NXStringTable

Inherits From:	HashTable : Object
Declared In:	objc/NXStringTable.h

Class Description

NXStringTable defines an object that associates a key with a value. Both the key and the value must be character strings. For example, these keys and values might be associated in a particular NXStringTable:

Key	**Value**
"Yes"	"Oui"
"No"	"Non"

By using an NXStringTable object to store your application's character strings, you can reduce the effort required to adapt the application to different language markets. Interface Builder give you direct access to NXStringTables, letting you create and initialize a string table and connect it into your application.

A new NXStringTable instance can be created either through Interface Builder's Classes window or through the inherited **alloc...** and **init...** methods. Similarly, you can establish the contents of an NXStringTable either directly through Interface Builder or programmatically through NXStringTable methods that read keys and values that are stored in a file (see **readFromFile:** and **readFromStream:**). Each assignment in the file can be of either of these formats:

```
"key" = "value";
"key";
```

If only *key* is present for a particular assignment, the corresponding value is taken to be identical to *key*.

A valid key or value—a valid token—is composed of text enclosed in double quotes. The text can't include double quotes (except in an escape sequence; see table) or the null character. It can include these escape sequences:

Escape Sequence	**Meaning**
\a	alert (bell)
\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\"	double quote

The backslash is stripped from any other character; consequently, numeric escape codes aren't interpreted. White space between tokens is ignored. A key or value can't exceed MAX_NXSTRINGTABLE_LENGTH characters.

The file can also include standard C-language comments; the NXStringTable ignores them. Comments can provide valuable information to a person who's translating or documenting the application.

To retrieve the value associated with a specific key, send a **valueForStringKey:** message to the NXStringTable. For example, assuming *myStringTable* is an NXStringTable containing the appropriate keys and values, this call would display an attention panel announcing a problem opening a file:

```
NXRunAlertPanel([myStringTable valueForStringKey:"openTitle"],
                [myStringTable valueForStringKey:"openError"],
                "OK",
                NULL,
                NULL);
```

If you're accessing NXStringTables through Interface Builder, please note the following. For efficiency, use several NXStringTables—each in its own interface file—rather than one large one. By using several NXStringTables, your application can load only those strings that it needs at a particular time. For example, you might place all the strings associated with a help system in an NXStringTable in one interface file and those associated with error messages in another NXStringTable in another file. When the user accesses the help system for the first time, the application can load the appropriate NXStringTable. Also, instantiate only one copy of any individual NXStringTable. Don't put an NXStringTable object in an interface file that will be loaded more than once, since multiple copies of the same table will result.

Instance Variables

None declared in this class.

Method Types

Initializing and freeing an NXStringTable	
	– init
	– free
Querying an NXStringTable	– valueForStringKey:
Reading and writing elements	– readFromFile:
	– writeToFile:
	– readFromStream:
	– writeToStream:

Instance Methods

free

– **free**

Frees the string table and its strings. You should never send a **freeObjects** (HashTable) message to an NXStringTable.

init

– **init**

Initializes a new NXStringTable. This is the designated initializer for the NXStringTable class. Returns **self**.

readFromFile:

– **readFromFile:**(const char *)*fileName*

Reads an ASCII representation of the NXStringTable's keys and values from *fileName*. The NXStringTable opens a stream on the file and then sends itself a **readFromStream:** message to load the data. See "Class Description" above for the format of the data. Returns **nil** on error; otherwise, returns **self**.

See also: – **readFromStream:**

readFromStream:

– **readFromStream:**(NXStream *)*stream*

Reads an ASCII representation of the NXStringTable's keys and values from *stream*. See "Class Description" above for the format of the data. Returns **nil** on error; otherwise, returns **self**.

See also: – **readFromFile:**

valueForStringKey:

– (const char *)**valueForStringKey:**(const char *)*aString*

Searches the string table for the value that corresponds to the key *aString*. Returns NULL if and only if no value is found for that key; otherwise, returns a pointer to the value.

writeToFile:

– **writeToFile:**(const char *)*fileName*

Writes an ASCII representation of the NXStringTable's keys and values to *fileName*. The NXStringTable opens a stream on the file and then sends itself a **writeToStream:** message. See "Class Description" above for the format of the data. Returns **nil** if an error occurs; otherwise, returns **self**.

See also: – **writeToStream:**

writeToStream:

– **writeToStream:**(NXStream *)*stream*

Writes an ASCII representation of the NXStringTable's keys and values to *stream*. See "Class Description" above for the format of the data. Returns **self**.

See also: – **writeToFile:**