write-resources *database pathname* &key **:write :test :test-not** Function

Writes resource bindings found in the *database* to the file given by the *pathname*. The output file is written in the standard X11 format. The **:write** function is used for writing resource values; the default is **#'princ**. The **:write** function is passed two arguments: a *resource-value* and a *stream*. The **:test** and **:test-not** functions are predicates which select resource bindings to write. For each resource binding consisting of a *resource-name* and a *resource-value*, the **:test** (or **:test-not**) function is called with the arguments *resource-name* and *resource-value*.

database — The **resource-database** to write.

pathname — A pathname of the file to write.

:write — A function for writing resource values.

:test, **:test-not** — Functions used to select which resource bindings from the resource file are merged with the *database*.

get-search-resource *table attribute-name attribute-class* Function

Returns:
value — Type **t**.

Returns the value of the resource binding in the search *table* that most closely matches the *attribute-name* and *attribute-class*. The *table* is computed by **get-search-table** and represents a set of resource bindings. The closest match is determined by the same algorithm used in **get-resource**.

The following two forms are functionally equivalent:

```
(get-resource
  database attribute-name attribute-class path-name path-class)

(get-search-resource
 (get-search-table database path-name path-class)
 attribute-name attribute-class)
```

However, the hard part of the search is done by **get-search-table**. Looking up values for several resource attributes that share the same path list can be done much more efficiently with calls to **get-search-resource**.

table — A search table returned by **get-search-table**.

attribute-name — A string or symbol giving an attribute name from a complete resource name.

attribute-class — A string or symbol giving an attribute class name from a complete resource class.

Resource Database Files

13.5 X users and application programs can save resource bindings in a file, using a standard file format shared by all X clients. The following paragraphs describe the CLX functions used to convert between the standard external format of resource files and the internal resource-database format used by application programs.

read-resources *database pathname* &key **:key :test :test-not** Function

Returns:
database — Type **resource-database**.

Reads resource bindings from a resource file in standard X11 format and merges them with the given resource *database*. The **:key** function is called to convert a file resource value into the value stored in the *database*. By default, **:key** is **#'identity**. The **:test** and **:test-not** functions are predicates that select resource bindings to merge, based on the result of the **:key** function. For each file resource binding consisting of a *resource-name* and a *resource-value*, the **:test** (or **:test-not**) function is called with the arguments *resource-name* and (**funcall** *key resource-value*).

database — The **resource-database** to merge.

pathname — A pathname for the resource file to read.

:key — A function used to convert a value from the resource file into a resource binding value.

:test, **:test-not** — Functions used to select which resource bindings from the resource file are merged with the *database*.

```
(mail * fill)                          'blue
```

Suppose an application program searched by using the following complete resource name and complete resource class:

```
(mail screen-1 reply background)
(application root button fill)
```

then the closest matching value returned would be `'red`. However, suppose the resource database contained the following resource bindings:

Resource Name	Resource Value
`(mail * background)`	`'red`
`(mail * button background)`	`'blue`

then the closest matching value returned would be `'blue`.

Resource Access Functions

13.4.3 The following paragraphs describe the CLX functions used to return a value from a resource database.

get-resource *database attribute-name attribute-class path-name path-class* Function

Returns:
value — Type **t**.

Returns the value of the resource binding in the *database* whose resource name most closely matches the complete resource name/class given by the *path-name*, *path-class*, *attribute-name*, and *attribute-class*. The lookup algorithm implements the precedence rules described previously to determine the closest match. When comparing name elements, case is significant only if both elements are strings; otherwise, element matching is case-insensitive.

database — A **resource-database**.

attribute-name — A string or symbol giving an attribute name from a complete resource name.

attribute-class — A string or symbol giving an attribute class name from a complete resource class.

path-name — The path list from a complete resource name. *path-name* and *path-class* must have the same length.

path-class — The path list from a complete resource class. *path-name* and *path-class* must have the same length.

get-search-table *database path-name path-class* Function

Returns:
search-table — Type **list**.

Returns a table containing the subset of the *database* that matches the *path-name* and *path-class*. Resources using the same *path-name* and *path-class* can be accessed much more efficiently by using this table as an argument to **get-search-resource**.

database — A **resource-database**.

path-name — The path list from a complete resource name. *path-name* and *path-class* must have the same length.

path-class — The path list from a complete resource class. *path-name* and *path-class* must have the same length.

Like any resource name list, a complete resource name consists of a path list and an attribute name. The first path list element is typically a symbol (or string) identifying the application as a whole. The second element can be a screen root identifier. Subsequent elements can be identifiers for each ancestor window of an application window. Thus, a path list typically identifies a specific window by tracing a path to it through the application window hierarchy. The final element of a complete resource name (its attribute name) is typically the name of a specific attribute of the window given by the path list (for example, `'background`). An attribute name can refer to a feature associated with the window by the application but not by the X server (for example, a font identifier). Similarly, a complete resource class typically represents a path to a window in the application window hierarchy and a specific window attribute. However, a complete resource class contains the class name for each window and for the window attribute.

For instance, in the previous example, the `mail` application can attempt to look up the value of the `background` resource for the `reply button` window by using the following complete resource name:

```
(mail screen-1 reply background)
```

and the following complete resource class:

```
(application root button fill)
```

This complete resource name contains a path list identifying the reply button window — `(mail screen-1 reply)` — and an attribute name for the window background. The corresponding resource class contains the class names for the same path list and window attribute.

Matching Resource Names

13.4.2 The resource lookup algorithm searches a specified resource data base and returns the value for the resource binding whose resource name is the closest match to a given complete resource name and class. The intent of the lookup algorithm is to formalize an intuitive notion of the closest match.

Precedence is given to a match which begins *higher* in the parent-child contact hierarchy. This allows a resource binding with a partial name to define a resource value shared by all members of a window subtree. For example, suppose the resource database contained the following resource bindings:

Resource Name	Resource Value
`(mail * background)`	`'red`
`(* reply background)`	`'blue`

Suppose an application program searched by using the following complete resource name:

```
(mail screen-1 reply background)
```

then the closest matching value returned would be `'red`.

Precedence is given to the more specific match. A name match is more specific than a class match. Either a name or class match is more specific than a wildcard match. For example, suppose the resource database contained the following resource bindings:

Resource Name	Resource Value
`(mail * background)`	`'red`

delete-resource *database name-list* Function

Removes the resource binding specified by *name-list* from the given *database*.

database — The **resource-database** containing the resource binding.

name-list — A list containing strings or symbols specifying the name for the deleted resource binding.

map-resource *database function* &rest *args* Function

Calls the function for each resource binding in the *database*. For each resource binding consisting of a *name-list* and a *value*, the form (**apply** *function name-list value args*) is executed.

database — A **resource-database**.

function — A **function** object or function symbol.

args — A list of arguments to the *function*.

merge-resources *from-database to-database* Function

Returns:
to-database —Type **resource-database**.

Merges the contents of the *from-database* with the *to-database*. **map-resource** invokes **add-resource** in order to add each resource binding in the *from-database* to the *to-database*. The updated *to-database* is returned.

from-database — The **resource-database** from which resource bindings are read.

to-database — The **resource-database** to which resource bindings are added.

Accessing Resource Values

13.4 The power and flexibility of resource management is the result of the way resource values in a resource database are accessed. A resource binding binding stored in the database generally contains only a partial resource name consisting of a mixture of name and class identifiers and wildcard elements (that is, *). To look up a resource value, an application program starts with two resource name lists of the same length containing no wildcard elements — a *complete resource name* and a *complete resource class*. The lookup algorithm returns the value for the resource binding whose resource name is the closest match to the complete name and class given. The definition of *closest match* takes into account the top-down, parent-child hierarchy of resource names and also the distinction between individual names and class names.

Complete Names and Classes

13.4.1 A resource binding contains a resource name list that can contain names, class names, or a mixture of both. A class name is a symbol or string that represents a group of related objects. The set of names used as class names are not specified by CLX. Instead, class names are defined by agreement between those who use class names when creating resource bindings (that is, users) and those who use class names when accessing resource values (that is, application programmers).

In order to access a value in a resource database, an application uses a key consisting of two items: a *complete resource name* and a *complete resource class*. A complete resource name is a resource name list containing no wildcard elements. A complete resource class is a list of exactly the same form. The distinction between a complete resource name and a complete resource class lies in how they are used to access resource bindings. The elements of a complete resource name are interpreted as names of individual objects; the elements of a complete resource class are interpreted as names of object classes. The complete resource name and class lists used in a resource database access must have the same length.

The final element of a resource name list is an attribute name. This symbol (or string) identifies a specific attribute of the object(s) named by the preceding path list. The attribute name can also be the symbol * or the string "*", in which case the resource name refers to all attributes of the path object(s). However, this form of resource name is rarely useful.

Some examples of resource bindings are shown below. In these examples, assume that `mail` is the resource name of a mail reading application. `mail` uses a window of the class `button` whose name is `reply`.

Resource Name	Resource Value
`(mail screen-1 reply background)`	`'green`
`(mail * background)`	`'red`
`(* button background)`	`'blue`

These resource bindings specify the following:

- The `background` attribute resource of `mail` application's `reply` button has the value of `green` on `screen-1`.
- The `background` attribute for the rest of the `mail` application is always `red` on all screens.
- In general, the `background` attribute for all `button` windows is `blue`.

Basic Resource Database Functions

13.3 A **resource-database** structure is a CLX object that represents a set of resource bindings. The following paragraphs describe the CLX functions used to:

- Create a resource database
- Add a resource binding
- Remove a resource binding
- Merge two resource databases
- Map a function over the contents of a resource database

make-resource-database Function

Returns:
resource-database — Type **resource-database**.

Returns an empty resource database.

add-resource *database name-list value* Function

Adds the resource binding specified by *name-list* and *value* to the given *database*. Only one value can be associated with the *name-list* in the *database*. This function replaces any value previously associated with the *name-list*.

database — The **resource-database** for the new resource binding.

name-list — A list containing strings or symbols specifying the name for the resource binding.

value — The value associated with the *name-list* in the resource binding. This can be an object of any type.

13 RESOURCES

Introduction

13.1 Users need a way to specify preferences for various user interface values (for example, colors, fonts, title strings, and so forth). Applications need a consistent method for determining the default interface values that are specific to them. It is also useful if application interface values can be modified by users without changes to the program code. For example, this capability can make it easy to change the color scheme of a user interface. In CLX, such interface values are referred to as *resources*. CLX defines functions for storing and retrieving interface resources from a resource database. A user can store various user interface values as resources in a resource database; a CLX application can then read these resource values and modify its user interface accordingly.

NOTE: The general term *resource* refers to any application user interface value stored in a resource database. The term *server resource* is used more specifically to refer to the types of objects allocated by an X server and referenced by clients (for example, windows, fonts, graphics contexts, and so forth).

Resource Bindings

13.2 Conceptually, a resource database is a set of resource name-value pairs (or *resource bindings*). The name in a resource binding is a list that is the concatenation of a *path list* and an *attribute name*.

A path list is a list of symbols (or strings) that corresponds to a path through a tree-structured hierarchy. For example, the path:

```
'(top middle bottom)
```

corresponds to a three-level hierarchy in which `middle` is the child of `top`, and `bottom` is the child of `middle`.

Typically, the path of a resource name corresponds to a path in a hierarchy of windows, and each symbol/string names a window in the hierarchy. However, the first element of the path can also represent the overall name of the entire program, and subsequent path elements can refer to an application-specific hierarchy of resource names not strictly related to windows. In addition, a resource name can contain a partially-specified path list. The asterisk symbol (*) is a wildcard that can correspond to any sequence of levels in the hierarchy (including the null sequence). For example, the path:

```
'(top * bottom)
```

corresponds to a hierarchy of two or more levels in which `top` is at the top level and `bottom` is at the bottom level. An element of a path list can be the name of an individual window or the name of a class of windows.