the NSRegistrationDomain. If users specify a different preference in their defaults database, the users’ preferences override the values from the NSRegistrationDomain. An NSUserDefaults object only uses values from the NSRegistrationDomain when a user hasn’t specified a different preference. So, you need to decide whether or not your application should delete backup files by default.

To register the application’s default behavior, you get the application's shared instance of NSUserDefaults and register default values with it. A good place to do this is in the **initialize** method of the class that uses the default. The following example registers the value “YES” for the default named “DeleteBackup”.

```
+ (void)initialize
{
```

```
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    NSDictionary *appDefaults = [NSDictionary
            dictionaryWithObject:@"YES" andKey:@"DeleteBackup"];

    [defaults registerDefaults:appDefaults];
}
```

The initialize message is sent to each class before it receives any other message, ensuring that the application's defaults are set before the application needs to read them.

Allow the user to specify a different default behavior.

To allow users to specify a different default behavior for deleting backups, you must provide an interface in which they can express their preference. Most applications provide a Preferences panel for this purpose. When your application detects that a user has specified a new preference, it should save it in the shared instance of NSUserDefaults.

For example, assume that your application has an instance variable called **deleteBackupButton** that is an outlet to an NSButton, and that users toggle this button's state to indicate whether or not the application should delete its backup files. Then you could use the following code to update the user's value for the DeleteBackup default:

```
if ([deleteBackupButton state]) {
    // The user wants to delete backup files.
    [[NSUserDefaults standardUserDefaults]
        setObject:@"YES" forKey:@"DeleteBackup"];
} else {
    // The user doesn't want to delete backup files.
    [[NSUserDefaults standardUserDefaults]
        setObject:@"NO" forKey:@"DeleteBackup"];
}
```

After determining the button's state, **setObject:forKey:** is used to set the value of the specified default in the application domain.

You don't have to use a Preferences panel to manage all defaults. For example, an NSWindow can store its placement in the user defaults system, so that it appears in the same location the next time the user starts the application.

Use the default value to determine behavior.

To find out whether or not to delete a backup file, you can use the following statement:

```
[[NSUserDefaults standardUserDefaults] boolForKey:@"DeleteBackup"];
```

As a convenience, NSUserDefaults provides **boolForKey:**, **floatForKey:**, and so on. Recall that a default's value can be only an NSData, NSString, NSArray, or NSDictionary. The **boolForKey:** and similarly named methods attempt to get the value for the specified default and interpret it as a different data type.

Method Types

Getting the shared instance	+ standardUserDefaults
Initializing an NSUserDefaults	– init – initWithUser:
Getting a default	– arrayForKey: – boolForKey: – dataForKey: – dictionaryForKey: – floatForKey: – integerForKey: – objectForKey: – stringArrayForKey: – stringForKey:
Setting and removing defaults	– removeObjectForKey: – setBool:forKey: – setFloat:forKey: – setInteger:forKey: – setObject:forKey:
Setting and getting the search list	– setSearchList: – searchList – dictionaryRepresentation
Maintaining persistent domains	– persistentDomainForName: – persistentDomainNames – removePersistentDomainForName: – setPersistentDomain:forName: – synchronize
Maintaining volatile domains	– removeVolatileDomainForName: – setVolatileDomain:forName: – volatileDomainForName: – volatileDomainNames
Registering defaults	– registerDefaults:

Class Methods

standardUserDefaults

+ (NSUserDefaults *)**standardUserDefaults**

Returns the shared defaults object. If it doesn't exist yet, it's created with a search list containing the names of the following domains, in this order:

- NSArgumentDomain, consisting of defaults parsed from the application's arguments
- A domain identified by the process (application) name
- NSGlobalDomain, consisting of defaults meant to be seen by all applications
- Separate domains for each of the user's preferred languages
- NSRegistrationDomain, a set of temporary defaults whose values can be set by the application to ensure that searches will always be successful

The defaults are initialized for the current user. Subsequent modifications to the standard search list remain in effect even when this method is invoked again—the search list is guaranteed to be standard only the first time this method is invoked. The shared instance is provided as a convenience; you may create custom instances with **initWithUser:** and **init**.

See also: – **init**, – **initWithUser:**

nstance Methods

arrayForKey:

– (NSArray *)**arrayForKey:**(NSString *)*defaultName*

Invokes **objectForKey:** with key *defaultName*. Returns the value associated with *defaultName* if it's an NSArray object and **nil** otherwise.

See also: –**boolForKey:**, – **dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**, **– stringForKey:**

boolForKey:

– (BOOL)**boolForKey:**(NSString *)*defaultName*

Invokes **stringForKey**: with key *defaultName*. Returns YES if the value associated with *defaultName* is an NSString containing uppercase or lowercase "YES" or responds to the **intValue** message by returning a non-zero value. Otherwise, returns NO.

See also: –**arrayForKey:**, – **dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**, **– stringForKey:**

dataForKey:

– (NSData *)**dataForKey:**(NSString *)*defaultName*

Invokes **objectForKey:** with key *defaultName*. Returns the corresponding value if it's an NSData object and nil otherwise.

See also: – **arrayForKey:**, –**boolForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**, **– stringForKey:**

dictionaryForKey:

– (NSDictionary *)**dictionaryForKey:**(NSString *)*defaultName*

Invokes **objectForKey:** with key *defaultName*. Returns the corresponding value if it's an NSDictionary object and **nil** otherwise.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**, **– stringForKey:**

dictionaryRepresentation

– (NSDictionary *)**dictionaryRepresentation**

Returns a dictionary that contains a union of all key-value pairs in the domains in the search list. As with **objectForKey:**, key-value pairs in domains that are earlier in the search list take precedence. The combined result doesn't preserve information about which domain each entry came from.

See also: – **searchList**

floatForKey:

– (float)**floatForKey:**(NSString *)*defaultName*

Invokes **stringForKey:** with key *defaultName*. Returns 0 if no string is returned. Otherwise, the resulting string is sent a **floatValue** message, which provides this method's return value.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **dictionaryForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**, **– stringForKey:**

init

– (id)**init**

Initializes defaults for the current user account and returns an NSUserDefaults instance with the argument and registration domains set up. This method doesn't put anything in the search list. Invoke it only if you've allocated your own NSUserDefaults instance instead of using the shared one.

See also: + **standardUserDefaults**

initWithUser:

– (id)**initWithUser:**(NSString *)*username*

Initializes defaults for the user account identified by *username* and returns an NSUserDefaults instance with the argument and registration domains set up. This method doesn't put anything in the search list. Invoke it only if you've allocated your own NSUserDefaults instance instead of using the shared one.

You wouldn't ordinarily use this method to initialize an instance of NSUserDefaults. It is provided for applications intended for use by a superuser who needs to update defaults databases for a number of users. The user who started the application must have appropriate access (read, write, or both) to the defaults database of the new user, or this method returns **nil**.

See also: + **standardUserDefaults**

integerForKey:

– (int)**integerForKey:**(NSString *)*defaultName*

Invokes **stringForKey:** with key *defaultName*. Returns 0 if no string is returned. Otherwise, the resulting string is sent a **intValue** message, which provides this method's return value.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **objectForKe**y:, **– stringArrayForKey:**, **– stringForKey:**

objectForKey:

– (id)**objectForKey:**(NSString *)*defaultName*

Returns the value of the first occurrence of the default identified by *defaultName*, searching the domains included in the search list in the order they're listed. Returns **nil** if the default isn't found.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**:, **– stringArrayForKey:**, **– stringForKey:**

persistentDomainForName:

– (NSDictionary *)**persistentDomainForName:**(NSString *)*domainName*

Returns a dictionary representing the persistent domain identified by *domainName*. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property-list object (NSString, NSDictionary, NSArray, or NSData).

See also: – **removePersistentDomainForName:**, – **setPersistentDomainForName:**

persistentDomainNames

– (NSArray *)**persistentDomainNames**

Returns an array containing the names of the current persistent domains. You can get each domain by using the domain names in the array as arguments to **persistentDomainForName:**.

See also: – **removePersistentDomainForName:**, – **setPersistentDomainForName:**

registerDefaults:

– (void)**registerDefaults:**(NSDictionary *)*dictionary*

Adds the contents of *dictionary* to the registration domain. If there is no registration domain, it's created using dictionary, and NSRegistrationDomain is added to the end of the search list

removeObjectForKey:

– (void)**removeObjectForKey:**(NSString *)*defaultName*

Removes the value for the default identified by *defaultName* in the standard application domain. Removing a default has no effect on the value returned by the **objectForKey:** method if the same key exists in a domain that precedes the standard application domain in the search list.

See also: – setObject:forKey:

removePersistentDomainForName:

– (void)**removePersistentDomainForName:**(NSString *)*domainName*

Removes the persistent domain identified by *domainName* from the user's defaults. The first time that a persistent domain is changed after **synchronize**, an NSUserDefaultsChanged notification is posted.

See also: – **setPersistentDomainForName:**

removeVolatileDomainForName:

– (void)**removeVolatileDomainForName:**(NSString *)*domainName*

Removes the volatile domain identified by *domainName* from the user's defaults.

See also: – **setVolatileDomainForName:**

searchList

– (NSArray *)**searchList**

Returns an array of domain names, identifying the domains that **objectForKey:** will search.

See also: – **setSearchList:**, – **dictionaryRepresentation**

setBool:forKey:

– (void)**setBool:**(BOOL)*value* **forKey:**(NSString *)*defaultName*

Sets the value of the default identified by *defaultName* to a string representation of YES or NO, depending on value. Invokes **setObject:forKey:** as part of its implementation.

See also: – **boolForKey:**

setFloat:forKey:

– (void)**setFloat:**(float)*value* **forKey:**(NSString *)*defaultName*

Sets the value of the default identified by *defaultName* to a string representation of value. Invokes **setObject:forKey:** as part of its implementation.

See also: – **floatForKey:**

setInteger:forKey:

– (void)**setInteger:**(int)*value* **forKey:**(NSString *)*defaultName*

Sets the value of the default identified by *defaultName* to a string representation of value. Invokes **setObject:forKey:** as part of its implementation.

See also: – **integerForKey:**

setObject:forKey:

– (void)**setObject:**(id)*value* **forKey:**(NSString *)*defaultName*

Sets the value of the default identified by *defaultName* in the standard application domain. Setting a default has no effect on the value returned by the **objectForKey:** method if the same key exists in a domain that precedes the application domain in the search list.

See also: – **objectForKey:**, – **removeObjectForKey:**

setPersistentDomain:forName:

– (void)**setPersistentDomain:**(NSDictionary *)*domain* **forName:**(NSString *)*domainName*

Sets the dictionary for the persistent domain named *domainName*; raises an NSInvalidArgumentException if a persistent domain with *domainName* already exists. The first time that a persistent domain is changed after synchronize, an NSUserDefaultsChanged notification is posted.

See also: – **persistentDomainForName:**, – **persistentDomainNames**

setSearchList:

– (void)**setSearchList:**(NSArray *)*array*

Sets the domains that **objectForKey:** will search. Domain names in *array* with no corresponding user default domain are ignored.

See also: – **searchList**

setVolatileDomain:forName:

– (void)**setVolatileDomain:**(NSDictionary *)*domain* **forName:**(NSString *)*domainName*

Sets the dictionary to *domain* for the volatile domain named *domainName*. This method raises an NSInvalidArgumentException if a volatile domain with *domainName* already exists.

See also: – **volatileDomainForName:**, – **volatileDomainNames**

stringArrayForKey:

– (NSArray *)**stringArrayForKey:**(NSString *)*defaultName*

Invokes **objectForKey:** with key *defaultName*. Returns the corresponding value if it's an NSArray object containing NSStrings, and **nil** otherwise.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringForKey:**

stringForKey:

– (NSString *)**stringForKey:**(NSString *)*defaultName*

Invokes **objectForKey:** with key *defaultName*. Returns the corresponding value if it's an NSString object and **nil** otherwise.

See also: – **arrayForKey:**, **–boolForKey:, – dataForKey:**, – **dictionaryForKey:**, – **floatForKey:**, – **integerForKey:**, – **objectForKey:**, **– stringArrayForKey:**

synchronize

– (BOOL)**synchronize**

Saves any modifications to the persistent domains and updates all persistent domains that were not modified to what is on disk. Returns NO if it could not save data to disk. Since the **synchronize** method is automatically invoked at periodic intervals, use this method only if you cannot wait for the automatic synchronization (for example if your application is about to exit), or if you want to update user defaults to what is on disk even though you have not made any changes.

See also: – **persistentDomainForName:**, – **persistentDomainName**s, – **removePersistentDomainForName:**, – **setPersistentDomainForName:**

volatileDomainForName:

– (NSDictionary *)**volatileDomainForName:**(NSString *)*domainName*

Returns a dictionary representing the volatile domain identified by *domainName*. The keys in the dictionary are names of defaults, and the value corresponding to each key is a property-list object (NSString, NSData, NSDictionary, NSArray).

See also: – **removeVolatileDomainForName:**, – **setVolatileDomainForName:**

volatileDomainNames

– (NSArray *)**volatileDomainNames**

Returns an array containing the names of the current volatile domains. You can get each domain by using the domain names in the array as arguments to **volatileDomainForName:**.

See also: – **removeVolatileDomainForName:**, – **setVolatileDomainForName:**

Notifications

NSUserDefaultsDidChangeNotification

Notification Object	The NSUserDefaults instance
UserInfo	None

This notification is posted the first time after a **synchronize** when a change is made to defaults in a persistent domain.