Notes on the MiscParseTable routines

Module Description

The MiscParseTable functions are four functions which are useful for dealing with the ASCII table files used extensively throughout NEXTSTEP. Examples of these files are the table files inside the WorkSpace .addresses file packages, the data.classes file inside .nib file packages, and the PB.project file. The MiscParseTable functions provide a simple mechanism for reading and writing these files by translating between the ASCII file and a very specific structure (described below) built up of MiscString, MiscList, and MiscDictionary objects.

To read a table file, use **MiscParseTableFile**() or **MiscParseTableStream**(), depending upon whether you have the data available in a UNIX file or in a NXStream. To write a file from the appropriate data structure, use either **MiscWriteTableFile**() or **MiscWriteTableStream**().

The ASCII files used by NeXT can contain three types of items. First are strings. These are usually enclosed by quotes (ª"º) unless they are a single word and/or number. There are also lists of items which are enclosed by parenthesis, with commas separating each item. Finally are tables filled with key/value pairs. These are enclosed by braces (ª{}º) and on each successive line use a string for a key, followed by an equal sign (ª=º), followed by any of the three item types listed here as a value, and ending with a semicolon (ª;º). The mapping used by the MiscParseTable functions is simple; a string becomes a MiscString, a list becomes a MiscList, and a table become a MiscDictionary. (Note that the MiscDictionary object is simply a HashTable that maps string keys to object (*id*) values.) Thus, the tables

and lists can be made up of strings, tables, or lists. Included below is an attempt at the BNF grammar used by this file format.

By way of example, here is a sample file and a diagram showing what it maps to when parsed: (This file happens to be part of the PB.project file for the TreeView MiscKit example.)

```
INSTALLDIR = /LocalApps;
APPICON = Tree.tiff;
GENERATEMAIN = YES;
DOCICONFILES = (Tree2.tiff);
FILESTABLE = {
    OTHER_LIBS = (Media_s, MiscKit, NeXT_s);
    OTHER_SOURCES = (Makefile);
    OTHER_LINKED = ();
};
LOCALIZABLE_FILES = {
    InfoPanel.nib;
    TreeView.nib;
    DocWindow.nib;
};
PROJECTNAME = TreeView;
```

paste.eps ¬
Fig 1. Example file's parsed data structure.

More complex files are possible, however, since you may nest objects in lists and tablesÐmaking it possible to build lists of tables and tables within tables, for example.

Exported Functions

MiscParseTableFile

id **MiscParseTableFile**(MiscString ***fileName**);

Opens a stream on **fileName** and calls MiscParseTableStream to parse the file. Returns the id of the root MiscDictionary making up the file. Errors are printed to the console, but are not currently reported to the calling routine. The ASCII read from the file should conform to the BNF grammar given below.

MiscParseTableStream

id **MiscParseTableStream**(NXStream ***aStream**)

Uses a recursive descent parser to parse an ASCII table file from off the stream **aStream**. A MiscDictionary object, the root object in the file, is returned. Errors are printed to the console, but are not currently reported to the calling routine. The ASCII read from the stream should conform to the BNF grammar given below.

MiscWriteTableFile

int **MiscWriteTableFile**(MiscString ***fileName**, MiscDictionary ***tableRoot**)

Opens a stream on **fileName** and calls MiscWriteTableStream to create an ASCII table file. The root of the file, a MiscDictionary, should be provided in **tableRoot**. Returns zero. (In the future, an error code may be returned.) The ASCII placed in the file will conform to the BNF grammar given below.

MiscWriteTableStream

int **MiscWriteTableFile**(NXStream ***aStream**, MiscDictionary ***tableRoot**)

Uses a recursive routine to write an ASCII table file to the stream **aStream**. The root of the file, a MiscDictionary, should be provided in **tableRoot**. Returns zero. (In the future, an error code may be returned.) The ASCII placed on the stream will conform to the BNF grammar given below.

Exported Methods

These routines add categories to MiscDictionary, MiscList, and MiscString which are used to parse and write the table files. Each object is responsible to write itself as an ASCII representation or read itself from an ASCII representation. The following methods are the ones used for this purpose. They should be considered private, but may be used by your programs for other purposes and may be overridden to change functionality. Be careful subclassing these methods since some use other private methods and functions in their implementations.

± parseFromASCIIStream:(NXStream *)*aStream*

Parses an object from an ASCII stream, passed in as **aStream**. Returns **self**. The table and list versions assume that the leading ª{º or ª(º have already been read from the stream and leave the trailing ª}º or ª)º on the stream, but the string will expect the leading ª"º to still be on the stream and will remove the trailing ª"º.

± writeASCIIToStream:(NXStream *)*aStream*

Writes an object to an ASCII stream, passed in as **aStream**. Returns **self**. The ASCII written will include the appropriate delimiter characters ({}, (), and "") around the object. Since tables need to be written without the brackets in some cases, there is a **±writeInnerASCIIToStream:** method for the MiscDictionary object which will write the object without the ª{}º characters.

Known Bug

Some of the NeXT files have string *arrays* in them stored as a single string constant with \000 (NULL) as a separator withing the string constant. The current implementation of the MiscTableParse routines *strips* the NULLs out of the string. This means that if you read a file in and then write it back out, there are possible problems. The ªAddress Book Shelfº key in address books is a good example (and only known case) of this problem. The most likely fix is to have a future implementation detect these NULLs and return a MiscStringArray instead of a MiscString. Since the fix is rather involved, it could not be made ready in time for the current release.

BNF grammar for table files

Here is the BNF implemented by this code, which is more or less accurate. Start parsing with the File non-terminal. ":=" to define non-terminals and ***nil*** is used to denote when a non-terminal can reduce to nothing (ie, it is not required).

Literals (terminals) are given in unitalicized bold and use the following:

- **[]** "any one of what is inside"
- ***** "zero or more of what preceeds"
- **()** used for grouping
- **-** "anything in between" (0-9 means "0123456789", etc.)

Nonterminals are italicized. Each line below a non-terminal is a possible reduction, so, for example, an ª*Association*º could be parsed as either a string, a dictionary in braces, or a list in parenthesis. Comments are to the righ in parenthesis and italicized.

Starting non-terminal is *File*:

File:
    Dictionary

Dictionary:

Dictionary Line *(Dictionaries*
Line

Line:
String **=** *Association* **;** *(string is hash key, uniqued by parser)*
String **;** *(string and association are identical)*
nil

Association:
{ *Dictionary* **}** *(Keyed groups of objects)*
(*List* **)** *(Groups of unkeyed objects)*
String

List:
List **,** *Association* *(Lists are comma separated groups)*
Association
nil

String: *(some don't **have** to be quoted)*
" *Eliteral* **"** *(quoted string)*
Literal *(non-quoted string)*

Literal: *(non-quoted string)*
Literal **[Any char except `;' , `"', and whitespace]**
[Any char except `;' , `"', and whitespace]

Eliteral: *(quoted string)*
Eliteral ELchar
ELchar

ELchar:

[Any char except `\' and `"']	
****	*(backslash character)*
\"	*(quote character)*
\t	*(tab)*
\n	*(newline)*
\r	*(return)*
\0x ([0-9A-Fa-f])*	*(hex constant)*
\0 [0-7]*	*(octal constant)*
\ [1-9][0-9]*	*(decimal constant)*