Defined Types

NXAtom

DECLARED IN objc/hashtable.h

SYNOPSIS

typedef const char ***NXAtom**;

DESCRIPTION NXAtom is the type for a unique string. A unique string is a string that is allocated once and for all (that is, never deallocated) and that has only one representation. Unique strings can therefore be compared using the equality operator (==) rather than using **strcmp()**. A unique string should never be modified (and in fact some memory protection is done to ensure that it won't be modified). To more explicitly declare that the string has been made unique, this synonym of **const char *** has been added.

SEE ALSO **NXUniqueString()**

NXDefaultsVector

DECLARED IN defaults/defaults.h

SYNOPSIS

typedef struct _NXDefault {
    char ***name**;
    char ***value**;
} **NXDefaultsVector**[];

DESCRIPTION This structure is used by the functions **NXRegisterDefaults()** and **NXWriteDefaults()**. It provides a way to specify an open-ended list of default name/value pairs as an argument to these functions.

NXExceptionRaiser

DECLARED IN objc/error.h

SYNOPSIS

typedef void **NXExceptionRaiser**(int *code*,
    const void **data1*,
    const void **data2*);

DESCRIPTION This type is used for the function that handles exceptions raised within an exception-handling domain. In NEXTSTEP, this function is by default **NXDefaultExceptionRaiser()**.

SEE ALSO **NXDefaultExceptionRaiser()**

NXHandler

DECLARED IN objc/error.h

SYNOPSIS

```
typedef struct _NXHandler {
    jmp_buf jumpState;
    struct _NXHandler *next;
    int code;
    const void *data1, *data2;
} NXHandler;
```

DESCRIPTION This structure is used by the NEXTSTEP exception-handling system to mark nodes in the chain of exception handlers. Its fields are:

jumpState	Place to jump to using **longjmp()**
next	Pointer to next exception handler
code	Error code of exception
data1	User-defined data about the exception
data2	User-defined data about the exception

SEE ALSO **NX_RAISE()**

NXHashState

DECLARED IN objc/hashtable.h

SYNOPSIS

```
typedef struct {
    int i;
    int j;
} NXHashState;
```

DESCRIPTION This type is used for the marker passed between the functions **NXInitHashState()** and **NXNextHashState()**. Its fields may change in the future, so your code shouldn't rely on the composition of an NXHashState structure.

SEE ALSO **NXInitHashState()** and **NXNextHashState()**

NXHashTable

DECLARED IN objc/hashtable.h

SYNOPSIS

```
typedef struct {
    const NXHashTablePrototype *prototype;
    unsigned count;
    unsigned nbBuckets;
    void *buckets;
    const void *info;
} NXHashTable;
```

DESCRIPTION This type is used to identify a hash table, such as the ones returned by **NXCreateHashTable()**. Its fields are private and shouldn't be accessed.

SEE ALSO **NXCreateHashTable()**

NXHashTablePrototype

DECLARED IN objc/hashtable.h

SYNOPSIS

```
typedef struct {
    unsigned (*hash)(const void *info, const void *data);
    int (*isEqual)(const void *info, const void *data1, const void *data2);
    void (*free)(const void *info, void *data);
    int style;
} NXHashTablePrototype;
```

DESCRIPTION This type is used as one of the arguments to **NXCreateHashTable()**. Its fields specify the functions to be used for hashing, comparing, and freeing data elements:

hash	Identifies the hashing function
isEqual	Identifies the comparison function
free	Identifies the function that frees a data element
style	Reserved for future use

SEE ALSO **NXCreateHashTable()**

NXUncaughtExceptionHandler

DECLARED IN objc/error.h

SYNOPSIS

```
typedef void NXUncaughtExceptionHandler(int code,
    const void *data1,
    const void *data2);
```

DESCRIPTION This type is used for the function that handles exceptions raised outside of an exception-handling domain. In NEXTSTEP, this function can be set using **NXSetUncaughtExceptionHandler()**.

SEE ALSO **NXSetUncaughtExceptionHandler()**

NXZone

DECLARED IN objc/zone.h

SYNOPSIS

```
typedef struct _NXZone {
    void *(*realloc)(struct _NXZone *zonep, void *ptr, size_t size);
    void *(*malloc)(struct _NXZone *zonep, size_t size);
    void (*free)(struct _NXZone *zonep, void *ptr);
    void (*destroy)(struct _NXZone *zonep);
} NXZone;
```

DESCRIPTION This structure is used to identify and manage memory zones. The fields of the structure are private and subject to change in future releases; they should not be directly accessed or

altered. Use **NXCreateZone()** or a similar function to establish a new zone.

SEE ALSO **NXCreateZone()** and **NXZoneMalloc()**

Symbolic Constants

List Constants

DECLARED IN objc/List.h

SYNOPSIS NX_NOT_IN_LIST

DESCRIPTION This constant is returned by List's **indexOf:** method when it can't find the object it's passed anywhere in the List.

NXStringTable Constants

DECLARED IN objc/NXStringTable.h

SYNOPSIS
MAX_NXSTRINGTABLE_LENGTH 1024

DESCRIPTION This constant defines the maximum length for keys or values within an NXStringTable object.

Zone Constants

DECLARED IN objc/zone.h

SYNOPSIS NX_NOZONE (NXZone *)0

DESCRIPTION This constant is used as a return value by **NXCreateChildZone()**, **NXZoneFromPtr()**, and other functions to indicate the absence of a zone.

Global Variables

Command Line Arguments

DECLARED IN defaults/defaults.h

SYNOPSIS extern int **NXArgc**;
extern char **NXArgv;

DESCRIPTION These global variables pass command-line arguments to a program when it begins executing. **NXArgc** is the number of command-line arguments the program was invoked with. **NXArgv** is a pointer to an array of character strings that contain the arguments, one per string.

HashTable Prototypes

DECLARED IN objc/hashtable.h

SYNOPSIS
const NXHashTablePrototype **NXPtrPrototype**;
const NXHashTablePrototype **NXStrPrototype**;
const NXHashTablePrototype **NXPtrStructKeyPrototype**;
const NXHashTablePrototype **NXStrStructKeyPrototype**;

DESCRIPTION These global variables identify hash table prototypes suitable for use with **NXCreateHashTable()**. The first two are used for hash tables of pointers and strings, respectively. They use **NXNoEffectFree()** as the freeing function (see **NXHashTablePrototype**).

NXPtrStructKeyPrototype and **NXStrStructKeyPrototype** identify prototypes that are useful for hash tables where the key is the first element of a structure and is either a pointer or a string.

For example, **NXStrStructKeyPrototype** can be used to hash pointers to Example, where Example is:

```
typedef struct {
        char *key;
        int data1;
        ...
} Example
```

For **NXPtrStructKeyPrototype** and **NXStrStructKeyPrototype**, **NXReallyFree()** is used as the freeing function.

SEE ALSO **NXHashTablePrototype** and **NXCreateHashTable()**