NAME

XrmGetResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource - retrieve database resources and search lists

SYNTAX

#include <X11/Xresource.h>

Bool XrmGetResource(XrmDatabase *database*, char **str_name*, char **str_class*, char ***str_type_return*, XrmValue **value_return*);

Bool XrmQGetResource(XrmDatabase *database*, XrmNameList *quark_name*, XrmClassList *quark_class*, XrmRepresentation **quark_type_return*, XrmValue **value_return*);

typedef XrmHashTable *XrmSearchList;

Bool XrmQGetSearchList(XrmDatabase *database*, XrmNameList *names*, XrmClassList *classes*, XrmSearchList *list_return*, int *list_length*);

Bool XrmQGetSearchResource(XrmSearchList *list*, XrmName *name*, XrmClass *class*, XrmRepresentation **type_return*, XrmValue **value_return*);

ARGUMENTS

class Specifies the resource class.

classes Specifies a list of resource classes.

database Specifies the database that is to be used.

list Specifies the search list returned by **XrmQGetSearchList**.

list_length Specifies the number of entries (not the byte size) allocated for list_return.

list_return Returns a search list for further use.

name Specifies the resource name.

names Specifies a list of resource names.

quark_class Specifies the fully qualified class of the value being retrieved (as a quark).

quark_name Specifies the fully qualified name of the value being retrieved (as a quark).

quark_type_return
Returns the representation type of the destination (as a quark).

str_class Specifies the fully qualified class of the value being retrieved (as a string).

str_name Specifies the fully qualified name of the value being retrieved (as a string).

str_type_return Returns the representation type of the destination (as a string).

type_return Returns data representation type.

value_return Returns the value in the database.

DESCRIPTION

The **XrmGetResource** and **XrmQGetResource** functions retrieve a resource from the specified database. Both take a fully qualified name/class pair, a destination resource representation, and the address of a value (size/address pair). The value and returned type point into database memory; therefore, you must not modify the data.

The database only frees or overwrites entries on **XrmPutResource**, **XrmQPutResource**, or **XrmMergeDatabases**. A client that is not storing new values into the database or is not merging the database should be safe using the address passed back at any time until it exits. If a resource was found, both **XrmGetResource** and **XrmQGetResource** return **True**; otherwise, they return **False**.

The **XrmQGetSearchList** function takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as **XrmGetResource** for determining precedence. If list_return was large enough for the search list, **XrmQGetSearchList** returns **True**; otherwise, it returns **False**.

The size of the search list that the caller must allocate is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is %3 sup n%, where *n* is the number of name or class components in names or classes.

When using **XrmQGetSearchList** followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to **XrmQGetSearchList**.

The **XrmQGetSearchResource** function searches the specified database levels for the resource that is

fully identified by the specified name and class. The search stops with the first match. **XrmQGetSearchResource** returns **True** if the resource was found; otherwise, it returns **False**.

A call to **XrmQGetSearchList** with a name and class list containing all but the last component of a resource name followed by a call to **XrmQGetSearchResource** with the last component name and class returns the same database entry as **XrmGetResource** and **XrmQGetResource** with the fully qualified name and class.

MATCHING RULES

The algorithm for determining which resource database entry matches a given query is the heart of the resource manager. All queries must fully specify the name and class of the desired resource (use of the characters "*" and "?" are not permitted). The library supports up to 100 components in a full name or class. Resources are stored in the database with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be the character "?") must be specified. The lookup algorithm searches the database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level before moving to the next level, until a rule selects a single entry over all others. The rules, in order of precedence, are:

1. An entry that contains a matching component (whether name, class, or the character "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).

2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using the character "?". An entry with a matching class takes precedence over entries that match using the character "?".

3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

SEE ALSO

XrmInitialize(3), XrmMergeDatabases(3), XrmPutResource(3), XrmUniqueQuark(3)
Xlib - C Language X Interface