Application (MiscAppDefaults)

Declared In: <misckit/MiscAppDefaults.h>

Category Description

The MiscAppDefaults category provides a convenient interface to the Defaults mechanism.   It extends the Application class so that you can look up and set defaults information with simple messages to NXApp.   It provides methods for querying and setting string, integer and boolean defaults, converting to and from the strings required by the regular defaults mechanism.

The defaults handled by this class are all assumed to belong to the owner [NXApp appName], (your application's name) which is generally what you want.   Be warned though that [NXApp appName] may be undefined in the early stages of your program's execution, so you probably want to wait at least until an appDidInit: method before using the ±**registerDefaults:** method.   (Of course, you can always call **NXRegisterDefaults**() yourself in an +**initialize** method, as NeXT's documentation suggests.   If you do that, be sure to use your application's name as the owner of the defaults.)

One way to use these methods would be to create a defaults vector in an object's ±**appDidInit:** method and register it with ±**registerDefaults:**, like this:

```
- appDidInit:sender
```

```
{
    static NXDefaultsVector myDefaults = {
        {"BeepOnError", "YES"},
        {"NumberOfCows", 4},
        {NULL}
    };

    [NXApp registerDefaults:myDefaults];
```

and then elsewhere in your program, you could check a default setting with

```
if ( [NXApp defaultBoolValue:"BeepOnError"] ) {
        NXBeep();
}
```

and modify an existing default (perhaps in a method that handles Preferences panels) with

```
[NXApp setDefaultInt:"NumberOfCows" to: 7];
```

The Defaults system stores all values as a string. This category provides methods for handling defaults as integers and BOOLeans as well.

Note: The Preferences management system that Don Yacktman is working on will include (1) functional enhancements underneath this interface that enhance the feature set, including a ªsystemº database for setting defaults that are global for all users (and using other arbitrary defaults databases) and (2) a set of GUI tools on top of this interface that allow preferences to be dealt with straight from IB. The upshot of all this is that it is safe for you to use this interface. When Don adds in his material the interface will remain the same and only the implementation will change. It is recommended that you use this interface exclusively and stop using the NX*() functions that deal with defaults so that your app will be able to easily take advantage of the new features when they become available.

Method Types

Registering a set of defaults	± (int) registerDefaults:
Querying the defaults database	± (BOOL) defaultBoolValue: ± (int) defaultIntValue: ± (const char *)defaultValue:
Setting defaults	± (int)setBoolDefault:to: ± (int)setDefault:to: ± (int)setIntDefault:to:

Instance Methods

defaultBoolValue:

- (BOOL)**defaultBoolValue:**(const char *)*defName*

Looks up and returns the value of the application's default named *defName*, converting it to a boolean value. Returns YES if the value of the default is the string ªYESº, returns NO if its value is something else or doesn't exist.

See also: ±defaultValue: and **±setBoolDefault:to:**

defaultIntValue:

- (int)**defaultIntValue:**(const char *)*defName*

Looks up and returns the value of the application's default named *defName*, converting it to an integer. If the *defName* default doesn't exist or has a non-numeric value, returns 0.

See also: ±defaultValue: and **±setIntDefault:to:**

defaultValue:

- (const char *)**defaultValue:**(const char *)*defName*

Returns the value of the application's default named *defName*, essentially by doing **NXGetDefaultValue([NXApp appName], defName)**. Returns NULL if there is no such default.

See also: **±setDefault:to:** and **NXGetDefaultValue()**

registerDefaults:

- (int)**registerDefaults:**(const NXDefaultsVector)*v*

This method registers a set of defaults by calling **NXRegisterDefaults()**, and using **[NXApp appName]** as the owner of the defaults. Returns 1 on success, 0 on failure.

See also: **NXRegisterDefaults()**

setBoolDefault:

- (int)**setBoolDefault:**(const char *)*defName* **to:**(BOOL)*defValue*

Sets a boolean default, by converting *defValue* from a boolean to a string (either "YES" or "NO") and then calling **±setDefault:to:**. Returns 1 on success, 0 on failure.

See also: **±defaultBoolValue:** and ±**setDefault:to:**

setDefault:to:

- (int)**setDefault:**(const char *)*defName* **to:**(const char *)*defValue*

Sets the value of the default named *defName* to *defValue*, and returns 1 on success, 0 on failure.

See also: ±defaultValue: and **NXGetDefaultValue()**

setIntDefault:to:

- (int)**setIntDefault:**(const char *)*defName* **to:**(int)*defValue*

Sets an integer default, by converting *defValue* from an integer to a string and then calling **±setDefault:to:**. Returns 1 on success, 0 on failure.

See also: ±defaultIntValue: and ±**setDefault:to:**