XrmGetFileDatabase, XrmPutFileDatabase, XrmGetStringDatabase, XrmLocaleOfDatabase, XrmGetDatabase, XrmSetDatabase, XrmDestroyDatabase – retrieve and store resource databases

XrmDatabase XrmGetFileDatabase(*filename***)**
    char ****filename*;**

void XrmPutFileDatabase(*database*, *stored_db*)
    XrmDatabase *database*;
    char **stored_db*;

XrmDatabase XrmGetStringDatabase(*data*)
    char **data*;

char *XrmLocaleOfDatabase(*database*)
    XrmDatabase *database*;

XrmDatabase XrmGetDatabase(*display*)
    Display **display*;

void XrmSetDatabase(*display*, *database*)
    Display **display*;
    XrmDatabase *database*;

void XrmDestroyDatabase(*database*)
    XrmDatabase *database*;

filename	Specifies the resource database file name.
database	Specifies the database that is to be used.
stored_db	Specifies the file name for the stored database.
data	Specifies the database contents using a string.
database	Specifies the resource database.
display	Specifies the connection to the X server.

The XrmGetFileDatabase function opens the specified file, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file should contain a sequence of entries in valid ResourceLine format (see section 15.1); the database that results from reading a file with incorrect syntax is implementation-dependent. The file is parsed in the current locale, and the database is created in the current locale. If it cannot open the specified file, **XrmGetFileDatabase** returns NULL.

The **XrmPutFileDatabase** function stores a copy of the specified database in the specified file. Text is written to the file as a sequence of entries in valid ResourceLine format (see section 15.1). The file is written in the locale of the database. Entries containing resource names that are not in the Host Portable Character Encoding or containing values that are not in the encoding of the database locale, are written in an implementation-dependent manner. The order in which entries are written is implementation-dependent. Entries with representation types other than ‘‘String’’ are ignored.

The **XrmGetStringDatabase** function creates a new database and stores the resources specified in the specified null-terminated string. **XrmGetStringDatabase** is similar to **XrmGetFileDatabase** except that it reads the information out of a string instead of out of a file. The string should contain a sequence of entries in valid ResourceLine format (see section 15.1) terminated by a null character; the database that results from using a string with incorrect syntax is implementation-dependent. The string is parsed in the current locale, and the database is created in the current locale.

If database is NULL, **XrmDestroyDatabase** returns immediately.

The **XrmLocaleOfDatabase** function returns the name of the locale bound to the specified database, as a null-terminated string. The returned locale name string is owned by Xlib and should not be modified or freed by the client. Xlib is not permitted to free the string until the database is destroyed. Until the string is freed, it will not be modified by Xlib.

The **XrmGetDatabase** function returns the database associated with the specified display. It returns NULL if a database has not yet been set.

The **XrmSetDatabase** function associates the specified resource database (or NULL) with the specified display. The database previously associated with the display (if any) is not destroyed. A client or toolkit may find this function convenient for retaining a database once it is constructed.

The syntax of a resource file is a sequence of resource lines terminated by newline characters or the end of the file. The syntax of an individual resource line is:

ResourceLine	=	Comment | IncludeFile | ResourceSpec | <empty line>
Comment	=	"!" {<any character except null or newline>}
IncludeFile	=	"#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName	=	<valid filename for operating system>
ResourceSpec	=	WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName	=	[Binding] {Component Binding} ComponentName
Binding	=	"." | "*"
WhiteSpace	=	{<space> | <horizontal tab>}
Component	=	"?" | ComponentName
ComponentName	=	NameChar {NameChar}
NameChar	=	"a"−"z" | "A"−"Z" | "0"−"9" | "_" | "-"
Value	=	{<any character except null or unescaped newline>}

Elements separated by vertical bar (|) are alternatives. Curly braces ({...}) indicate zero or more repetitions of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word ‘‘include’’ must be in lowercase. The file name is interpreted relative to the directory of the file in which the line occurs (for example, if the file name contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with single ‘‘.’’ character if the sequence contains only ‘‘.’’ characters; otherwise, the sequence will be replaced with a single ‘‘*’’ character.

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any white space characters before or after the name or colon in a ResourceSpec are ignored. To allow a Value to begin with white space, the two-character sequence ‘‘\\space’’ (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence ‘‘\\tab’’ (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence ‘‘\n’’ is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence ‘‘\\newline’’ (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence ‘‘\\nnn’’, where each *n* is a digit character in the range of ‘‘0’’−‘‘7’’, is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence ‘‘\\\\’’ is recognized and replaced with a single backslash.

XrmGetResource(3X11), XrmInitialize(3X11), XrmPutResource(3X11)
Xlib – C Language X Interface