SINIX/windows ONLINE Documentation

SINIX/windows User Environment V3.0
XDCL Desktop Configuration Language

Edition April 1996

1 Preface

The SINIX®/windows User Environment is an OSF/Motif-based user interface for SINIX systems running SINIX V5.41 and above. It is a graphical interface with which you work interactively.
It is an integrated, consistent working environment in which you can access applications on your screen using a mouse and the keyboard. The SINIX/windows User Environment owes its consistency of appearance and behavior to the fact that it was developed on the basis of OSF/Motif™, which itself builds on the network-transparent, hardware-independent X Window System™.

1.1 Target group and manual structure

The present manual, "SINIX/windows User Environment; XDCL Desktop Configuration Language", is intended for experienced users of the graphical user interface on SINIX systems. It is a work of reference dealing with desktop configuration using the XDCL Desktop Configuration Language. Its structure reflects the structure of an XDCL file and the syntax used for desktop configuration with XDCL. The manual builds on the information provided by the manual "SINIX/windows User Environment; Guide for Experts and System Administrators".

1.2 Changes

The tables below list the functional changes made to clients between Version 2.2 and Version 3.0 of the SINIX/windows User Environment. Changes of an editorial nature and corrections of mistakes in the documentation are only listed if they are of particular significance.
New and modified functions are included in the chapters "XDCL variables", "XDCL internal methods" and "XDCL object classes and elements".
The changes are as follows:

Class	**Elements**	**Status**
iconic_object	log;type, MIME_body_part_type	New
logmanager_view	initial_poll_rate, font, small_font, log_types, methods, menubar, window_characteristics, popup_menu	New
log_type	iconic_object	New
mimemanager_view	menubar, pushbutton, methods, popup_menu, initial_poll_rate, new_window_on_navigate, font, small_font, MIME_body_part_types, mimemanager_mode	New
MIME_body_part_type	iconic_object, MIME_type_pattern, MIME_subtype_pattern, view_command, edit_command, compose_command, print_command, show_deiconified	New

Table 1: New object classes

Object manager	**Variables**	**Status**
Communication manager	%viewed_folder	New
Log manager	%LogCount	New
	%LogDate	New
	%LogEntry	New
	%LogModulName	New

	%LogNumber	New
	%LogTime	New
	%LogType	New
MIME manager	%compose_cmd	New
	%content_transfer_encoding	New
	%edit_cmd	New
	%MIME_body_part	New
	%MIME_body_part_data	New
	%MIME_subtype	New
	%MIME_type	New
	%MIME_message_label	New
	%print_cmd	New
	%view_cmd	New

Table 2: New variables

Object manager	**Internal method**	**Status**
(General)	XDsysadminMode	Chang ed
Log manager	XDLMattributes	New
	XDLMsortByDate	New
	XDLMsortByLevel	New
	XDLMsortByApplication	New
	XDLMselect	New
MIME manager	XDMIMEapplyCompose	New
	XDMIMEchangeContainer	New
	XDMIMEcomposeOK	New
	XDMIMEcreateDialog	New
	XDMIMEdeiconifyObject	New
	XDMIMEExportBodyPart	New
	XDMIMEexternalCompose	New
	XDMIMEexternalEdit	New
	XDMIMEiconifyObject	New
	XDMIMEmaybeDeiconified	New
	XDMIMEmodifyDialog	New
	XDMIMEnextMessage	New
	XDMIMEpreviousMessage	New
	XDMIMEreset	New

	XDMIMEshowInformation	New
	XDMIMETextWordWrap	New
Software manager	XDSMcheckCons	New
	XDSMdeleteObject	Changed
	XDSMdiskspaceProd	New
	XDSMinteractSelPackages	New
	XDSMinteractPackage	New

Table 3: New internal methods

Object manager	**Boolean method**	**Status**
MIME manager	XDMIMEhasComposeCommand	New
	XDMIMEhasEditCommand	New
	XDMIMEhasPrintCommand	New
	XDMIMEhasViewCommand	New
	XDMIMEisIconic	New

Table 4: New boolean methods

1.3 Notational conventions

This guide uses the following conventions:

Key names	Keys on the keyboard and mouse buttons that you have to press are displayed with their virtual key symbol in angle brackets. For example, <MCtrl> stands for the control key and <BTransfer> for the mouse button by means of which an object can be dragged.
Bold	Bold text is used to indicate the first occurrence of terms that are described in the *Glossary* at the end of this guide.
`Fixed pitch`	This font is used for screen output, examples or the names of methods and classes. The fixed pitch font is also used to display the names of files and folders.
Italics	Italics are used in commands to indicate the type of information you are to enter rather than the specific characters.
Tip	Indicates important information which you should note.
Warning	Warns you of the loss of data and of unapparent problems.

Table 5: Notational conventions

2 Configuring the desktop with XDCL

This chapter summarizes how you can configure the desktop using the desktop configuration language (XDCL). XDCL is a descriptive language; in other words, it is not an algorithmic programming language. XDCL files are interpreted by object managers on startup. The following clients are the object managers:

xdesk	Desktop manager and file manager
xdcm	Communication manager
xdfm	File manager
xdhm	Host manager
xdnm	NFS manager
xdsm	Software manager
xdum	User manager
xdvm	Volume manager, consisting of archive manager and device manager

Table 6: Object managers

The XDCL files determine the appearance and behavior of the above object managers. The standard XDCL files can be modified by system builders, system administrators and advanced users to meet their requirements of the desktop. Alternatively, they may wish to configure the desktop to correspond to the needs and abilities of a specific user or set of users.
XDCL gives you the power to:
- create new menu systems or modify existing ones
- create new file types or modify the appearance and behavior of existing ones
- determine the file manager and desktop manager characteristics
- add new desktools or modify the appearance and behavior of existing ones

The basic unit of XDCL is the **object**, for example files, folders and desktools. An object performs its own **methods** in response to a particular event taking place and usually has its own visual appearance as an icon.

2.1 XDCL file structure

An XDCL file consists of object descriptions. The XDCL-configurable clients read the XDCL description when started up. If you modify the XDCL description, the changes take effect from the next startup.
The clients interpreting XDCL files look for their XDCL_ file by searching the path specified by the `XDESK_CFG_PATH` environment variable.

The client:	**looks for the file:**
xdesk	.xdesk.cfg
xdcm	.xdcm.cfg

xdfm	.xdesk.cfg
xdhm	.xdhm.cfg
xdnm	.xdnm.cfg
xdsm	.xdsm.cfg
xdum	.xdum.cfg
xdvm	.xdvm.cfg

Table 7: Configuration files

If you do not define the `XDESK_CFG_PATH` environment variable, the clients look in the user's `$HOME` directory. If the client does not find its XDCL file in `$HOME`, it looks in `/usr/lib/X11/xdesk/cfg`.

The client:	**looks for the file:**
xdesk	xdesk.cfg
xdcm	xdcm.cfg
xdfm	xdesk.cfg
xdhm	xdhm.cfg
xdnm	xdnm.cfg
xdsm	xdsm.cfg
xdum	xdum.cfg
xdvm	xdvm.cfg

Table 8: Configuration files

If the client cannot find this either, an error box appears. When you acknowledge this dialog box, the client terminates.
Before making a custom XDCL file for a client, copy the default file from `/usr/lib/X11/xdesk/cfg` to your `$HOME` directory to use it as a template.

2.1.1 Including text from other files

An XDCL file can include other XDCL files by means of an `#include` directive, as follows:

```
#include "filename"
```

The '#' must be the first character on a new line. *filename* is the name of an XDCL file, either specified as an absolute or relative pathname. When a relative pathname or no pathname is used, `XDESK_CFG_PATH` is searched to locate the file.
The text contained within the included file is processed as though it had been physically copied into the source file at the point where the `#include` occurs.
A file that is included using this mechanism may itself include text from other files.
Recursive use of `#include` is prohibited.
The `#include` mechanism allows you to break down an XDCL description into logical components.

2.1.2 Comments

XDCL allows you to add comments to your XDCL description. You are encouraged to

comment your descriptions fully, as a sophisticated implementation can be over a 1000 lines long.
XDCL comments begin with '//' and can occupy a whole line, or simply a portion of the line. Anything to the right of the '//' is ignored. Here are two examples of legal comments:

```
/////////////////////////////////////////////////////
// This is a comment block
// that you can use to break up the XDCL description
/////////////////////////////////////////////////////
select = "external_method EMULATOR ls -l"; // call SINIX ls -l command
```

2.1.3 Parsing errors

If the XDCL files read contain any errors, diagnostic messages are printed to `stderr`.
A Motif error box is also displayed to indicate that the XDCL files contain errors. When the user acknowledges this box, the client interpreting the XDCL file exits. The client will only continue to run if the XDCL files are error-free.

2.2 XDCL objects

All entities within the desktop environment are represented as objects and all objects must have an object description. The object description defines the properties of the object. For example, some of the properties of a menu object would be pushbuttons, and the pushbutton labels, mnemonics, and methods. XDCL is the means by which each of these components is explicitly defined.
Only certain **object classes** are recognized. Some examples of object classes are pushbuttons, menus and desktools. Using XDCL you can define the properties of any object class, create new instances of object classes or edit an existing instance of an object class. The recognized object classes are listed in the chapter on "XDCL object classes and elements".

2.2.1 Object structure

XDCL describes the appearance and behavior of desktools by defining objects. An object is a basic unit that performs its own methods and usually has its own visual appearance as an icon, for example files, folders and desktools.
XDCL objects have a hierarchy. That is, some types of objects are defined in terms of other objects. For example, an XDCL menu object is defined by the **pushbuttons**, **radiobuttons** and **checkbuttons** that it contains.
It's useful at this point to introduce some terms to help us describe the hierarchy of objects:

parent	an object that contains the object you are currently considering.
child	an object contained by the object you are currently considering.
sibling	an object that has the same parent as the object you are considering.
top-level object	an object that has no parent.
ancestor	an object that is the parent, or the parent's parent (etc) of the object you are currently considering.
descendant	an object that is the child, or the child of a child (etc) of the object you are considering.

Table 9: Terms to describe object hierarchy

The following example shows an artificial hierarchy of the objects A, B, C, and D:

```
A {
B {
C {
...
}
}
D {
```

```
...
}
}
```

- A is the parent of B
- B is a child of A
- B and D are siblings
- A is a top-level object
- A and B are ancestors of C
- C is a descendent of A and B

Notice that each object starts with a '{' and ends with a '}', so that for example, A's brackets contain the whole of B's and D's definition, and B's brackets contain the whole of C's definition. The definition of the object between the brackets is known as the object body.

2.2.2 XDCL class and instance names

Each object definition has a class; that is, each object belongs to a category of objects. Examples of classes in XDCL are menubar and pushbutton. An object must belong to one of the predefined XDCL classes.
As well as a class, an object must have an instance name. Some XDCL class definitions require child objects to have specific instance names, for example the `desktop_objects` object of the `group` class as child object of an object of the `desktop_as_window` class; but often you are free to choose your own instance names for objects.
There are some rules regarding the choice of instance names:

- two siblings of the same class may not have the same instance names.
- instance names must start with a letter or the underscore '_' character.
- with the exception of the first character instance names may contain upper and lower case letters, digits and the underscore character '_'
- instance names are case-sensitive, that is 'Pb1' is different to 'pb1'

The following fragment of XDCL shows an object of class `menubar` with instance name `mb1`. This object contains an object of class `cascadebutton` with instance name `cb1`.

```
menubar mb1 {
cascadebutton cb1 {
...
}
}
```

2.2.3 Object descriptions

An object description introduces and defines the properties of an object. Each object description contains a class name, an instance name, an optional derived instance name and the object body. When creating an object description, the following rules must be observed:

- The class name must be one of the XDCL class names.
- An object is always named. The instance name has to be included. An example is shown below:
  ```
  pushbutton default_pushbutton {}
  ```
  where:
  pushbutton is the object class name;
  default_pushbutton is the instance name of the object instance.
  In a number of cases, as described in the chapter "Formal syntax of XDCL ", you can

define an instance of an object even more precisely. This entails adding the class name *class_name* to the instance name *instance_name* in an object description, using the notation "*class_name::instance_name*".
In general, x::y means "the object with the instance name y belonging to the class x".
Consider "button" and "button_1" in the following example:

```
pushbutton button {
label = "pushbutton" ;
}
radiobutton button {
label = "radiobutton" ;
}
menubar m {
cascadebutton c {
menu {
button_1 = pushbutton::button ;
}
menu {
button_2 = radiobutton::button ;
}
}
}
```

A radiobutton and a pushbutton share the same name, "button". The elements "button_1" and "button_2" use "x::y" notation to differentiate between the two instances with the same name.

- An object can be a derived object where the object is derived in part or in full from another object instance. The object description of the derived object refers to a previously defined object whose definition forms the basis for the object description. For example:

  ```
  pushbutton delete : default_pushbutton {}
  ```

  where:
  "pushbutton" is the object class name;
  "delete" is the instance name of the object instance;
  "default_pushbutton" is the instance name of the object from which the object description is derived.

It is possible to modify an existing object description. If you want to change an attribute of an already defined object, you can copy and edit the configuration file, or redefine that attribute by writing down the whole hierarchy of the particular object, including all object descriptions involved.
XDCL also allows you to override a predefined object by means of the object's instance name. An override statement starts with the keyword `override` followed by the full path through the hierarchy of the predefined objects to the object that you want to modify. You can modify the whole object or an attribute of the object. If you want to modify the whole object, you can derive the description from an existing object description. If the path names on both sides of the ':' are the same, this means that the previous description of the object is preserved. This is useful if you want to add to a description of an object.
Objects that derive from an object that is subsequently overridden are deemed to derive from the new definition of the object.
The following examples show how to override the `label` attribute of the "default_pushbutton" radiobutton which is part of the "default_menu" menu in the "default_menubar" menubar. The first example assumes that all instance names are unique.

The second example only assumes that instance names are unique within a XDCL class.

```
override default_menubar.\
default_menu.default_pushbutton.label = "modified";
override menubar::default_menubar.\
menu::default_menu.radiobutton::default_pushbutton.label = "modified";
```

2.2.4 Object attributes and their types

An object attribute describes the properties of the object. An object's definition is made up of attribute definitions and in some cases child object definitions. An attribute is something that defines the object's visual or behavioral characteristics. For example, the `font` attribute of a `pushbutton` object defines the font that will be used when displaying the pushbutton. The attributes of an object are always contained within the body of the object definition, delimited by braces. There are two ways in which an attribute can be described:

- By using the assignment operator '='. In this case the left-hand side of the assignment is the name attribute and the right-hand side is a string or an object description.
  The example below shows an object with a string to the right of the assignment. For the object named "default_pushbutton" the `font` name "-adobe-helvetica-medium-r-normal--18-180-75-75-p-98-iso8859-1" is assigned to the font attribute and the color name "black" to the `foreground` attribute.

  ```
  pushbutton default_pushbutton {
  font = "-adobe-helvetica-medium-r-normal\
  --18-180-75-75-p-98-iso8859-1";
  foreground = "black";
  }
  ```

  Tip:

  Because of the length of the `font` line, it has been split with a '\' character and continued on the next line.

  The example below shows an object with an object description to the right of the assignment. The `foreground` color of the top level object named "default_pushbutton" "default_pushbutton" is assigned to the foreground attribute of the object named "file_pushbutton".

  ```
  pushbutton file_pushbutton {
  foreground = default_pushbutton.foreground ;
  }
  ```
- By using object derivation. In the example shown below an object of class "pushbutton" is created and derived from the object "default_pushbutton" shown in the example above. All assignments for attributes of the object "default_pushbutton" result in assigning the same values to the same attribute of the object "rename". Only assignments for attributes in the object description of the object "rename" will override those values for the object "rename". In the example shown below the font of the object "default_pushbutton" is assigned to the font attribute of the object "rename" by derivation.
  The `foreground` attribute of the object "default_pushbutton" is overriden by the assignment of the color "red" to the foreground attribute in the object description of the object "rename". The string "Rename..." is assigned to the `label` attribute of the object "rename".

```
pushbutton umbenennen : default_pushbutton {
label = "Rename..." ;
foreground = "red" ;
}
```

An object attribute always has one of the following types:

boolean

indicating one out of two string values. Valid variations are "True", "False", "Yes", "No", "On", "Off". The case of letters is ignored.
Another way to specify a boolean value is to use a boolean method in the following form:

`boolean_method function`

`function`
is a boolean function that is defined in the application when the configuration file is read. The value of the attribute is defined by the return value of the function.

comparative

indicating a numeric comparison of the following form:

`comparative_operator { number | age }`

`comparative_operator`
is one of `=`, `<`, `>`, `<=`, `>=`.

`number`
is a positive or negative integer between
-4,294,947,295 (= - (2^{32} - 1)) and
+4,294,947,295 (= + (2^{32} - 1)).

`age` is a string in the following format:

```
[YY[  ]"years"  ][WW[  ]"weeks"  ][DD[  ]"days"  ]
[HH[  ]"hours"  ][MM[  ]"minutes"  ][SS[  ]"seconds"]
```

Each units string (years, weeks, days, hours, minutes, seconds) can be abbreviated to its singular form (year, week, day, hour, minute, second), or as its initial letter (y, w, d, h, m, s). The total value used for the age is the sum of all the components: no limits are placed on the values, so, for example, the user can specify a comparison such as `> 1 day 100 seconds`. A year is assumed to consist of 365 days.

file type

indicating one of the following string values: "file", "directory", "char_special", "fifo", "block_special". The case of letters is ignored.

geometry

indicating a string value in the format `width'x'height'+'x_off'+'y_off` or `auto`.
If `auto` is specified, then the geometry is specified by the controlling application.

`width` is the window width.

`height`
is the window height.

`x_off` Position of window on x axis (horizontally) in pixels

+x_off

distance between left edge of window and screen

-x_off

distance between right edge of window and screen

`y_off` Position of window on y axis (vertically) in pixels

+y_off

distance between top edge of window and screen

-y_off

distance between bottom edge of window and screen

magic

indicating an entry in the file header of the file. The format corresponds to entries in the system file `/etc/magic`.

Mapping

indicating, depending on where it is used, one of the following strings: "_", "user", "host", "group", "network" and "none". This attribute allows a name to be assigned to the file name of an image. For example, a user name can be assigned to the file name of a bitmap file, which contains an image representing this user.
Example: In the case of the mail manager, an image is assigned to a sender.

```
image image {
name = {%sender_id}.64;
mapping= "user";      // Uses the file user_mapping to obtain
// the name of the image file
}
```

method

indicating a string value or an instance of the XDCL class `nls_string`.
Valid formats for values of type `method` are

```
"external_method external_name"
"internal_method internal_name"
"shell_method command"
```

`external_method`, `internal_method` and `shell_method` are keywords and must be entered as such. They define the type of method you want to use.

`external_name`

is a string defining an application and its arguments. The application interpreting the XDCL file starts a new process to execute `external_name`. Where such applications not being clients require user input, they should be started from a suitable terminal emulator.

`internal_name`

is a string defining an XDCL internal method followed by its arguments.

`command`

is a shell command passed to the shell defined by the SHELL environment variable or to /bin/sh if SHELL is undefined. If the command expects user input, it should be run from a suitable terminal emulator.

number

indicating a numeric value of the primitive class (see the chapter "Formal syntax of XDCL ").

network name

indicating a valid network name, i.e. one of those in `/etc/networks`.

panel ID

defining the panel (window area) in which an icon is inserted if the object manager view includes several panels. The panel ID serves to arrange icons in the window according to logical criteria. Objects for which no panel attribute is set are inserted in the lowest panel. No more than ten panels can be displayed at once.
An attribute of the type panel ID has the following format:
`base_number`

`base` is the name of a panel that is available to an object manager for displaying icons.

`number`
is a numeric value.

Panels are created in the window of an object manager in the order of their appearance in the XDCL description. Panels whose panel IDs have the same base are sorted on the basis of their number component and arranged accordingly in the window. The numbers are sorted as characters rather than numbers, so that xd-1 is greater than xd-02, for example. Panels with a base component that appears for the first time are added at the end; all other panels are inserted at the correct position.

Example:

A number of objects with the panel IDs xdfm-3, xdfm-1, xdesk-2, xdesk-1 and xdfm-02 appear in the XDCL description in the specified order. When you choose "Split Browser" from the "View" menu, the panels are output as follows:
1st panel xdfm-02
2nd panel xdfm-1
3rd panel xdfm-3
4th panel xdesk-1
5th panel xdesk-2

mail address

indicating a valid mail address of the mail protocol used.

sensitivity kind

indicating a string value. Valid strings are:

`menu_sensitivity`
the object inherits the menu_sensitivity of the view containing it.

`always`
the object is always available for selection.

`never` the object is never available for selection.

`"items_selected" comparative_operator number`
compares the number of selected items to the number given in the string.
`comparative_operator` is one of `=`, `<`, `>`, `<=`, `>=`.
`number` is a positive or negative integer between
-4,294,947,295 (= - (2^{32} - 1)) and
+4,294,947,295 (= + (2^{32} - 1)).

`"boolean_method" function`
is a boolean internal method.
`function` is a boolean function that is defined in the application when the configuration file is read. The value of the attribute is defined by the return value of the function.

shell regexp

indicating a shell regular expression string as defined for the SINIX shell command.

SINIX user group name
indicating a name valid for a user group, i.e. one of the group names specified in `/etc/group`.

SINIX filename
indicating a valid SINIX filename of up to 255 characters.

SINIX pathname
indicating a valid SINIX pathname.

string
indicating an identifier of the primitive class.
(See the chapter "Formal syntax of XDCL ").

X atom list
indicating a list of atoms. As defined in the ICCCM (Inter-Client Communications Conventions Manual), atoms are unique names that clients can use amongst themselves when communicating. The X atom list value contains a list of object characteristics that can be exported by a client when an object is dragged and imported when an object is dropped. These characteristics are referred to as targets. The various elements in the list are separated from each other by space characters. The use of targets is described in detail in the"chapter "Dragging and dropping ".
The following targets are supported:

ICCCM targets

Target	**Contents**	**Meaning**
FILE_NAME	absolute_pathname	Any file
USER	user	User name
HOST	host	Host name
STRING	ISO Latin 1 string	Text
TEXT	string	Text

Table 10: ICCCM targets

Targets in SINIX/windows

Target	**Contents**	**Meaning**
_SNI_FILE	user@host:absolute_pathname	Any file
_SNI_FILE_REG	user@host:absolute_pathname	Regular file
_SNI_FILE_DIR	user@host:absolute_pathname	Directory
_SNI_FILE_BLK	user@host:absolute_pathname	Block special file

_SNI_FILE_CHR	user@host:absolute_pathname	Character special file
_SNI_FILE_FIFO	user@host:absolute_pathname	Fifo file
_SNI_FILE_EXEC	user@host:absolute_pathname	Executable file
_SNI_FILE_Z	user@host:absolute_pathname	Compressed file
_SNI_DESKTOOL	user@host:desktool_object_ID	Desktool
_SNI_TOOLBOX	user@host:toolbox_object_ID	Toolbox
_SNI_MAIL_MSG	user@host:absolute_pathname:number	Message
_SNI_MAIL_FOLDER	user@host:absolute_pathname	Mail folder
_SNI_USER	user@host:group:user	User
_SNI_GROUP	user@host:group	User group
_SNI_HOST	user@host:network:host	Host
_SNI_NETWORK	user@host:network	Network
_SNI_ARCHIVE	user@host:local cmd\0remote cmd	Archived file
_SNI_MIME_BODY_PART	user@host:absolute_pathname	Regular file containing the contents of the MIME body part

Table 11: Targets in SINIX/windows

X color name
: indicating a string color specification acceptable to the X server used.

X font name
: indicating a string font specification acceptable to the X server used.

XDCL object reference
: indicating an object referenced in the XDCL description of an object manager by means of an entry in the following form:

```
reference::=       component{'.'component}
component::=       [class_name'::']instance_name
class_name::=      IDENTIFIER
instance_name::=   IDENTIFIER
```

(See the chapter "Formal syntax of XDCL ").

2.2.5 Object derivation

Derivation is a feature of XDCL that enables you to re-use objects you have already defined to define new object instances. Derivation allows you to take a particular object definition and selectively override some of the behavior of the object.
To take a simple example, say you want to define all your pushbuttons to have the same foreground and background colors and font. You could start by defining a default pushbutton that simply exists to define these common settings:

```
pushbutton default_pb {
foreground = "black";
background = "cyan";
font = "gs-13";
}
```

Later in our XDCL description you can derive a new pushbutton instance from the default one. In a typical system all pushbuttons in the menu structure would derive from a common default pushbutton.

```
pushbutton a_pb : default_pb {
label = "A pushbutton";
}
```

The line:

```
pushbutton a_pb : default_pb
```

means the object of class `pushbutton` with instance name `a_pb` is derived from the object with instance name `default_pb`.
The above object definition is equivalent to:

```
pushbutton a_pb {
foreground = "black";
background = "cyan";
font = "gs-13";
label = "A pushbutton";
}
```

but using the derivation syntax gives us several advantages:

- The XDCL description is more maintainable. If you want to change the font of all pushbuttons, you only need to change the `default_pb` object definition.
- Less memory is used. As the XDCL description is read in at startup time, the amount of memory required by the client interpreting the XDCL files depends on the XDCL description. By writing more compact XDCL, you reduce the memory overhead required.

Specifying the derived-from object

The above example uses the ':' modifier to specify the object from which the current object is derived. The instance name of the derived-from object follows the ':'.
In certain circumstances, just specifying the instance name of the object you want to derive from can be ambiguous, as in the following case:

```
pushbutton default {
...
}
```

```
cascadebutton default {
...
}
```

It is legal (though not recommended) to use the same instance name for sibling objects, as long as their class is different. However, deriving from one of these objects causes problems. Just using the instance name 'default' is not sufficient to identify the object. This problem can be overcome by also including the class name of the object you want to derive from, as follows:

```
pushbutton a_pb : pushbutton::default {
...
}
```

By using a specification of the form *className*::*instanceName* the ambiguity is resolved.

Tip:

This ambiguity only comes about when siblings do not have unique names.
To keep the derivation specification simple, it is therefore a good idea always to choose unique names for siblings.

Further complexity is introduced when you start to deal with more than a single level of hierarchy. For example, to derive from an object that is a child of another object:

```
menu menu1 {
pushbutton pb1 {
...
}
...
}
menu menu2 {
pushbutton pb2 {
...
}
}
```

To derive from the pushbutton that is a child of `menu1`, you need to specify a complete path through the hierarchy to the object:

```
pushbutton a_pb : menu1.pb1 {
...
}
```

Each level in the hierarchy is separated by a '.'. With the optional class name the specification looks like this:

```
pushbutton a_pb : menu::menu1.pushbutton::pb1 {
...
}
```

You can also use the derivation mechanism to derive attributes of objects from an equivalent attribute of another object. For example:

```
pushbutton default_pb {
font = "gs-13";
}
cascadebutton default_cb {
font = default_pb.font;
}
```

In the above XDCL fragment the cascadebutton defines its font to be that of the pushbutton.

Using derivation to extend object definitions

Certain XDCL object classes allow you to have any number of a certain class of child. For example a menu object can have any number of pushbutton children. A feature of derivation is to allow you to extend the number of such children.

Starting with a menu definition:

```
menu menu1 {
pushbutton A {
...
}
pushbutton B {
...
}
pushbutton C {
...
}
}
```

then `menu1` would be displayed containing the pushbuttons `A`, `B` and `C`. This menu could be extended as follows:

```
menu menu2 : menu1 {
pushbutton D {
...
}
pushbutton E {
...
}
}
```

`menu2` would be displayed containing the pushbuttons `A`, `B`, `C`, `D` and `E`; `A`, `B` and `C` being inherited from `menu1`. To be clear what this means, contrast this with the following XDCL fragment:

```
menu menu2 : menu1 {
pushbutton A {
label = "A new label";
}
}
```

In this case, `menu2` inherits `A`, `B`, and `C` but pushbutton `A` is completely redefined in `menu2`.

Taking this a stage further, to derive the full definition of pushbutton `A`, and override the `label` attribute, you must add the following:

```
menu menu2 : menu1 {
pushbutton A :menu1.A {
label = "A new label";
}
}
```

2.2.6 Overriding object descriptions

It is possible to modify an existing object description. If you want to change an attribute of an already defined object, you can copy and edit the configuration file, or redefine that attribute by writing down the whole hierarchy of the particular object, including all object descriptions involved.

XDCL also allows you to override a predefined object by means of the object's instance name. An override statement starts with the keyword `override` followed by the full path

through the hierarchy of the predefined objects to the object that you want to modify. You can modify the whole object or an attribute of the object. If you want to modify the whole object, you can derive the description from an existing object description. If the path names on both sides of the ':' are the same, this means that the previous description of the object is preserved. This is useful if you want to add to a description of an object.
Objects that derive from an object that is subsequently overridden are deemed to derive from the new definition of the object.
The following examples show how to override the `label` attribute of the "default_pushbutton" radiobutton which is part of the "default_menu" menu in the "default_menubar" menubar. The first example assumes that all instance names are unique. The second example only assumes that instance names are unique within a XDCL class.

```
override default_menubar.\
default_menu.default_pushbutton.label = "modified";
```

```
override menubar::default_menubar.\
menu::default_menu.radiobutton::default_pushbutton.label = "modified";
```

2.3 XDCL methods and variables

A method is an action that is associated with an object; that is, it defines what the object does in response to a particular event taking place. For example a pushbutton object has a `select` method. This defines what happens when the user selects that pushbutton from a menu.
Having designed an XDCL object you must also define what the object is to do, which means you have to define methods. There are three types of method that can be used when writing XDCL configuration files:

- **internal methods** which use routines internally understood by the client interpreting the XDCL description. They are accessed more rapidly than external methods because they are part of the client.
- **external methods** which call routines external to the client. These are generally in the form of special SINIX shell scripts or SINIX commands. You can also write your own shell scripts. There are more external methods available to you than internal ones. Usually external methods are accessed more slowly than internal methods.
- **shell methods** which transfer shell commands to the shell specified in the SHELL environment variable, or to `/bin/sh` if the environment variable is not defined.

XDCL's internal and external methods are documented in the chapter "XDCL internal methods " and in the chapter "XDCL external methods " of this manual.

2.3.1 Defining methods

Method attributes can either call internal methods within the executing application or external methods. All methods have the form:
methodName = `"internal_method` *methodDescription*`";`
or:
methodName = `"external_method` *methodDescription*`";`
or:
methodName = `"shell_method` *methodDescription*`";`
The restrictions on the use of the XDCL internal methods as documented in the chapter "XDCL internal methods " should be noted with care.
External methods can be any SINIX program or shell script, but the client interpreting the XDCL description must be able to find the program on your path $PATH. External methods allow you to customize the desktop to call your own programs from the menu system, from pop-up menus on icons, or by direct interaction with an icon, such as double-clicking or dragging and dropping.
The following very simple example defines a pushbutton, which when selected by the user calls the SINIX program `ls`.

```
pushbutton list_pb {
label = "List";
select = "external_method EMULATOR ls -l";
}
```

2.3.2 Using variables

XDCL allows you to use SINIX environment variables in method definitions. The previous example could be modified to list the files in your home directory, as follows:

```
pushbutton list_pb {
```

```
label = "List";
select = "external_method EMULATOR ls -l {$HOME}";
}
```

Notice that the HOME environment variable starts with a '$' and is enclosed in braces.
Use of environment variables lets you write more general XDCL scripts that can be tailored to suit individual needs by defining a number of environment variables. As a system administrator, you may consider this as a method of providing customized configurations of the desktop to a particular user or group of users.
In addition to SINIX environment variables, the client interpreting XDCL descriptions provides a number of XDCL variables that are substituted for a given value at runtime. XDCL variables, like environment variables, are enclosed in brackets, but start with a '%' character.
XDCL variables fall into two sub-categories: static and dynamic. Static variables are variables that are fixed and do not vary during the runtime. Dynamic variables change during runtime, depending on the current selections, which folder is being viewed and so on.
To illustrate the difference between static and dynamic variables here are some examples. The title of a file manager view in the vanilla configuration is defined to be:

```
filemanager_view fmv {
window_characteristics {
title = "{%hostname}:{%viewed_folder}";
}
...
}
```

The variable `%hostname` is a static variable that is substituted with the name of the host machine on which the client interpreting the XDCL description is running.
The variable `%viewed_folder` is a dynamic variable that is substituted with the name of the folder you are currently viewing. As you navigate around the filesystem, this variable is dynamically updated.
You could customize your file manager windows to give you a more personal window title, such as:

```
title = "{%this_user} is viewing {%hostname}:{%viewed_folder}";
```

A group of XDCL dynamic variables are specially related to file characteristics, for example `%filename`, `%pathname`, `%size` and so on. These variables are substituted with the characteristics of a file. These variables are discussed in the chapter "XDCL variables ".
Dynamic variables must be used in the correct context within your XDCL description. For example, `%dropped_files` is expanded to be a list of files dropped on an object. It therefore only has meaning in the context of the user dropping a group of files. To use it for a pushbutton label would be completely meaningless. During startup, the client interpreting the XDCL description checks that you are using variables that it recognizes. However, no checks can be made to ensure you are using the variables sensibly. If the client cannot expand a dynamic variable sensibly it leaves the variable unexpanded in the string.

2.4 A tour of the XDCL classes

This section introduces the XDCL classes, showing you how the classes interrelate and build to make a complete specification for your desktop.

2.4.1 Views and the xdesk object

There are three classes that define the types of main window that the desktop manager and the file manager use. There is also one class each that defines window types for the mail manager, host manager, NFS manager, user manager, and volume manager. These classes are collectively referred to as `view` classes, as they define the way you view the file system, the desktools, your mail, hosts and login names. archives, devices, mount and share point, and the desktools. The examples given here concern only the three classes for the file manager and the desktop manager.

"filemanager_view" class

A filemanager view is a Motif window that displays the files in a given folder as icons and has a menubar with options for the user to select.

The `filemanager_view` class defines:

- the menubar
- window characteristics that determine what the window looks like; for example, whether or not it has a close box, what the title of the window is and so on.
- the mechanism by which files in the filesystem are 'typed' - that is, how a file's characteristics are matched to a particular kind of icon.
- a pop-up menu, invoked when the user clicks on the window background.
- special attributes that define the behavior of the filemanager view; for example, whether or not a new window is opened when you choose to view another folder.

In a skeleton form, a typical `filemanager_view` object looks like this:

```
filemanager_view fmv1 {
// Child objects
menubar {
...
}
window_characteristics {
...
}
group file_types {
...
}
popup_menu {
...
}
// Attributes
new_window_on_navigate = "False";
...
}
```

Compare the above with the definition of the `filemanager_view` class, to see how a class definition looks when written out in XDCL.

Defining menubars, pop-up menus and file types is discussed later in this section. Of interest here are the special attributes of the filemanager view.

The `new_window_on_navigate` attribute determines whether or not a new filemanager view is

created when you navigate to another folder by double-clicking on a folder icon.
The `initial_folder` attribute indicates the folder to be displayed when the filemanager view is first opened.
The `navigate_up_past_initial_folder` attribute determines whether the parent folder is displayed when you view the folder on which the filemanager view was originally opened. Using this attribute together with the `initial_folder` attribute allows you to restrict a user to a particular area of the filesystem. For example, setting the attributes as follows:

```
initial_folder = "/home/project/fred";
navigate_up_past_initial_folder = "False";
```

means that the initial folder displayed by the file manager is `/home/project/fred`. The user can navigate down to folders below `/home/project/fred` but cannot view higher level directories.
The `initial_poll_rate` attribute determines how often the file manager view re-reads the contents of the folder it is currently viewing. Polling the filesystem enables the file manager to keep the user up-to-date with changes to the files being viewed. Obviously, regularly polling the filesystem consumes system resources and so you should set the poll rate to a value that does not place too greater burden on the system. If the folders you work with do not change very often you may consider setting the poll rate to 60 seconds or longer. Because the poll rate can be set independently for each filemanager view, you can choose settings that are appropriate to the part of the filesystem you are viewing. This attribute combines with the `Poll rate...` item on the `Options` menu, which allows the user to modify the poll rate interactively.

"desktop_as_window" and "desktop_as_root" classes

These classes define the desktop manager. It is possible to display the desktop icons on the root window, or within a normal decorated window.
The `desktop_as_window` class defines:

- the menubar.
- window characteristics that determine what the window looks like; for example, whether or not it has a close box, what the title of the window is and so on.
- the mechanism by which files in the filesystem are 'typed' - that is, how a file's characteristics are matched to a particular kind of icon. This is required, because the desktop has to know how to type files that are placed on the desktop.
- a pop-up menu, invoked when the user clicks on the window background
- the objects that appear on the desktop - these can be toolboxes with desktools or they can be file references. A toolbox can contain further toolboxes.
- a set of methods that define what happens when a user drops an object onto the desktop.
- special attributes that define the behavior of the desktop object.

In a skeleton form, a typical `desktop_as_window` object looks like this:

```
desktop_as_window daw1 {
// Child objects
menubar {
...
}
window_characteristics {
...
}
group file_types {
...
}
popup_menu {                  // when click on background
...
}
toolbox {                     // toolbox and file refs
...
}
methods {                     // drop methods for desktop
...
}
// Attributes
initial_poll_rate = 20;       // poll every 20 seconds
...
}
```

The `desktop_as_root` class is similar to the `desktop_as_window` class except that it appears on the background window of the display. These classes are defined independently due to inherent differences in the two forms of representation. For example, the desktop in a decorated window, can have a menubar. This is not possible when using the root window. In the vanilla configuration this problem is solved by having a desktop manager icon whose pop-up menu is identical to the menubar in the decorated window desktop. Additionally, no pop-up menu can be defined for the desktop on the root window, as this would conflict with the pop-up menu defined for the Motif Window Manager mwm.

In skeleton form the `desktop_as_root` class looks like this:

```
desktop_as_root dar1 {
// Child objects
group file_types {
...
}
group desktop_objects {       // desktools and file refs
...
}
methods {                     // drop methods for desktop
...
}
// Attributes
initial_poll_rate = 20;       // poll every 20 seconds
...
}
```

The "xdesk" object class

You can define any number of `filemanager_view`, `desktop_as_window`, and `desktop_as_root` objects as you wish within your XDCL description. The last instance of the `xdesk` object determines the default appearance and behavior of the file manager and the desktop manager during the next startup. In this respect the `xdesk` object should be thought of as the key object in the XDCL descriptions of the desktop and file managers.

The following example illustrates how the `xdesk` object is used.

```
// define two filemanager views
filemanager_view fmv1 {
...
}
filemanager_view fmv2 {
...
}
// define root and decorated window desktops
desktop_as_root root_desktop {
...
}
desktop_as_window window_desktop {
...
}
// define the 'xdesk' object
xdesk {
filemanager = "fmv1";              // use 1st filemanager view
rootdesktop = "root_desktop";      // use 2nd root desktop
decorated_desktop = "window_desktop";     // use 3rd window desktop
}
```

When the desktop manager starts, you use the rootDesktop resource of the RootDesktop class of the xdesk client to specify whether the desktop appears in its own window (decorated_desktop) or in the root window (rootdesktop).

It's worth briefly discussing at this point how the `xdesk` object interacts with the `-open` command-line option of the xdfm and xdtell clients.

If you run xdtell as follows:

```
xdtell -manager fm -open /tmp
```

The file manager opens a filemanager view on the `/tmp` folder. The filemanager view object definition used, is as defined by the `xdesk` object. In the previous example, the definition called `fmv1` would be used. However this can be overridden by a further argument to `-open` that specifies the instance name of the class `filemanager_view` to be used. For example:

```
xdtell -manager fm -open /tmp fmv2
```

causes the file manager to use the filemanager view object definition called `fmv2`. This distinction is useful, as it allows you to define specialized filemanager views for specific purposes.

2.4.2 Defining a menu system

Menus appear in two forms within XDCL:
- as pull-down menus from the menubar.
- as pop-up menus on an icon or window background.

The following XDCL classes help you to build menu structures:
- menubar
- popup_menu
- cascadebutton
- menu
- pushbutton
- radiobutton
- checkbutton
- separator

Defining a menubar

Menubars appear at the top of filemanager view windows and in the desktop window.

In outline form, this is the structure of the pull-down menu system for the file manager view delivered in the vanilla system:

```
menubar FMmenubar {
```

```
cascadebutton {
label = "File";
menu {
...
}
}
cascadebutton {
label = "Edit";
menu {
...
}
}
cascadebutton {
label = "View";
menu {
...
}
}
cascadebutton {
label = "Options";
menu {
...
}
}
cascadebutton {
label = "Help";
justify_left = "False";      // Help menu on right
menu {
...
}
}
}
```

The direct children of the menubar object are cascade buttons. These buttons appear on the menubar. When the user clicks on a cascade button, the menu that is the child of the cascade button is displayed. Defining the contents of a menu is described a little further on.

Defining a menu

An XDCL menu object contains the buttons that will be shown when the user clicks on a cascadebutton on the menubar.

The basic type of button is the pushbutton. This is used to select an action. The checkbutton and radiobutton types are special buttons that let the user choose settings for the program. Radiobuttons let the user make a single selection from a number of possible choices.

Checkbuttons let you choose a number of selections from a number of possible choices.

The `Save on exit` menu that can be seen from the `Options` menu of a filemanager view, is an example of a set of radiobuttons collected on a menu.

Tip:

You cannot mix radiobuttons and checkbuttons within a single menu.

The fourth type of button is the cascadebutton. You have already met this type of button in relation to the menubar, but cascadebuttons can also be used within menus to connect in

further menus. When the user focuses on a cascadebutton, the associated menu is displayed.
Separator objects allow you to group together other buttons within the menu.
The following example gives an outline of the `File` menu from the vanilla filemanager view:

```
menu {
cascadebutton {
label = "New";
menu {
pushbutton {
label = "File...";
}
pushbutton {
label = "Folder...";
}
}
}
pushbutton {
label = "Select...";
}
separator {}
pushbutton {
label = "Save";
}
separator {}
pushbutton {
label = "Restart...";
}
pushbutton {
label = "Exit...";
}
}
```

Defining a pop-up menu
The XDCL `popup_menu` class is much the same as the menu class described in the previous section. However, instead of being linked to a cascadebutton parent, as a menu is, a `popup_menu` stands alone, but may be connected to an icon or a window background. Like a menu, a pop-up menu can also contain cascadebuttons that lead to submenus.

A more detailed look at the button classes
The examples given so far only define minimal buttons, where the label attribute is only defined. Of course, such a menu system would not be very useful.
All button classes let you define a `label` attribute and a `mnemonic` attribute. The `label` attribute defines the string that appears in the menu. The `mnemonic` attribute defines which character which, when the corresponding key is pressed selects that menu item. The `mnemonic` character must exist in the button label, and must be unique within the parent menu, menubar or pop-up menu.
All button classes have a method or methods associated with them that are invoked when the user interacts with the button.
The `pushbutton` class simply has a select method that is invoked when the user selects the pushbutton in the menu. The `radiobutton` and `checkbutton` classes can have two states: selected or not selected. When a user clicks on such a button and the button changes from

not selected to selected, the `set` method is called. When the button changes from selected to not selected, the `unset` method is called. The `radiobutton` and `checkbutton` classes also have an `initially_set` attribute that enables you to define the initial state of the button when the client interpreting the XDCL description first starts up.

Tip:

The `set` or `unset` method is also called during creation of the menu system.

The `sensitivity` attribute enables you to decide circumstances under which the button should not be available for selection. Motif represents this by graying-out the button. You can make buttons available according to the number of icons selected in a window. If your select method wants to operate on the selected files in filemanager view, you can use the `{%selected_files}` variable in your method description, which will be expanded to the fully-qualified names of the files currently selected.
All button types have a help method. This is invoked when the user requests help; normally by focusing on the button in a menu and pressing the help key. The vanilla XDCL description defines help methods for all buttons in menus. These methods call up the Help Manager to display appropriate help.

2.4.3 File typing

The `file_type` class connects characteristics of a file with an object representation in XDCL. The `iconic_object` class defines the picture and label, pop-up menu and methods for the object. Iconic objects are discussed in the next section.
When displaying the files in a folder, the file manager compares each file in turn against the file types defined for the filemanager view being used. Match characteristics include name pattern, path pattern, owner, read-write-execute permissions, age, size and so on.
When defining a `file_type` you can include as many or as few match characteristics as you wish. However, for a file to be matched all the characteristics in the `file_type` must match all the characteristics of the file.
For example:

```
file_type document_file {
// match characteristics
name_pattern = "*.doc";
path_pattern = "/home/project/fred";
// iconic object to represent matched files
iconic_object document_object {
...
}
}
```

will match any file that ends in '.doc', but only if the file resides in the folder `/home/project/fred`.
Filemanager views and desktops collect together the file types that define how they should display files in a special `group` object that must have the instance name `file_types`. When the view wants to determine the iconic representation of a file, it compares the file against each `file_type` object in the `file_types` group in turn, starting from the end of the group. The view stops as soon as an exact matching `file_type` is found.
This is best illustrated with an example:

```
group file_types Ï
```

```
file_type A Ã
name_pattern = "*";           // matches any file
iconic_object {
...
}
}
file_type B {
name_pattern = "*.doc";       // matches .doc files
iconic_object {
...
}
}
file_type C {
name_pattern = "*.doc";       // matches .doc files you
writable_by_you = "True";     // have write access to
iconic_object {
...
}
}
}
```

A file called 'file.doc' to which the user had write access would match `file_type C`.
A file called 'file.doc' to which the user did not have write access would match `file_type B`.
A file called 'file.txt' would match `file_type A`.

Tip:

The recommended arrangement of `file_type` objects should be that more general file types come earlier in the group; more specific file types come later. Derivation can be used to define more specific file types from more general ones.
It's a good idea to define a catch-all file type at the start of the group. Any files that fail to match later file types will get caught by the catch-all.
The client interpreting the XDCL description provides a default iconic representation if a file is not matched by any `file_type` object in the group.
Although it is possible for filemanager and desktop view objects to have different sets of file types, you should in general make sure that your main filemanager and desktop view objects share common file types. Otherwise files dragged from a filemanager view onto the desktop will be typed differently. The best way to do this is to derive the `file_types` group in each view from a common definition.

2.4.4 Iconic objects

An iconic object defines how a file or desktool icon looks and behaves.
The `iconic_object` class is made up of:
- an icon definition that defines the picture and label of the object.
- a pop-up menu definition, displayed when the user clicks on the object with <BMenu>.
- a set of methods that define what happens when the user double-clicks on the object, drops another object on this object, or requests help by focusing on the object and pressing the help key.
- special attributes, such as whether or not the iconic object is visible or selectable.

In skeleton form, an XDCL description of an iconic object looks like this:

```
iconic_object an_object {
// Child objects
icon {
...
}
popup_menu {
...
}
methods {
...
}
// Attributes
visibility = "True";          // The user can see the object
selectability = "False";      // but not select it
}
```

The elements of an icon object which affect the drag-and-drop capability are described in the chapter "Dragging and dropping ".

Icons

The `icon` class defines the picture and label of the iconic object. This object in turn has an `image` class child that defines the name of the image files to be used. The `image` class lets you define:

- names for the basic image and mask,
- an alternative image (if you wish) to be displayed when the icon is selected
- and (also if you wish) a smaller alternative image to be used automatically when the client interpreting the XDCL description runs on screen resolutions less than 1024*768 pixels.

The environment variable `XDESK_ICON_PATH` is used to locate the image files specified in the XDCL description. The default path is `/usr/lib/X11/xdesk/ICONS`. The image files must be standard X11 xbm or GIF format files.

In addition to an image, the icon class has two attributes: `label` and `long_label`. These attributes define the label string that is shown under the icon picture. For icons that represent files and folders in the vanilla configuration, these attributes are defined as follows:

```
label = "{%filename}";
long_label = "{%permissions} {%links} {%owner} {%owners_group} \
{%size} {%last_modified_time} {%filename}";
```

The `label` attribute uses the dynamic variable `%filename` so that the name of file is displayed when the filemanager view displays normal icons.

The `long_label` attribute is used when the filemanager view displays the icons in extended form.

You can, of course change these attribute assignments to display information about the file that is most useful to your circumstances. For example:

```
label = "{%owner}:{%filename}";
```

would also display the file's owner in each icon's label.

Methods for iconic objects

Iconic objects have a set of methods associated with them that define the behavior of the object.

The following predefined method names are understood by the file manager and the desktop manager:

`default_action`executed when the user double-clicks on the object. For example, the following method definition invokes the `vi` text editor passing it the fully-qualified pathname of the file which the object represents:

```
default_action = "external_method vi {%pathandfilename}";
```

`dropped_on_A, dropped_on_B`
executed when the user drops another
`dropped_on_C`object on this object.
`help`executed when the user focuses on the iconic object and asks for help; for example:

```
help = "internal_method XDtellHelp -chapter FileTypes -item Regular";
```

In addition to the above methods that have well-known names, you can add further methods with names of your own choosing; for example:

```
compress = "external_method compress -f {%pathandfilename}";
```

Methods that you name yourself are executed when you use `xdtellfm` with the `-exec` option. For example, to execute the `compress` method on the file `usr/project/fred/afile`:

```
xdtell -manager fm -exec compress usr/project/fred/afile
```

The `edit` and `print` desktools in the vanilla configuration are examples of so-called method-based desktools that, when files are dropped on them, instruct the filemanager view to execute the method `edit` or the method `print` on the dropped file. These desktools use `xdtell -manager fm- exec`, as illustrated above.

2.4.5 Desktop objects

Two classes of objects can be placed on the desktop: `desktool` and `file_object`.
The `desktool` class is very simple. It consists of an `iconic_object` child and two attributes. The `iconic_object` child defines all the visual and behavioral characteristics of the desktool. The attributes are as follows:

`geometry`	defines the position of the desktool, if the desktop is displayed on the root window.
`deletabil ity`	defines whether or not the desktool can be deleted from the desktop.

Table 12: Desctool class: attributes

The file object is simply a reference to a file in the filesystem, together with a position of the object if the desktop is displayed on the root window. You can predefine files or folders to appear on the desktop, by adding `file_objects` to the `desktop_objects` group of the `desktop_as_window` or `desktop_as_root` object.
For example, adding the following file object, places an icon that refers to the folder `/usr/bin/X11` on your desktop:

```
file_object Xbin {
filename = "X11";
pathname = "/usr/bin";
}
```

2.5 Internationalizing the desktop

The examples in this chapter use hard-wired strings for text that appears in window titles, menus and icon labels, but it is possible to write XDCL descriptions that are international - that is the XDCL description contains no strings that are specific to a particular language such as English or German.
Internationalization of the XDCL description is possible by using the XDCL class `nls_string`. This class enables you to define all the language-specific strings in a separate file, or files. This class uses the Native Language Support (NLS) mechanism described in the X/Open Portability Guide. This section briefly describes the NLS mechanism, but you should refer to the appropriate documentation for full details.
Language-specific strings are collected together in NLS message files, that look like this:

```
$set 1
$ Messages for the .xdesk.cfg and desktop.cfg files.
1 "this folder"
2 "parent folder"
3 "File"
...
```

Each message line starts with the message number, followed by the string, as it is written in a given language. Messages can be grouped together in sets as defined by lines of the type `$set 1`. The message file can then be translated into different languages, for example, German:

```
$set 1
$ Messages for the .xdesk.cfg and desktop.cfg files.
1 "Aktuelles"
2 "Vater"
3 "Datei"
...
```

Message files are then compiled into NLS message catalogs, using the NLS utility `gencat`. The catalog used at runtime is defined by the setting on the `LANG` environment variable.
The `nls_string` class lets you define where the client interpreting the XDCL description can find the appropriate message when it starts up, by enabling you to set:

`filename`	the name of the message catalog that contains the message.
`set_number`	the number of the set that contains the message.
`msg_number`	the message number.
`default_string`	if the client cannot find the message indicated by the above three attributes, it will use the string you define here.

Table 13: nls_string class

Here is an example instance of an `nls_string` object, and its reference by a menu pushbutton:

```
nls_string nls_file
{
filename = "xdesk";
set_number = 1;
msg_number = 3;
default_string = "File";
}
...
cascadebutton file_menu {
label = nls_file;
...
}
```

2.6 Customizing the desktop configuration

The preceding sections of this chapter introduce the concepts and classes of XDCL. You can customize the desktop configuration in one of two ways.
If you are system administrator you may wish to take the set of XDCL files delivered with the product and modify them to your own requirements, simply replacing the modified at the appropriate location in the system when you make your integrated system available to the end users.
If you are an advanced user and you want to use XDCL to customize your desktop configuration, you are recommended to do this by:

- making local `.xdesk.cfg`, `.xdcm.cfg`, `.xdhm.cfg`, `.xdnm.cfg`,`.xdsm.cfg`, `.xdum.cfg`, and `.xdvm.cfg` files and placing them where they will be found by `XDESK_CFG_PATH`.
- #including the delivered XDCL description file: of the relevant manager:
  `#include "/usr/lib/X11/xdesk/cfg/xdesk.cfg"` for xdesk and xdfm;
  `#include "/usr/lib/X11/xdesk/cfg/xdcm.cfg"` for xdcm;
  `#include "/usr/lib/X11/xdesk/cfg/xdhm.cfg"` for xdhm;
  `#include "/usr/lib/X11/xdesk/cfg/xdnm.cfg"` for xdnm;
  `#include "/usr/lib/X11/xdesk/cfg/xdsm.cfg"` for xdsm;
  `#include "/usr/lib/X11/xdesk/cfg/xdum.cfg"` for xdum;
  `#include "/usr/lib/X11/xdesk/cfg/xdvm.cfg"` for xdvm.
- using derivation to modify the behavior of the vanilla XDCL description.

Tip:

Deriving from the system configuration makes it more likely that your customized XDCL description will work with future upgrades of XDCL.
Tips, hints and examples of customization are included in a README file that is delivered with the product. This file can be found in the directory `/usr/lib/X11/xdesk/cfg`.

3 Dragging and dropping

A standardized drag-and-drop mechanism has been incorporated into OSF/Motif V1.2. This allows uniform support of inter-client communication via user-friendly OSF/Motif-based user interfaces. The drag-and-drop functionality of OSF/Motif ensures the standard appearance and behavior of these interfaces when dragging and dropping.
When you click with the mouse on an element of the user interface, drag it across the screen and then release (drop) it on another element, thereby executing an action, this is referred to as dragging and dropping an object. As a result, data is exchanged between the clients involved. You can use the <MShift> and <MCtrl> keys to choose between moving, copying and linking.
OSF/Motif allows certain user interface elements - labels or images in pixmap format, for example - to be dragged and dropped. In addition, the SINIX/windows User Environment offers extensive drag-and-drop functionality for the object managers xdesk, xdfm, xdhm, xdmm, xdnm, xdum and xdvm, allowing you to drag their objects - files, mailboxes, desktools, etc. - across the screen and drop them on other objects. For example, you can drag a file represented by an icon in the xdfm file manager to the printer icon in a window of the xdesk desktop manager and drop it there, thus causing this file to be printed.
The XDCL description of each object manager defines the drag-and-drop functionality available for it in the standard configuration, i.e.:

- which objects can be dragged
- which objects can have objects dropped on them
- which operations are permitted
- which methods can be used for which operations

When customizing your desktop configuration, you can also customize the drag-and-drop functionality of the object managers.

Prerequisites for dragging and dropping
Objects cannot be successfully dragged and dropped unless the following prerequisites are met:

- The client sending the object must inform the client receiving it what kind of object it is. The client receiving it must accept this kind of object.
- The operation you want to use must be permitted for the dragged object.
- A method must be defined for the operation you want to use, and this method must then be executed for the object.

These prerequisites are explained in more detail during the course of the chapter.

3.1 Objects and targets

When an object is transferred from one client to another, the clients must agree on the kind of object it is. If it is a file, for example, it must be clear whether the client sending it is transferring the filename alone or a full pathname, or whether the data transferred with the file also contains information on it such as the name of the owner or the file type.
Each object can be assigned a number of characteristics known as targets. A target defines the form in which an object is transferred. Several targets can be assigned to each object.

Example:

> The target FILE_NAME is assigned to an object. FILE_NAME is defined as the absolute pathname of a file. The object is also assigned the target FILE_USER, which contains the name of the file owner as well as the pathname.

Every client can define its own targets for objects. However, so that data can be transferred between clients when an object is dragged and dropped, at least one of the dragged object's targets must match one of the targets defined for the element on which the object is dropped, i.e. the client receiving the object must accept at least one of the targets offered.

3.1.1 Predefined targets

Various targets are used in SINIX/windows. Targets that must be recognized by X clients if they are to exchange data when objects are dragged and dropped are defined in the Inter-Client Communication Conventions Manual (ICCCM).

ICCCM targets:	
FILE_NAME	Name of any file
USER	User name
HOST	Hostname
STRING	Text in ISO Latin 1 string format
TEXT	Text in standard string format

Table 14: ICCM targets

The following additional targets are defined in OSF/Motif:	
COMPOUND _TEXT	Text for labels, lists and text boxes in XmString format
PIXMAP	Images in pixmap format

Table 15: Additional targets in OSF/Motif

Targets in SINIX/windows

The following targets are SNI enhancements. They allow dragging and dropping between applications on different hosts in the network and under different login names.

Target	**Contents**	**Meaning**
_SNI_FILE	user@host:absolute_pat hname	Any file
_SNI_FILE_RE G	user@host:absolute_pat hname	Regular file
_SNI_FILE_DIR	user@host:absolute_pat hname	Directory
_SNI_FILE_BLK	user@host:absolute_pat	Block special

	hname	file
_SNI_FILE_CHR	user@host:absolute_pathname	Character special file
_SNI_FILE_FIFO	user@host:absolute_pathname	Fifo file
_SNI_FILE_EXEC	user@host:absolute_pathname	Executable file
_SNI_FILE_Z	user@host:absolute_pathname	Compressed file
_SNI_DESKTOOL	user@host:desktool_object_ID	Desktool
_SNI_TOOLBOX	user@host:toolbox_object_ID	Toolbox
_SNI_MAIL_MSG	user@host:absolute_pathname:number	Message
_SNI_MAIL_FOLDER	user@host:absolute_pathname	Mail folder
_SNI_USER	user@host:group:user	User
_SNI_GROUP	user@host:group	User group
_SNI_HOST	user@host:network:host	Host
_SNI_NETWORK	user@host:network	Network
_SNI_ARCHIVE	user@host:local cmd\0remote cmd	Archived file
_SNI_MIME_BODY_PART	user@host:absolute_pathname	Regular file containing the contents of the MIME body part

Table 16: Additional targets in SINIX/windows

The targets are specified in the following format:

`_SNI_base_extension`

`base` The object type, e.g. FILE for file or HOST for host. `base` is always specified.

`extension`
A specific form of the object type specified by `base`. _SNI_FILE_REG stands for a regular file, for example. `extension` is not always specified.
In XDCL descriptions you can add further extensions to define object types valid only in XDCL descriptions. For example, _SNI_FILE_GIF can be defined for a GIF file.

The contents of the SNI targets have the following format:

`user@host:object_identifier`

`user` Valid user name of the user dragging the object.

`hostname`

Valid hostname of the machine that is host of the object to be dragged.
The hostname must comply with the rules of the transport protocol used.

`object_identifier`

Depends on the object dragged. In the case of a file, for example, `object_identifier` is its absolute pathname.

3.1.2 Export and import target lists

The sending client, i.e. the one making the object available for dragging, exports the list of all possible targets of the dragged object. This is specified in an XDCL description, as follows:

```
export_targets = "_SNI_FILE_REG FILE_NAME STRING TEXT";
```

If there is no export list for an object, although this would be useful, a default is used.
In the XDCL description of the recipient, an import list is included defining all possible targets for objects that can be imported, as follows:

```
import_targets = "_SNI_GROUP _SNI_USER USER";
```

If there is no import list for an object, although this would be useful, a default is used.
When you drop an object, the receiving client requests the object from the sending client in a form determined by the first common target in its import list. An object is always transferred on the basis of the first common target. The order of the targets in the import list is therefore significant.
A target defined in an XDCL description overwrites the default target list for this object. To prevent objects from being dropped indiscriminantly on an object, an empty import list can be defined as follows:

```
import_targets="";
```

For information on the default values of the targets defined in the source code of the clients, refer to `import_targets` and `export_targets` in the descriptions of the object classes in the chapter "XDCL object classes and elements ". You obtain the default values defined in the XDCL descriptions of the object managers from the corresponding XDCL configuration files.

3.2 Senders and recipients for dragging and dropping

The client whose object is dragged is referred to as the sender, and the on which an object is dropped as the recipient. The dragged object is processed by means of a defined method.

3.2.1 Draggable objects

All clients can define objects as draggable.
Labels, lists, text boxes and images are defined in Motif as standard draggable objects.
For each object manager you can define objects of the class `iconic_object` as draggable in the XDCL description.
Example:
The following XDCL description specifies that the described object can be dragged and has the targets _SNI_GROUP, STRING and TEXT.

```
iconic_object {
export_targets = "_SNI_GROUP STRING TEXT";
}
```

An element in the `export_targets` list of the dragged object must match an element in the `import_targets` list of the recipient for the transfer to be successful.
For a successful transfer the sender and recipient must also have at least one operation (move, copy or link) in common (see section "Operations and methods ").

3.2.2 Recipient drop sites

All clients can define drop sites for dragged objects. Drop sites are also assigned targets and operations.
A drop site is valid for a dragged object if at least one of its targets and one of its operations matches the transferred targets of the dragged object.
As standard, labels are defined in Motif as drop sites.
The following drop sites are possible for object managers in SINIX/windows:

- an object of the class `iconic_object`
- the view of an object manager if the dragged object is dropped on a window background, i.e.:
  - `desktop_as_root` or `desktop_as_window` for the desktop manager
  - `archivemanager_view` and `devicemanager_view` for the volume manager
  - `filemanager_view` for the file manager
  - `hostmanager_view` for the host manager
  - `mailmanager_view` and `mimemanager_view` for the and the communication manager
  - `nfsmanager_view` for the nfs manager
  - `pkgmanager_view` for the software manager
  - `usermanager_view` for the user manager

Example:
The following description specifies that objects with the targets _SNI_GROUP, _SNI_USER and USER can be dropped on the user manager view:

```
usermanager_view default {
...
import_targets = "_SNI_GROUP _SNI_USER USER";
}
```

3.3 Operations and methods

Draggable objects are assigned operations. There are three operations: move, copy and link. You select an operation by means of the <MCtrl> and <MShift> keys. The key combination pressed when dropping the object applies. A method must be defined in the recipient's description for the desired operation. This method must then be executed.

Key	**Mouse button**	**Operation**
<MShift>	<BTransfer>	Move
<MCtrl>	<BTransfer>	Copy
<MShift> <MCtrl>	<BTransfer>	Link
	<BTransfer>	Uses a method defined for the sender and recipient in the following order: move, copy, link.

Table 17: Draggable object: operations

In an XDCL description, the operations and methods are specified as follows:
For a draggable object, move, copy and link operations can be permitted or prevented by assigning a value of true or false to the elements `dragabilityA`, `dragabilityB` and `dragabilityC`. For each permitted operation, a method must be defined for `dropped_on_A`, `dropped_on_B` and `dropped_on_C` in the recipient's description.

Key	**Mouse button**	**Operation permitted by the sender**	**Method defined for the recipient**
<MCtrl>	<BTransfer>	dragabilityA (copy)	dropped_on_A
<MShift>	<BTransfer>	dragabilityB (move)	dropped_on_B
<MShift> <MCtrl>	<BTransfer>	dragabilityC (link)	dropped_on_C
	<BTransfer>		uses a defined method in the sequence B, A, C.

Table 18: Draggable object: permit or prevent operations

Example
The following description for a draggable object of the class `group_type` with the name

`group` specifies that the object receives the targets _SNI_GROUP, STRING and TEXT, and permits copy and move operations but not linking.

```
group_type group {
iconic_object icon {
export_targets = "_SNI_GROUP STRING TEXT"
dragabilityA = "True";
dragabilityB = "True";
dragabilityC = "False";
}
```

The following description, for example, allows the object to be dropped on the recipient, in this case the user manager view:

```
usermanager_view default {
import_targets = "_SNI_GROUP _SNI_USER USER";
methods def_view_methods {
dropped_on_A = "internal_method XDcopyDroppedObjects";
dropped_on_B = "internal_method XDmoveDroppedObjects";
}
}
```

Both the sender and recipient descriptions have the target _SNI_GROUP, and methods are defined in the recipient description for the `DragabilityA` and `DragabilityB` operations permitted by the sender.

4 Formal syntax of XDCL

The following is a formal description of the XDCL syntax. The syntax description consists of a set of transformation rules, each with a right-hand side and left-hand side, separated by the symbol ::= .
The left-hand side consists of a single, unique non-terminal symbol. The right-hand side consists of the possible transformations of the left-hand side into terminal symbols. Terminal symbols are signified by ASCII characters in single quotes, for example: ':', or by one of the following:

IDENTIFIER::=	[A-Za-z_][A-Za-z0-9_]*
STRING::=	\"([^"\\\n]|\\["\\\n])*\"
DECIMAL::=	[+-]?[1-9][0-9]*
OCTAL::=	[+-]?[0][0-7]*
HEXADECIMAL::=	[+-]?(0x|0X)[0-9a-fA-F]+
OVERRIDE::=	"override"

Table 19: Formal syntax of XDCL: left-hand side non-terminal symbol

See your SINIX reference manual for a fuller explanation.
The right-hand side of a rule can also contain the following operators:

|	The OR operator. For example, a|b means either a or b.
{}	The repetition operator, meaning zero or more occurrences of the enclosed symbol(s).
[]	The optional operator, meaning zero or one occurrences of the enclosed symbol(s).

Table 20: Formal syntax of XDCL: transformations of the left-hand side into terminal symbols

The following table details the components of XDCL using the above conventions.

configuration_file::=	{ [objects] | [override_objects] }
objects::=	named | named_derived
override_objects::=	override_object | override_object_derived | override_attribute
named::=	class_name instance_name object_body
named_derived::=	class_name instance_name':'reference object_body

override_object::=	OVERRIDE reference object_body
override_object_derived::=	OVERRIDE reference ':' reference object_body
override_attribute::=	OVERRIDE reference object_body
object_body::=	'{' { [attributes] | [objects] } '}'
attributes::=	string_attribute | constant_attribute | object_attribute | reference_attribute
string_attribute::=	IDENTIFIER'='STRING ';'
constant_attribute::=	IDENTIFIER'='number ';'
object_attribute::=	IDENTIFIER'='objects [';']
reference_attribute::=	IDENTIFIER'='reference ';'
reference::=	component{'.'component}
component::=	[class_name'::']instance_name
number::=	DECIMAL | OCTAL | HEXADECIMAL
class_name::=	IDENTIFIER
instance_name::=	IDENTIFIER

Table 21: Components of XDCL

For example, from the above list:
- a configuration file consists of zero or more objects.
- each object has an instance name.
- each object can be derived.
- each object will have a class name followed by an instance name, followed by an object body.

An object body is enclosed within braces "{}", which must be present although they can be empty, in which case the defaults will be used. The object body attributes can be assigned as required. Default values for attributes not explicitly assigned are defined in the class reference.

Tokens

An object description is made up of tokens. There are five types of XDCL token:
- identifiers,
- numbers (also treated as primitive class),
- strings (also treated as primitive class),
- operators,
- separators.

All XDCL tokens must be used in accordance with the XDCL syntax rules. Blanks, tabs and newlines are ignored except where they serve to separate tokens within a language and inside strings. Where the sequence // appears on an input line, all text up to the end of the line is treated as a comment.
Each of the token types used in the XDCL is described below.

4.1 Primitive class allowing for variable substitution

The XDCL token types string and number are treated as primitive classes.
A primitive class can contain variable substitutions. These cause the inclusion of text from a SINIX environment variable or a special string known to the object class whose attribute this string has been assigned to. These variable substitutions are introduced within the body of the string within braces.

Tip:

Variables can only be used inside strings and must be enclosed in braces {}. Environment variables appear in the form that the SINIX shell uses, for example `$HOME}`. **Expansion variables** are preceded by a '%' character, for example, `%filename}`. Shell variables are preceded by a '$' character, for example `{$HOME}`.

A complete list of all variables is given in the chapter "XDCL variables ".

4.2 Identifier

An identifier is a sequence of characters, including the '_' (underscore) character. There is no limit to the length of an identifier. The only rule is that the first character must be a letter or an underscore. All identifiers are case sensitive, which means that an uppercase 'A' is seen as a different character from a lowercase 'a'.

4.3 Number

A number is a numeric value in either decimal, octal or hexadecimal format, as follows:

decimal	A sequence of digits beginning with any digit other than a 0. For example, 1347.
octal	A sequence of digits beginning with a 0 and containing the digits 0 through 7. For example, 02 and 077.
hexade cimal	A sequence of alphanumeric characters beginning with 0x or 0X and containing the digits 0 through 9 and/or the characters A through F. The letters A through F can be either upper or lower case.

Table 22: Formal syntax of XDCL: number

All three of these constants can be signed (prefixed by a plus or minus sign).

4.4 String

A string can be any sequence of letters or digits included between the double quote characters", for example:

```
"This is a string"
```

The double quote character can appear in a string if it is preceded by a '\' character.
A string can extend across multiple lines in a configuration file if each newline is preceded by the escape character '\'. The literal character '\' can be included if it is preceded by a '\' character.

4.5 Operator

The only operator recognized by XDCL is the assignment operator '='.

4.6 Separator

XDCL uses braces {} to delimit the body of object descriptions and variables.

5 XDCL variables

Throughout a configuration specification, variables must be enclosed in braces and can only be used inside a string. For example:

```
select = "internal_method XDFMdeleteFiles {%pathandfilename}";
```

XDCL scripts can also use environment variables such as `$HOME`. Environment variables must be enclosed in braces and start with a '$' character. For example:

```
initial_folder = "{$HOME}";
```

Environment variables are only evaluated on startup. If the environment variable is subsequently changed, the client will only recognize the change when restarted.

5.1 Static variables

The following is a list of static variables that can be used within an XDCL description. These are expanded when the client interpreting the XDCL description is started:

`%hostname`	Is expanded to the name of the host on which the client is running.
`%this_user`	Is expanded to the name of the user running the client that is interpreting the description.
`%this_users_group`	Is expanded to the name of the group of the user running the client that is interpreting the description.

Table 23: Static variables

The following is a list of environment variables especially useful in the context of XDCL files. These are expanded when the client is started. Refer to the manual "SINIX/windows User Environment; Clients Reference Manual" for a full list and full descriptions of the environment variables clients expand when interpreting XDCL descriptions.

`$DISPLAY`	Enables you to specify the display on which the client is to create its windows. If this variable is not set, no default is assigned.
`$G_EDITOR`	The name of the graphical text editor. If this variable is set, it takes precedence over VISUAL and EDITOR when the external method TEXT_EDITOR is used.
`$VISUAL`	The name of the text editor to be used. if G_EDITOR is not set. This environment variable is used by the XDCL external method TEXT_EDITOR.
`$EDITOR`	The name of the text editor to be used if

	the G_EDITOR and VISUAL environment variables are undefined. If EDITOR is undefined, the `vi` text editor is used. This environment variable is used by the XDCL external method TEXT_EDITOR.
`$XDESK_CFG_PATH`	Specifies a list of directories which will be searched for the XDCL configuration. This is `$HOME` by default.
`$XDESK_SAVE_DIR`	Specifies the directory to be used to store the object managers' save files on save or exit. This is set to `$HOME` by default.
`$XDESK_ICON_PATH`	Specifies a list of directories that will be searched for images.

Table 24: Environment variables

5.2 Dynamic variables

The following is a list of the dynamic variables used by the clients interpreting XDCL descriptions. Dynamic variables fall into the following categories:

- general variables and
- variables for the respective object manager

Note that the volume manager consists of the archive manager, the device manager, and the software manager.
All dynamic variables are expanded according to runtime context by the controlling Object Manager.

5.2.1 General variables

`%dropped_files`	List of the absolute pathnames of dropped objects. Elements in the list are separated by space characters.
`%drop_host`	Hostname of the machine running the application that initiated the drop.
`%drop_user`	Name of the user who started the application dropping the object.
`%objectname`	Full name of the current object. The format of the name depends on the object type. For normal file objects, `%objectname` is the absolute pathname of the file. For desktools and toolboxes, %objectname is the XDCL class name of the appropriate object (`desktool` or `toolbox`).
%selected_object _count	Is expanded to the number of selected objects in the current window.
`%selected_objects`	List of object names selected in the current window when a method is invoked. In the case of objects representing files, the object name is taken to be the absolute pathname of the file it represents.
%total_object_co unt	Is expanded to the total number of objects in the window.
`%viewed_folder`	Absolute pathname of the **container** currently viewed by the object manager.

Table 25: Dynamic variables: general

5.2.2 Variables of the archive manager

The archive manager is part of the volume manger.
The following variables may only be assigned to methods associated with instances of the`iconic_object` class, associated with instances of the `archive_entry_type` class:

`%name`	Is expanded to the name of the archive entry.
`%pathname`	Is expanded to the full path of the archive entry matched. The trailing "/" character is removed, however.
`%pathandfilename`	Is expanded to the full path and entry name, separated by a "/" character.
`%permissions`	Is expanded to the permissions component of the output of the `ls -l` command.
`%owner`	Is expanded to the name of the owner of the entry.
`%owner_group`	Is expanded to the name of the owner of the entry's group.
`%size`	Is expanded to the size of the entry in bytes.
`%time`	Is expanded to the last time the entry was modified. The format is defined by `ctime` (XPG3), the trailing newline characters are removed, however.

Table 26: Variables of the archive manager

5.2.3 Variables of the communication manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `message_type` class.

`%date`	Is expanded to an NLS string containing the date on which the message was sent. It takes the form: *day month day_of_the_month* time year
`%folders_dir`	Is expanded to the absolute pathname of the folders directory.
`%mail_items`	NLS string containing the number of message objects that exist in the current folder of the mail manager view.
`%mailbox`	Is expanded to the filename of the

	mailbox on the host in which the mail service leaves incoming mail for the user.	
`%mailbox_dir`	Is expanded to the absolute pathname of the directory containing the `%mailbox`.	
`%message`	Is expanded to the absolute pathname of a file containing the message.	
`%message_body`	Is expanded to the absolute pathname of a file containing the body of a message.	
`%message_header`	Is expanded to the absolute pathname of a file containing the header of a message.	
`%parent_dir`	Is expanded to the absolute pathname of the parent of the folders directory.	
`%sender`	Is expanded to the name of the sender of the message, or the sender's address if no name is specified.	
`%sender_id`	Is expanded to the local user name of the sender of the message without host and domain information. This is used mainly to establish which image is assigned to the name.	
`%size`	Is expanded to the size of the message in lines.	
`%viewed_folder`	Is expanded to the mail folder name of the viewed mail message in the Communication Manager.	
`%status`	Is expanded to a code indicating the status of the message. The following codes are possible:	
	N	new unread message
	O	old unread message
	R	read
	U	urgent or important message
	D	message marked for deletion. It is deleted from the current folder at the next update or when the view is closed.
`%subject`	Is expanded to the contents of the message's "Subject" field.	
`%trash`	Is expanded to the absolute pathname of the mailbox in which messages consigned to trash are stored.	
`%trashdir`	Is expanded to the absolute pathname of the folder in which messages and	

	mailboxes consigned to trash are stored.

Table 27: Variables of the communication manager

5.2.4 Variables of the device manager

The device manger is part of the volume manager.
The following variables can only be assigned to methods associated with instances of the `iconic_object` class associated with instances of the `device_type` class.

`%bdevice`	Is expanded to the block device name as given in the file `/etc/device.tab.`
`%capacity`	Is expanded to the capacity of the device as given in the file `/etc/device.tab.`
`%cdevice`	Is expanded to the character device name as given in the file `/etc/device.tab.`
`%default_capacity`	Is expanded to the default capacity of the device as given in the file `/etc/device.tab.`
`%desc`	Is expanded to the description of the device as given in the file `/etc/device.tab.`
`%hwtype`	Is expanded to the hardware type as given in the file `/etc/device.tab.`
`%mode_indicator`	Is expanded to "[root]" if the manager is in system administrator mode or "" if the manager is not in system administrator mode.
`%mount_point`	Is expanded to the mount point of the file system on the device if the device is a file system device.
`%name`	Is expanded to the alias-name of the device as given in the file `/etc/device.tab.`

Table 28: Variables of the device manager

5.2.5 Variables of the file manager

The following variables may only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `file_type` class.

`%associated_progr am`	Is expanded to the assigned value of the element `associated_`program of the `iconic_object`. Typically, this variable is used in conjunction with the interactive file type configuration tool supplied

	with the product. This tool enables the user to assign an associated program name to a file type. The methods for the object can then use `%associated_program` to access this name.
`%filename`	Is expanded to the name of the file matched.
`%last_accessed_time`	Is expanded to the last time the file was accessed, in a format defined in the *X/Open Portability Guide Volume 2* (see the `ctime()` command), but with the trailing newline character removed.
`%last_modified_time`	Is expanded to the last time the file was modified, in a format defined in the *X/Open Portability Guide Volume 2* (see the `ctime()` command), but with the trailing newline character removed.
`%links`	Is expanded to the number of links to the file.
`%owner`	Is expanded to the name of the file owner.
`%owners_group`	Is expanded to the name of the file owner's group.
`%pathname`	Is expanded to the full path of the file matched, with no trailing '/' character.
`%pathandfilename`	Is expanded to the full path and filename, separated by a '/' character.
`%permissions`	Is expanded to the permissions component of `ls -l`.
`%selected_file_size`	Is expanded to the size of the selected files in bytes.
`%size`	Is expanded to the size of the file in bytes.
%total_file_size	Is expanded to the total size of all files in the window. The size is shown in bytes.

Table 29: Variables of the file manager

5.2.6 Variables of the host manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `host_type` class.

`%comment`	Is expanded to the comment for the host in the file `/etc/hosts`.
`%host_address`	Is expanded to the internet address of the host.
`%host_aliases`	Is expanded to the list of the host's aliases, separated by space characters.
`%host_name`	Is expanded to the hostname.
`%mode_indicator`	Is expanded to "[root]" if the manager is in system administrator mode or "" if the manager is not in system administrator mode.
`%reachable`	Is expanded to R for reachable hosts and to N for unreachable hosts.
`%reachable_label`	Is expanded to ? if the host is unreachable and to ' ' if the host is reachable.
`%reachable_object_count`	Is expanded to the number of reachable hosts in the browser window

Table 30: Variables of the host manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class , associated with instances of the `network_type` class.

`%comment`	Is expanded to the comment for the network in the file `/etc/networks`.
`%network_address`	Is expanded to the network address.
`%network_aliases`	Is expanded to the list of the network's aliases, separated by space characters.
`%network_name`	Is expanded to the name of the network.

Table 31: Variables of the host manager

5.2.7 Variables of the log manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `log_type` class.

`%LogCount`	Running number of log message
`%LogDate`	Date of log message
`%LogEntry`	up to 60 characters of log message text

`%LogModulName`	Application name	
`%LogNumber`	Error number	
`%LogTime`	Time of log message	
`%LogType`	Level of log message:	
	ALERT	alert
	CRIT	critical
	EMERG	emergency
	ERR	error
	WARNING	warning
	NOTICE	notice
	INFO	information
	DEBUG	debug
	SYSLOG	syslogd
	TLOG20	TLOG20 (old Targon logging system)
	UNKNOWN	all other levels

Table 32: Variables of the log manager

5.2.8 Variables of the MIME manager

These variables are only expanded within the context of an `iconic_object` belonging to a specific MIME body part object in a MIME Manager View:
Note that the MIME manager is part of the communication manager.

`%compose_cmd`	Is expanded to the configured "compose_command" of a given `MIME_body_part_type` object.
`%content_transfer_encoding`	Is expanded to the MIME content transfer encoding definition of the corresponding MIME body part.
`%edit_cmd`	Is expanded to the configured "edit_command" of a given `MIME_body_part_type` object.
`%MIME_body_part`	Is expanded to the full UNIX

	pathname of the temporary file used by a MIME body part object to store the data of the MIME body part. This has only a meaning for body parts that are no container objects.
`%MIME_body_part_data`	Is expanded to the full UNIX pathname of temporary file that contains the data of a MIME body part object. This file is created in the moment the expansion is done. This variable has only a meaning for body parts that are no container objects.
`%MIME_message_label`	Is expanded to the MIME message label string of the corresponding MIME body part.
`%MIME_subtype`	Is expanded to the MIME subtype string of the corresponding MIME body part.
`%MIME_type`	Is expanded to the MIME type string of the corresponding MIME body part.
`%print_cmd`	Is expanded to the configured "print_command" of a given `MIME_body_part_type` object.
`%view_cmd`	Is expanded to the configured "view_command" of a given `MIME_body_part_type` object.

Table 33: Variables of the MIME manager

5.2.9 Variables of the NFS manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `mount_type` class.

`%mode_indicator`	Is expanded to "[root]" if the manager is in system administrator mode or "" if the manager is not in system administrator mode.
`%mount_automnt`	Is expanded to the auto mount option in the file `/etc/mnttab`, which is alway 'yes'.
`%mount_fstype`	Is expanded to the file system type, which is always 'nfs'.

`%mount_options`	Is a comma-separated list of mount options as required by the `mount(1M)` command.
`%mount_point`	Is expanded to the mount point.
`%mount_special`	Is expanded to the 'special device' for the mount operation, consisting of *server:pathname.*

Table 34: Variables of the NFS manager

The following variables can only be assigned to methods associated with instances of the`iconic_object` class, associated with instances of the `share_type` class.

`%share_descriptio n`	Is expanded to the comment as recorded in the file `/etc/dfs/dfstab`.
`%share_fstype`	Is expanded to the file system type, which is always 'nfs'.
`%share_options`	Is a comma-separated list of share options as required by the share(1M) command.
`%share_pathname`	Is expanded to the pathname.

Table 35: Variables of the NFS manager

5.2.10 Variables of the software manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `pkg_type` class. The expansions are defined in the pkginfo file of the package.

`%category`	Is expanded to the CATEGORY of the software package.
`%description`	Is expanded to the DESCRIPTION of the software package.
`%load`	Is expanded to the LOAD of the software package.
`%pkgname`	Is expanded to the PKGINST of the software package.
`%mode_indicator`	Is expanded to "[root]" if the manager is in system administrator mode or "" if the manager is not in system administrator mode.
`%version`	Is expanded to the VERSION of the software package.

Table 36: Variables of the software manager

The following variables can only be assigned to methods associated with instances of the`iconic_object` class, associated with instances of the `prod_type` class. The expansions are defined in the prodtab file of the CD-ROM

`%location`	Is expanded to the actual software location which is displayed in the window title.
`%prodarch`	Is expanded to the product architecture in the prodtab file entry.
`%proddescription`	Is expanded to the product description in the proddes file.
`%prodlicense`	Is expanded to the license type in the prodtab file entry.
`%prodload`	Is expanded to the product load in the prodtab file entry.
`%prodmachine`	Is expanded to the set of valid machines in the prodtab file entry.
`%prodname`	Is expanded to the product name in the prodtab file entry.
`%prodproc`	Is expanded to the processor type in the prodtab file entry.
`%prodstatus`	Is expanded to the release state in the prodtab file entry.
`%prodversion`	Is expanded to the product version in the prodtab file entry.

Table 37: Variables of the software manager

5.2.11 Variables of the user manager

The following variables can only be assigned to methods associated with instances of the`iconic_object` class, associated with instances in descriptions of the `user_type` class.

`%comment`	Is expanded to the comment in the entry of the user in the file `/etc/passwd`.
`%group_id`	Is expanded to the group ID of the primary user group of the user.
`%group_name`	Is expanded to the name of the primary user group of the user.
`%gtype`	Is expanded to the group type of the user group, which is `global` (i.e. NIS) or `local.`
`%home_directory`	Is expanded to the user's HOME directory.
`%login_shell`	Is expanded to the user's login shell.

`%mode_indicator`	Is expanded to "[root]" if the manager is in system administrator mode or "" if the manager is not in system administrator mode.
`%nis_indicator`	Is expanded to "master" if the system is a NIS master or to " " if the system is not a NIS master.
`%user_id`	Is expanded to the user ID of the user.
`%user_name`	Is expanded to the name of the user.
`%utype`	Is expanded to the user type of the user , which is `global` (i.e. NIS) or `local`.

Table 38: Variables of the user manager

The following variables can only be assigned to methods associated with instances of the `iconic_object` class, associated with instances of the `group_type` class.

`%group_id`	Is expanded to the group ID of the user group.
`%group_name`	Is expanded to the name of the user group.
`%users_in_group`	Is expanded to a list of users, separated by commas, who belong to the group.

Table 39: Variables of the user manager

6 XDCL internal methods

The following is a list of XDCL internal methods and a brief description of their function. Some internal methods take arguments and have restrictions on their use.
All the internal functions are listed below in alphabetical order. The uppercase letters at the beginning of the method name indicate which object managers the method applies to.

- Methods that begin with "XD..." can be used by all object managers providing no special restrictions are specified.
- Methods that begin with "XDAM..." apply to to the archive manager. Note that the volume manager consists of the archive manager and the device manager.
- Methods that begin with "XDDM..." apply to the desktop manager.
- Methods that begin with "XDDV..." apply to the device manager. Note that the volume manager consists of the archive manager, the device manager, and the software manager.
- Methods that begin with "XDFM..." apply to the file manager.
- Methods that begin with "XDHM..." apply to the host manager.
- Methods that begin with "XDLM..." apply to the log manager.
- Methods that begin with "XDMIME..." apply to the MIME manager.
- Methods that begin with "XDMM..." apply to the communication manager.
- Methods that begin with "XDNM..." apply to the NFS manager.
- Methods that begin with "XDSM..." apply to the software manager.
- Methods that begin with "XDUM..." apply to the user manager.

Methods can be invoked from instances of classes that describe the view of an object manager or from instances of the class `iconic_object`. The following list indicates which object classes are available for the object manager views:

Object manager	**Methods**	**View**
Desktop manager	XDDM...	`desktop_as_window` and `desktop_as_root`
Communication manager	XDMIME...	`mimemanager_view`
	XDMM...	`mailmanager_view`
File manager	XDFM...	`filemanager_view`
Host manager	XDHM...	`hostmanager_view`
NFS manager	XDNM...	`nfsmanager_view`
Software manager	XDSM...	`pkgmanager_view`
User manager	XDUM...	`usermanager_view`
Volume manager	XDAM...	`archivemanager_view`
	XDDV...	`devicemanager_view`

Table 40: Internal methods

An object manager's methods can be invoked from only one of its views. This restriction applies to all methods and is therefore not mentioned separately for each one. Where further restrictions apply to particular methods, they are described in each case.

Warning:

You should note carefully the restrictions under which these functions must be used. Failure to observe these restrictions may lead to runtime errors during execution of the internal function.

6.1 XDAMchangeTarget

Arguments
None

Purpose
Shows a target selection dialog box and lists the contents of the archive continued on the target device.

Restrictions
None

6.2 XDAMlistArchive

Arguments
    None
Purpose
    Displays the contents of the current archive in a new archive manager view.
Restrictions
    None

6.3 XDAMreadArchive

Arguments

None

Purpose

Displays the 'Read archive' window for reading in from the current archive.

Restrictions

None

6.4 XDAMstopListArchive

Arguments
    None
Purpose
    Stops listing the contents of the current archive.
Restrictions
    None

6.5 XDAMwriteArchive

Arguments

None

Purpose

Displays the 'Write archive' window for writing an archive to the current target device.

Restrictions

None

6.6 XDDMmodifyDesktool

Arguments
    None
Purpose
    Shows a desktool configuration dialog box and modifies the selected desktool.
Restrictions
    Only one desktool can be selected.

6.7 XDDMnewDesktool

Arguments
    None
Purpose
    Shows a desktool configuration dialog box.
Restrictions
    None

6.8 XDDMnewToolbox

Arguments

None

Purpose

Requests input of a label and then creates a new toolbox with this label.

Restrictions

None

6.9 XDDMopenDesktoolsCatalog

Arguments

None

Purpose

Opens a `desktools_catalog`, i.e. a catalog containing all desktools available in the current instance of the class `desktop_as_window` or `desktop_as_root`.

Restrictions

None

6.10 XDDMopenToolbox

Arguments

None

Purpose

Opens a toolbox.

Restrictions

None

6.11 XDDMreorganizeIcons

Arguments

None

Purpose

Reorganizes the icons on the desktop according to the order in which the corresponding objects are defined in the XDCL description. Desktools or files which are dropped on the desktop later on are appended to the end of the list. For the class `desktop_as_root`, reorganizing of icons also means to align the icons in columns on the right-hand side of the screen.

Restrictions

None

6.12 XDDMversionHelpShow

Arguments
None
Purpose
Shows the desktop manager version help dialog box.
Restrictions
None

6.13 XDDVduplicate

Arguments

None

Purpose

Allows you to identically copy the contents of the current volume to another medium of exactly the same capacity. Displays a dialog box informing you of the current state of the operation.

Restrictions

None

6.14 XDDVerase

Arguments

None

Purpose

Allows you to erase the contents of the current volume. Displays a dialog box informing you of the current state of the operation.

Restrictions

None

6.15 XDDVformat

Arguments

    None

Purpose

    Allows you to format the current volume physically. Displays a dialog box informing you of the current state of the operation.

Restrictions

    None

6.16 XDDVinformation

Arguments

None

Purpose

Displays a dialog box informing you about the current device.

Restrictions

None

6.17 XDDVlistArchive

Arguments
    None
Purpose
    Displays the archive on the current volume.
Restrictions
    None

6.18 XDDVmkfs

Arguments

None

Purpose

Allows you to create a file system on the current volume. Displays a dialog box informing you of the current state of the operation. The file system type is determined by the device's entry in the file
`/etc/device.tab.`

Restrictions

None

6.19 XDDVmount

Arguments

option[,*option*]...

Purpose

Allows you to mount a device with the options *option* as documented for the mount(1M) command.

Restrictions

None

6.20 XDDVreadArchive

Arguments
    None
Purpose
    Allows you to read all files from the current archive. Displays the 'Read archive' window for reading in from the current archive.
Restrictions
    None

6.21 XDDVuseWithArch

Arguments
    None
Purpose
    Allows you to change the nature of the current device from a device containing a file system to an archive device.
Restrictions
    None

6.22 XDDVuseWithFS

Arguments

None

Purpose

Allows you to change the nature of the current device from an archive device to a device containing a file system.

Restrictions

None

6.23 XDDVwriteArchive

Arguments
    None
Purpose
    Displays the 'Write archive' window for writing an archive to the current target device.
Restrictions
    None

6.24 XDFMchangeDir

Arguments

None

Purpose

Displays the contents of the current folder object in a file manager window. If the file manager view is configured such that `new_window_on_navigate` is set true, then a new file manager window is opened, otherwise the current file manager window is used.

Restrictions

Must only be called from an iconic object XDCL description of a directory file type, either from a pop-up menu or through a direct method, such as `default_action`.

6.25 XDFMdeleteFiles

Arguments

List of fully qualified filenames separated by spaces.

Purpose

Deletes the list of files passed as arguments. Displays a Motif question box asking if you want to then delete each file in the argument list. In the case of failure an appropriate Motif error box is displayed indicating the cause of the failure. Only files to which you have write access may be deleted.

Restrictions

Must only be called from an iconic object XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.26 XDFMduplicateFileOrFolder

Arguments

None

Purpose

Shows a Motif prompt box that asks you to supply a name to use when duplicating the current file. If you supply a valid name, the current file or directory is duplicated under the supplied name.

Restrictions

Must only be called from an iconic object XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.27 XDFMfileInformation

Arguments
: None

Purpose
: Shows the information dialog box for the current file.

Restrictions
: Must only be called from an iconic object XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.28 XDFMmodifyFileType

Arguments

  None

Purpose

  Shows a file type configuration dialog box.

Restrictions

  None

6.29 XDFMmoveDroppedObjects

Arguments
    None
Purpose
    Moves dropped objects either to the container represented by the object on the drop site or, if the object is dropped on the background, to the container represented by the object manager's background.
Restrictions
    None

6.30 XDFMnewFile

Arguments
: None

Purpose
: Shows a Motif prompt box asking you for a file name. If you click on `OK`, a new file with the supplied name is created in the current folder. If creation fails, a suitable Motif error box is displayed.

Restrictions
: None

6.31 XDFMnewFileType

Arguments
    None
Purpose
    Shows the new file type dialog box.
Restrictions
    None

6.32 XDFMnewFolder

Arguments

    None

Purpose

    Shows a Motif prompt box asking you for the folder name. If you click on `OK`, a new folder with the supplied name is created in the current folder. If creation fails, a suitable Motif error box is displayed

Restrictions

    None

6.33 XDFMrenameFileOrFolder

Arguments

None

Purpose

Shows a Motif prompt box asking you for a new name for the current file or folder object. If you click on `OK`, the current file or folder object is renamed using the supplied name.

Restrictions

Must only be called from an iconic object XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.34 XDFMnextIconPathElement

Arguments

None

Purpose

Opens a bitmap browser for the next folder in the file manager's icon path, which is defined by means of the iconPath resource or the XDESK_ICON_PATH environment variable. If the bitmap browser is not currently showing a folder of the icon path, the first folder is opened.

Restrictions

None

6.35 XDFMpreviousPathElement

Arguments
	None
Purpose
	Opens a bitmap browser for the previous folder in the file manager's icon path. If the bitmap browser is not currently showing a folder of the icon path, the last folder is opened.
Restrictions
	None

6.36 XDFMselectedFilePermissions

Arguments

None

Purpose

Displays a dialog box for each selected file in the current file manager window in turn, allowing you to change access permissions for the file. If you click on `OK`, the file is set to the access permissions chosen. A Motif error box is displayed if the access permissions cannot be changed. If you click on `All`, all the remaining files selected have their access permissions set to the access permissions chosen.

Restrictions

None

6.37 XDFMsingleFilePermissions

Arguments

None

Purpose

Displays a dialog box for the current file in the current file manager window allowing you to change the access permissions of the file. If you click on `OK`, the file is set to the access permissions chosen. A Motif error box is displayed if the access permissions cannot be changed.

Restrictions

Must only be called from an iconic object XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.38 XDFMsortByAccessTime

Arguments
    None
Purpose
    Sets an internal variable in the file manager such that files are sorted and displayed according to access time.
Restrictions
    None

6.39 XDFMsortByExtension

Arguments

None

Purpose

Sets an internal variable in the file manager such that the files are sorted and displayed according to filename extension.

Restrictions

None

6.40 XDFMsortByModificationTime

Arguments
    None
Purpose
    Sets an internal variable in the file manager such that the files are sorted and displayed according to modification time.
Restrictions
    None

6.41 XDFMsortByName

Arguments

None

Purpose

Sets an internal variable within the file manager such that the files are sorted and displayed according to file name.

Restrictions

None

6.42 XDFMsortBySize

Arguments

None

Purpose

Sets an internal variable within the file manager such that the files are sorted and displayed according to file size.

Restrictions

None

6.43 XDFMversionHelpShow

Arguments

None

Purpose

Shows the file manager version help dialog box.

Restrictions

None

6.44 XDHMattributes

Arguments
: None

Purpose
: Used in an instance of the class `host_type`, the method displays a dialog box in which you can enter host attributes. Used in an instance of the class `network_type`, the method displays a dialog box in which you can enter network attributes. Attributes can only be changed in system administrator mode on a NIS master or a host not included in a NIS domain.

Restrictions
: Must only be called from within a `host_type` or `network_type` XDCL description.

6.45 XDHMChooseDistribution

Arguments
  1 (meaning: Immediately)
  2 (default; meaning: Ask immediately)
  3 (meaning: Ask before exit)
Purpose
  Defines the distribution mode for the NIS database changes.
Restrictions
  None

6.46 XDHMemptytrash

Arguments
    None
Purpose
    Deletes hosts and networks from the trash.
Restrictions
    None

6.47 XDHMexit

Arguments
    None
Purpose
    Exits the host manager and handles the distribution of NIS database changes.
    Displays a Motif question box asking you if you want to exit the object manager.
Restrictions
    None

6.48 XDHMnewHost

Arguments
    None
Purpose
    If the host manager is running in system administrator mode on a NIS master server or a host not included in a NIS domain, the method displays a dialog box in which you can enter the new host. In other cases all input fields are set insensitive.
Restrictions
    None

6.49 XDHMnewNetwork

Arguments

None

Purpose

If the host manager is running in system administrator mode on a NIS master server or a host not included in a NIS domain, the method displays a dialog box in which you can enter the new network. In other cases all input fields are set insensitive.

Restrictions

None

6.50 XDHMnisAdministration

Arguments

None

Purpose

Opens the NIS administration box. If the box is already open, it will be popped to the top of the window stack.

Restrictions

None

6.51 XDHMnisDistribute

Arguments
    None
Purpose
    Distributes the NIS database to the NIS servers.
Restrictions
    Must only be called if the system is running in system administrator mode, is master of a NIS domain and the NIS is activated.

6.52 XDHMnisFetch

Arguments
    None
Purpose
    Fetches the NIS database from the NIS master server.
Restrictions
    Must only be called if the system is running in system administrator mode, is server of a NIS domain and the NIS is activated.

6.53 XDHMnisMap

Arguments
    None
Purpose
    Builds system files out of the actual NIS database.
Restrictions
    Must only be called if the system is running in system administrator mode, and the NIS is activated.

6.54 XDHMShowAll

Arguments

None

Purpose

Switches the mode of the object manager to show all hosts, whether they are reachable or not.

Restrictions

None

6.55 XDHMShowOnlyReachable

Arguments

None

Purpose

Switches the mode of the object manager to show only reachable hosts.

Restrictions

None

6.56 XDHMsortByAddress

Arguments
: None

Purpose
: Sets an internal variable of the host manager in such a way that hosts and networks are then sorted by their Internet addresses.

Restrictions
: None

6.57 XDHMversionHelpShow

Arguments

None

Purpose

Displays a dialog box with information on the host manager version.

Restrictions

None

6.58 XDLMattributes

Arguments
    None
Purpose
    Shows a log message box.
Restrictions
    None

6.59 XDLMsortByDate

Arguments

  None

Purpose

  Sorts log messages by date.

Restrictions

  None

6.60 XDLMsortByLevel

Arguments

None

Purpose

Sorts log messages by level.

Restrictions

None

6.61 XDLMsortByApplication

Arguments
    None
Purpose
    Sorts log messages by application.
Restrictions
    None

6.62 XDLMselect

Arguments
    None
Purpose
    Shows a log selection box.
Restrictions
    None

6.63 XDMIMEapplyCompose

Arguments

None

Purpose

If the MIME Manager View is in the compose mode this function sends the currently composed message and resets the view if sending was successful.

Restrictions

None

6.64 XDMIMEchangeContainer

Arguments

None

Purpose

Displays the contents of the MIME container object that is in the current context. The contents is displayed in the MultiMediaPanel of the MIME Manager View.

Restrictions

Can be called only in the context of an associated `MIME_body_part_type` object that corresponds to a MIME container.

6.65 XDMIMEcomposeOK

Arguments

None

Purpose

If the MIME Manager View is in the compose mode, the composed message is send and if this was successful the view is closed.

Restrictions

None

6.66 XDMIMEcreateDialog

Arguments

    None

Purpose

    Pops up the "Create Body Part" dialog box.

Restrictions

    Can be called only in the context of an associated `MIME_body_part_type` object.

6.67 XDMIMEdeiconifyObject

Arguments

    None

Purpose

    Deiconifies a `MIME_body_part_type` object if this object supports this action. (Only objects of MIME type "text/plain" and some image objects support this.)

Restrictions

    Can be called only in the context of a `MIME_body_part_type` object.

6.68 XDMIMEExportBodyPart

Arguments

None

Purpose

A function to export a `MIME_body_part_type` to a file.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

6.69 XDMIMEexternalCompose

Arguments

Command string of the external compose command.

Purpose

Starts the external compose command.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object that is not a container object.

6.70 XDMIMEexternalEdit

Arguments

Command string of the external edit command.

Purpose

Starts the external edit command.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object that is not a container object.

6.71 XDMIMEmaybeDeiconified

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object can be deiconified.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

6.72 XDMIMEmodifyDialog

Arguments
    None
Purpose
    Pops up the "Modify Body Part" dialogue.
Restrictions
    Can be called only in the context of an associated `MIME_body_part_type` object.

6.73 XDMIMEnextMessage

Arguments

None

Purpose

If the MIME Manager View is in the view mode this function displays the mail message that is the next message to view in the list of messages given to the MIME Manager View at creation time.

Restrictions

None

6.74 XDMIMEpreviousMessage

Arguments

None

Purpose

If the MIME Manager View is in the view mode this function displays the mail message that was previously viewed in this view (if there is any previous message). This is only possible if the MIME Manager View was created with a list of viewable messages passed to it.

Restrictions

None

6.75 XDMIMEreset

Arguments

None

Purpose

Make a reset for the MIME Manager View. If the MIME Manager View is in the view mode this means to display the contents of the MIME container at the root of the MIME tree. If the MIME Manager View is in the compose mode this means to destroy the currently composed message and reset the view to the initial state.

Restrictions

None

6.76 XDMIMEshowInformation

Arguments

  None

Purpose

  Pops the "Bodypart Information" dialogue for the `MIME_body_part_type` object in the current context.

Restrictions

  Can be called only in the context of an associated `MIME_body_part_type` object.

6.77 XDMIMETextWordWrap

Arguments
: String with the value {"ON","on","OFF","off"}

Purpose
: Sets the TextWordWrapping on or off, if the `MIME_body_part_type` object is of type TEXT

Restrictions
: Can be called only in the context of a `MIME_body_part_type` object

6.78 XDMMaliases

Arguments

None

Purpose

Displays a dialog box in which you can read system-specific or change personal aliases, providing you have the requisite access permissions.

Restrictions

None

6.79 XDMMbellVolume

Arguments
    None
Purpose
    Displays a dialog box in which you can move a slider along a scale to adjust the relative volume of the audible signal that indicates the arrival of new mail when the window is iconified. This applies only when the window is a view of the system mailbox.
Restrictions
    None

6.80 XDMMcopySelectedMessages

Arguments
: None

Purpose
: Displays a dialog box in which you can enter the name of a mailbox in which the selected messages are to be stored. If the mailbox does not exist, it is created.

Restrictions
: None

6.81 XDMMexportFolder

Arguments

None

Purpose

Requests you to enter a filename, and saves the contents of the mailbox to the specified file or folder.

Restrictions

Must only be called from the pop-up menu or as one of the defined methods of a mailbox.

6.82 XDMMexportMessage

Arguments
  None
Purpose
  Requests you to enter a filename, and saves the contents of the message to the specified file.
Restrictions
  Must only be called from the pop-up menu or as a defined method of a message.

6.83 XDMMforwarding

Arguments

None

Purpose

Displays a dialog box informing you of the current forwarding status and allowing you to change the current values for forwarding.

Restrictions

None

6.84 XDMMforwardMessage

Arguments
    None
Purpose
    Allows you to forward a message to another user. A dialog box appears asking if you want to change the message before forwarding it. A window then opens by means of which you can forward the message.
Restrictions
    Must only be called from the pop-up menu or as a defined method of a message.

6.85 XDMMgroupReply

Arguments

None

Purpose

Allows you to send a group reply to the current message. You are asked whether the current contents of the message are to be included in the text of the reply.

Restrictions

Must only be called from the pop-up menu or as a defined method of a message.

6.86 XDMMmessageInformation

Arguments
    None
Purpose
    Creates a dialog box with information on the context of the message.
Restrictions
    Must only be called from the pop-up menu or as a defined method of a message.

6.87 XDMMmodifyMessageType

Arguments
  None
Purpose
  Creates a dialog box allowing you to modify all messages of the selected message type.
Restrictions
  None

6.88 XDMMnewFolder

Arguments
    None
Purpose
    Requests input of the name of a mailbox to be created in your folder for mailboxes.
Restrictions
    None

6.89 XDMMrenameFolder

Arguments

None

Purpose

Requests input of a new name for the mailbox in your folder for mailboxes. The system mailbox and trash mailbox cannot be renamed.

Restrictions

Must only be called from the pop-up menu or as a defined method of a message.

6.90 XDMMreplyAddress

Arguments

None

Purpose

Requests input of a new reply address which then appears automatically in the "Reply to:" field when a message is sent, forwarded or replied to.

Restrictions

None

6.91 XDMMreplyToMessage

Arguments

None

Purpose

Allows you to send a reply to the current message. You are asked whether the current contents of the message are to be included in the text of the reply.

Restrictions

Must only be called from the pop-up menu or as a defined method of a message.

6.92 XDMMsendMessage

Arguments

None

Purpose

Allows you to create a text message to send to another user. A window appears for the purpose.

Restrictions

None

6.93 XDMMsetFoldersDirectory

Arguments

None

Purpose

Allows you to change the name of the folder in which mailboxes are stored. A dialog box appears for the purpose. $HOME/Mail is used as the default.

Restrictions

None

6.94 XDMMshowMessage

Arguments
    None
Purpose
    Displays the current message in a window.
Restrictions
    Must only be called from the pop-up menu or as a defined method of a message.

6.95 XDMMshowSelectedMessages

Arguments

  None

Purpose

  Displays the selected messages in a window. You can scroll up and down through the list of messages that were selected when the method was called.

Restrictions

  None

6.96 XDMMsignature

Arguments
    None
Purpose
    Allows you to specify another signature file, which is appended to outgoing messages. By default it is the file $HOME/.signature.
Restrictions
    None

6.97 XDMMsortByDate

Arguments

None

Purpose

Sorts the messages in the mail manager view according to the dates on which they were sent. Mailboxes are sorted alphabetically by name.

Restrictions

None

6.98 XDMMsortBySender

Arguments

None

Purpose

Sorts the messages in the mail manager view alphabetically by the names of their senders. Mailboxes are sorted alphabetically by name.

Restrictions

None

6.99 XDMMsortBySize

Arguments

None

Purpose

Sorts the messages and mailboxes in the mail manager view according to their size.

Restrictions

None

6.100 XDMMsortByStatus

Arguments

None

Purpose

Sorts the messages in the mail manager view according to their "Status:" field. Mailboxes are sorted alphabetically.

Restrictions

None

6.101 XDMMsortBySubject

Arguments
    None
Purpose
    Sorts the messages in the mail manager view alphabetically according to their "Subject:" field. Mailboxes are sorted alphabetically.
Restrictions
    None

6.102 XDMMstoreMessage

Arguments

None

Purpose

Stores the current message in the folder you specify when requested to do so.

Restrictions

Must only be called from the pop-up menu or as a defined method of a message.

6.103 XDMMstoreSelectedMessages

Arguments

None

Purpose

Stores the currently selected messages in the mailbox you specify when requested to do so.

Restrictions

None

6.104 XDMMversionHelpShow

Arguments

None

Purpose

Displays a dialog box with information on the mail manager version.

Restrictions

None

6.105 XDNMattributes

Arguments
None
Purpose
Displays a dialog box to set the attributes for the current share or mount point.
Restrictions
None

6.106 XDNMnewMount

Arguments
None
Purpose
Displays a dialog box to set the attributes for a new mount point.
Restrictions
None

6.107 XDNMnewShare

Arguments
    None
Purpose
    Displays a dialog box to set the attributes for a new share point.
Restrictions
    None

6.108 XDNMsortBySpecial

Arguments

None

Purpose

Sets the sort order of mount points according to their remote name `server:pathname`.

Restrictions

None

6.109 XDNMversionHelpShow

Arguments
    None
Purpose
    Displays a dialog box with information on the NFS manager version.
Restrictions
    None

6.110 XDSMchangeMedium

Arguments

None

Purpose

Allows the user to change the software location.

Restrictions

None

6.111 XDSMcheckCons

Arguments

None

Purpose

Allows the user to check and restore the consistency of the product data base. A Check consistency dialog box is opened for this purpose.

Restrictions

None

6.112 XDSMcheckObject

Arguments

None

Purpose

Allows the user to check the accuracy of a package.

Restrictions

Must only be called from the popup menu or as a defined method of a package.

6.113 XDSMdeleteObject

Arguments

  None

Purpose

  Allows the user to delete the selected package from the system.

Restrictions

  Must only be called from the popup menu or as a defined method of a package or a product.

6.114 XDSMdeleteSelObjects

Arguments
    None
Purpose
Allows the user to delete the selected packages or products. An installation dialog box is opened for this purpose.
Restrictions
    None

6.115 XDSMdiskspaceProd

Arguments
    None
Purpose
    Shows the diskspace needed by the selected object.
Restrictions
    Must only be called from the popup menu or as a defined method of a product.

6.116 XDSMinfoOnObject

Arguments
    None
Purpose
    Shows extended information on a selected object in a window
Restrictions
    Must only be called from the popup menu or as a defined method of a package or product.

6.117 XDSMinfoOnSelObjects

Arguments

None

Purpose

Displays extended information on selected objects in a window. A scroll list is opened displaying information on the selected objects.

Restrictions

None

6.118 XDSMinstallObject

Arguments

  None

Purpose

  Allows a selected package or product to be installed.

Restrictions

  Must only be called from the popup menu or as a defined method of a package or product.

6.119 XDSMinstallSelObjects

Arguments

None

Purpose

Allows selected packages or products to be installed. An installation dialog box is opened for this purpose.

Restrictions

None

6.120 XDSMinteractSelPackages

Arguments

None

Purpose

Allows the user to execute the dialog with the selected software pakkage and store the responses in a file for later automatic installation. A dialog box is opened for this purpose.

Restrictions

None

6.121 XDSMinteractPackage

Arguments

None

Purpose

Allows the user to execute the dialog with the selected software package and store the responses in a file for later automatic installation.

Restrictions

Must only be called from the popup menu or as a defined method of a package.

6.122 XDSMreadinPackage

Arguments
 None
Purpose
 Allows selected software packages to be read from an installation medium into a spool directory for later installation.
Restrictions
 Must only be called from the popup menu or as a defined method of a product.

6.123 XDSMreadinSelPackages

Arguments

None

Purpose

Allows software packages to be read from an installation medium into a spool directory for installation. A read-in dialog box is opened for this purpose.

Restrictions

None

6.124 XDSMreleaseNote

Arguments
 None
Purpose
 Displays a release notice of a product in a window.
Restrictions
 Must only be called from the popup menu or as a defined method of a product.

6.125 XDSMshowSoftware

Arguments
    None
Purpose
    Displays the software packages or products on a new medium.
Restrictions
    None

6.126 XDSMshowSpace

Arguments

    None

Purpose

    Shows the disk space occupied by installed software.

Restrictions

    None

6.127 XDUMattributes

Arguments

None

Purpose

If the method is called in the `user_type` context, it displays a dialog box in which you can set user attributes. If the method is called in the `group_type` context, you can set group attributes. If the user manager is not running in system administrator mode, you can only change your password. If the user manager is running in system administrator mode, you can make changes in every field and assign a new image by dragging it from a bitmap browser and dropping it on the old image.

Restrictions

Must only be called from within a `user_type` or `group_type` XDCL description.

6.128 XDUMeditDefaults

Arguments

None

Purpose

Allows you to edit default settings used as a template for defining a new local user. Changes will be saved in `/usr/sadm/defadduser`.

Restrictions

The method is valid only when the user manager is in system administrator mode.

6.129 XDUMmakeLocal

Arguments

None

Purpose

Allows you to give a global user a local working environment. Changes to a local login are made by using `passwd -l` and will not be overwritten by the NIS services. The login remains however a global login on other machines since it is still included in the NIS maps.

Restrictions

Must only be called from within a `user_type` or `group_type` XDCL description.

6.130 XDUMnewGroup

Arguments

global | local

Purpose

Displays a dialog box in which you can define new groups with their characteristics.

Restrictions

It is not possible to define a new global user account on a NIS client host. The method is valid only when the user manager is in system administrator mode. New global goups may only be defined on the NIS master server.

6.131 XDUMnewUser

Arguments
: global | local

Purpose
: Displays a dialog box in which you can define new users with their characteristics.

Restrictions
: It is not possible to define a new global user account on a NIS client host. The method is valid only when the user manager is in system administrator mode. New global users may only be defined on the NIS master server.

6.132 XDUMpasswd

Arguments

None

Purpose

Allows you to change the password of the current user, i.e. the one whose icon is currently selected. This method can be used when the user manager is running in system administrator mode or when the real user ID is the same as the current user ID of the user. Normal users can change only their own passwords.

Restrictions

Must only be called from within a `user_type` or `group_type` XDCL description.

6.133 XDUMremoveUsersFromGroup

Arguments

List of users to be removed from a group.

Purpose

Removes the specified users from the current group.

Restrictions

It is not possible to remove a user from the `all_users` group or from his or her original group.

6.134 XDUMsortByID

Arguments

None

Purpose

Sets an internal variable of the user manager so that users are sorted and displayed according to their user IDs.

Restrictions

None

6.135 XDUMversionHelpShow

Arguments

None

Purpose

Displays a dialog box with information on the user manager version.

Restrictions

None

6.136 XDalwaysSave

Arguments

None

Purpose

Sets an internal program state in the object manager, so that on exit the current object manager state is always saved.

Restrictions

None

6.137 XDaskSave

Arguments
: None

Purpose
: Sets an internal program state in the object manager so that on exit you are asked whether the current object manager state is to be saved.

Restrictions
: None

6.138 XDbadDropSite

Arguments

None

Purpose

Displays a Motif error box, indicating that drops on the object have no effect.

Restrictions

None

6.139 XDcontextHelp

Arguments

None

Purpose

Changes the mouse pointer to a question mark. When you click the mouse the help method of the object under the pointer is invoked.

Restrictions

None

6.140 XDchangeFolder

Arguments

Optionally the name of a container, otherwise none

Purpose

Opens the specified container. If no container is specified, a prompt dialog is posted. If the `new_window_on_navigate` element has the value True, a new object manager window is opened; if not, an error message is displayed and the current window is re-used.

Restrictions

Unless a container name is specified as an argument, this method must only be called from within an `iconic_object` XDCL description of a file type, either from a pop-up menu or through a direct method such as `default_action`.

6.141 XDcloseWindow

Arguments
    None
Purpose
    Closes the current object manager window.
Restrictions
    None

6.142 XDcopyDroppedObjects

Arguments

None

Purpose

Copies dropped objects either to the container represented by the object on the drop site or, when the object is dropped on the background, to the container represented by the background of the object manager.

Restrictions

None

6.143 XDcopySelectedObjects

Arguments
    None
Purpose
    Displays a dialog box in which you can enter the container to which selected objects are to be copied.
Restrictions
    None

6.144 XDdefineHost

Arguments

None

Purpose

Shows a Motif prompt box asking you to enter the name of the host that the desktool is to use for remote object managers.

Restrictions

Must only be called for the remote object manager desktool.

6.145 XDdefineUser

Arguments
 None
Purpose
 Displays a dialog box in which you can specify the user name to be used by the desktool for remote object managers.
Restrictions
 Must only be called for the desktool for remote object managers.

6.146 XDdeleteObject

Arguments

None

Purpose

Deletes an object. In the case of some object managers - the file manager, for example - a question dialog box appears in which you are asked whether you really want to delete the object.

Restrictions

Must only be deleted from within an iconic object XDCL description; either from a pop-up menu or through a direct method such as `default_action`.

6.147 XDdeleteSelectedObjects

Arguments

None

Purpose

Deletes the selected objects from the current view of an object manager. In the case of some object managers - the file manager, for example - a question dialog box appears in which you are asked whether you really want to delete the objects.

Restrictions

None

6.148 XDdeselect

Arguments

  None

Purpose

  Shows a Motif prompt box asking you to enter a shell regular expression to match objects in the current object manager window. The matching objects are subtracted from the current selection. Objects not matching the pattern are unaffected. A match is made against the `label` attribute of the displayed objects, not against real file names.

Restrictions

  None

6.149 XDexit

Arguments
    None
Purpose
    Exits the current object manager and thereby implicitly the object manager program. In the case of the file manager, the program is only exited if no desktop manager is running. Displays a Motif question box asking you if you want to exit the object manager.
Restrictions
    None

6.150 XDextendSelect

Arguments

None

Purpose

Shows a Motif prompt box asking you to enter a shell regular expression to match objects in the current object manager window. The matching objects are added to the current selection. Previously selected objects are not deselected. A match is made against the `label` attribute of the displayed objects, not against real file names.

Restrictions

None

6.151 XDmoveSelectedObjects

Arguments

None

Purpose

Moves selected objects in the current view to another container. You can specify the name of the target container in a dialog box.

Restrictions

None

6.152 XDneverSave

Arguments

    None

Purpose

    Sets an internal program state in the object manager, so that on exit the current object manager state is never saved.

Restrictions

    Must only be called from within a `desktop_as_window` or `desktop_as_root` XDCL description or an object manager view.

6.153 XDnewSelect

Arguments

None

Purpose

Shows a Motif prompt box asking you to enter a shell regular expression to match objects in the current object manager window.

A match is made against the `label` attribute of the displayed objects, not against real file names. Previously selected objects are deselected if they do not match the entered expression.

Restrictions

None

6.154 XDnewWindowOnNavigate

Arguments

None

Purpose

Sets the `new_window_on_navigate` attribute of the current object manager window to true.

Restrictions

None

6.155 XDnoNewWindowOnNavigate

Arguments
  None
Purpose
  Sets the `new_window_on_navigate` attribute of the current object manager window to false.
Restrictions
  None

6.156 XDpollRateShow

Arguments

None

Purpose

Shows the poll rate dialog box for the current file manager or desktop manager view. If you click on `OK`, the value chosen using the slider bar in the box is used as the new poll rate. If there is an existing timed poll expected this is cancelled and a new poll timer initiated.

Restrictions

None

6.157 XDremoveObject

Arguments

None

Purpose

Removes the currently selected icon from the desktop.

Restrictions

Must only be called from an iconic object XDCL description of a file type or desktool within a `desktop_as_root` or `desktop_as_window` XDCL description; either from a pop-up menu or through a direct method, such as `default_action`.

6.158 XDremoveSelectedObjects

Arguments
    None
Purpose
    Removes icons selected in the current desktop window from the desktop.
Restrictions
    None

6.159 XDrestart

Arguments

None

Purpose

Shows a Motif question box asking you if you really wish to restart the object manager. If you select `OK` the object manager is restarted, using the last saved state.

Restrictions

None

6.160 XDrestoreFactorySettings

Arguments

None

Purpose

Restarts the desktop manager ignoring local configuration files. Local configuration files are renamed, so they can be retrieved if required.

Restrictions

None

6.161 XDsaveNow

Arguments

None

Purpose

Displays a question box asking whether you really want to save the current configuration of the object manager. If you click on `Ok`, the current state of the object manager is saved.

Restrictions

None

6.162 XDsetSortForward

Arguments
None
Purpose
Sets an internal object manager variable so that sorting algorithms sort in forwards order. The meaning of forwards or reverse is dependent on the sort key as follows:

Sort Key	**Forwards**	**Reverse**
Name	Ascending ASCII	Descending ASCII
Extension	Ascending ASCII	Descending ASCII
Time since last modified	Ascending Age	Descending Age
Time since last accessed	Ascending Age	Descending Age
Size	Ascending Size	Descending Size

Table 41: XDsetSortForward: sort key

Restrictions
None

6.163 XDsetSortReverse

Arguments

None

Purpose

Sets an internal file manager variable so that sorting algorithms sort in reverse order. The meaning of forwards or reverse is dependent on the sort key as follows:

Sort Key	**Forwards**	**Reverse**
Name	Ascending ASCII	Descending ASCII
Extension	Ascending ASCII	Descending ASCII
Time since last modified	Ascending Age	Descending Age
Time since last accessed	Ascending Age	Descending Age
Size	Ascending Size	Descending Size

Table 42: XDsetSortReverse: sort key

Restrictions

None

6.164 XDshowActiveUsers

Arguments

None

Purpose

Displays a dialog box in which a list of active users is shown. If you select the update button, an update is forced.

Restrictions

None

6.165 XDshowIconPathElement

Arguments

Pathname

Purpose

Opens a bitmap browser for the first element of the specified icon path.

Restrictions

None

6.166 XDsortByName

Arguments
    None
Purpose
    Set the sort order of mount point according to their absolute local path name.
Restrictions
    None

6.167 XDsysadminMode

Arguments

-on | -off | -toggle

Purpose

If -on is passed as the argument and the real user ID is 0, the method requests you to enter the system administrator password and changes to system administrator mode if the correct password is entered. If the argument is -off and the real user ID is not 0, system administrator mode is deactivated. If the argument is -toggle and the real user ID is not 0, system administrator mode is activated.

Restrictions

Must only be called from within a `devicemanager_view`, `nfsmanager_view`, `hostmanager_view`, `pkgmanager_view`, or `usermanager_view` XDCL description.

6.168 XDupdateObjectmanager

Arguments

None

Purpose

Causes the object manager to immediately refresh its view of the filesystem.

Restrictions

None

6.169 XDviewAsAlternateSet

Arguments

None

Purpose

Sets an internal variable in the object manager such that the alternative set of icon images is used to display the icons instead of the one defined as the default at system startup. There is one set of large and one set of small icon images. Which set is used as the default depends on the size of the screen. The icons are displayed with their images and labels.

Restrictions

None

6.170 XDviewAsExtended

Arguments
  None
Purpose
  Sets an internal variable in the file manager or desktop manager so that objects are displayed with extended labels (`long_labels`) showing more information on them.
Restrictions
  None

6.171 XDviewAsLabels

Arguments

None

Purpose

Sets an internal variable in the file manager or desktop manager such that objects in the window are displayed with their `label` only.

Restrictions

None

6.172 XDviewAsStandardSet

Arguments

None

Purpose

Sets an internal variable in the object manager such that the set of icon images defined as the default at system startup is used to display the icons. There is one set of large and one set of small icon images. Which set is used as the default depends on the size of the screen. The icons are displayed with their images and labels.

Restrictions

None

7 Boolean methods

Boolean methods do not execute actions initiated by the user; instead, they check the existence or otherwise of a particular state. If the state exists, boolean methods return the value true; if not, they return the value false. boolean methods are used to determine the sensitivity of menu items and buttons, i.e. whether or not they can be chosen.
For example, some menu items can only be chosen when the user is working in system administrator mode (UID 0). The `XDisNotSuperuser` method checks whether this is the case.

7.1 XDDVisMounted

Arguments
    None
Purpose
    Returns the value true if the current device is mounted.
Restrictions
    Only applicable if the current device is a file system device.

7.2 XDHMisSysadmOnNisServer

Arguments
: None

Purpose
: Returns the value true if the host is a NIS server and the user ID is 0.

Restrictions
: Must only be called from within a `usermanager_view`, `hostmanager_view`, `nfsmanager_view` or `devicemanager_view` definition.

7.3 XDHMisSysadmOnNonNisMaster

Arguments
    None
Purpose
    Returns the value true if the host is not a NIS master and the user ID is 0.
Restrictions
    Must only be called from within a `usermanager_view`, `hostmanager_view`, `nfsmanager_view` or `devicemanager_view` definition.

7.4 XDMIMEhasComposeCommand

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object has a compose command configured for it.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.5 XDMIMEhasEditCommand

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object has an edit command configured for it.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.6 XDMIMEhasPrintCommand

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object has a print command configured for it.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.7 XDMIMEhasViewCommand

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object has a view command configured for it.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.8 XDMIMEiconifyObject

Arguments

None

Purpose

Iconifies a `MIME_body_part_type` object if this object supports this action. (Only objects of MIME type "text/plain" and some image objects support this.)

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.9 XDMIMEisDeiconified

Arguments
: None

Purpose
: A boolean function that returns whether a MIME_body_part_type object is deiconified or not

Restrictions
: Can be called only in the context of a `MIME_body_part_type` object.

7.10 XDMIMEisIconic

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` object is iconic or not.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.11 XDMIMETextWordWrapIsOn

Arguments

None

Purpose

A boolean function that returns whether a `MIME_body_part_type` objects TextWordWrapping is set or not.

Restrictions

Can be called only in the context of a `MIME_body_part_type` object.

7.12 XDMMareMessagesSelected

Arguments

None

Purpose

Returns the value true if at least one message is selected in the mail manager view.

Restrictions

None

7.13 XDMMisOneMessageSelected

Arguments

    None

Purpose

    Returns the value true if only one message is selected in the mail manager view.

Restrictions

    None

7.14 XDisNisMaster

Arguments

None

Purpose

Returns the value true if the system is a master server of a NIS-Domain (i.e. YP_MODE=master) and running in system administrator mode.

Restrictions

Must only be called from within a `usermanager_view`, `hostmanager_view`, `nfsmanager_view` or `devicemanager_view` definition.

7.15 XDisNotLastOpenView

Arguments
    None
Purpose
    Determines whether the current view is the most recently opened view of an object manager. The method returns the value true if the current view is not the most recently opened one. If it is the most recently opened view, the method returns the value false.
Restrictions
    None

7.16 XDisNotNisMaster

Arguments

None

Purpose

Returns the value true if the system is running in system administrator mode on a client or slave server (i.e. YP_MODE!=master).

Restrictions

Must only be called from within a `usermanager_view`, `hostmanager_view`, `nfsmanager_view`, or `devicemanager_view` definition.

7.17 XDisNotSuperuser

Arguments

None

Purpose

Returns the value true if the real user ID of the user who started the manager is not 0, and false if the user ID is 0.

Restrictions

Must only be called from within a `devicemanager_view`, `hostmanager_view`, `nfsmanager_view`, `pkgmanager_view`, or `usermanager_view` definition.

7.18 XDisNotSysadminMode

Arguments

None

Purpose

Returns the value true if the object manager is running in system administrator mode.

Restrictions

Must only be called from within a `devicemanager_view`, `hostmanager_view`, `nfsmanager_view`, `pkgmanager_view`, or `usermanager_view` definition.

7.19 XDisShowBrowserSensitive

Arguments

None

Purpose

Determines the sensitivity of the pushbutton for displaying the individual browsers. The method returns the value false if the individual browsers are displayed and true if the browser overview is displayed.

Restrictions

None

7.20 XDisShowOverviewSensitive

Arguments

None

Purpose

Determines the sensitivity of the pushbutton for displaying the browser overview. The method returns the value false if the browser overview is displayed and true if the individual browsers are displayed.

Restrictions

None

7.21 XDisSysadminMode

Arguments

  None

Purpose

  Returns the value true if the manager is running in system administrator mode.

Restrictions

  Must only be called from within a `devicemanager_view`, `hostmanager_view`, `nfsmanager_view`, `pkgmanager_view`, or `usermanager_view` definition.

7.22 XDisSysadmDelMode

Arguments

None

Purpose

Returns the value true if the software manager is running in system administrator mode and the software location is an SVR4 medium.

Restrictions

Must only be called from within a `pkgmanager_view` definition.

7.23 XDisSysadmInstMode

Arguments

None

Purpose

Returns the value true if the software manager is running in system administrator mode and the software location is not "system".

Restrictions

Must only be called from within a `pkgmanager_view` definition.

7.24 XDisSysadmReadMode

Arguments

    None

Purpose

    Returns the value true if the software manager is running in system administrator mode and the software location is "system" or a spool directory.

Restrictions

    Must only be called from within a `pkgmanager_view` definition.

7.25 XDprogramIsAvailable

Arguments

-on_path | -display_vendor_string *regular_expression* |
-display_is_local

Purpose

Enables the entry and icon on a local machine.
If -on_path is passed as the argument and the program is available in the path, the method returns the value true.
If -display_vendor_string *regular_expression* is passed as the argument and *regular_expression* matches the X-Server vendor string (see xdpyinfo), the method returns the value true.
If -display_is_local is passed as the argument, the method returns the value true.

Restrictions

None

8 XDCL external methods

External methods are programs (with arguments) that are forked and executed as separate processes. Arguments are not passed to the shell for preprocessing, and shell built-in commands such as `cd` are not accepted.
A number of scripts in the directory `/usr/bin/X11/.xdeskdir` are used as predefined external methods by the object managers. These are listed below. You can use other external methods in the form of SINIX shell scripts or SINIX commands available on the search path defined by your PATH environment variable.

Method	Action
COMPRESS	Invokes the SINIX `compress` or `uncompress` command for a file depending on its extension and displays error messages in dialog boxes.
DECOMPRESS	Invokes the SINIX `uncompress` command for a file and displays error messages in dialog boxes.
DOCMD	Invokes a command and a wait command for use with emulator windows. Must be specified for some external methods.
EMPTYTRASH	Empties the whole trash directory.
EMULATOR	Invokes a command from a terminal emulator.
OMOPEN	Opens an object manager view.
PRINT	Invokes the print service.
TEXT_EDITOR	Invokes the text editor $VISUAL from the terminal emulator EMULATOR.
XAUTH	Transfers authorization information of the local login name to the authorization file of a user on a remote system and thereby allows remote clients to access the local display.
XDESKMORE	Invokes a pager when viewing long documents.
XRCP	Invokes the SINIX `rcp` command and displays error messages in dialog boxes.
XRSH	Invokes the SINIX `rsh` command which invokes a command for a user on a remote host and displays error messages in dialog boxes.

Table 43: XDCL external methods

9 XDCL object classes and elements

This chapter details the object classes and elements used in writing XDCL files.
It is not necessary for all elements to be defined and named. If an element is not defined, the default value will be used. Where it is not necessary to name an element, the element is enclosed in parentheses in the following tables.
The classes are arranged in alphabetical order. Each description begins by telling you the object manager to which the class belongs. General classes can be used for all object managers.

9.1 archivemanager_view

Archive manager class
This class defines the archive manager view.

Element	**Type**	**Default**	**Notes**
new_window_ on_navigate	boolean	False	If true, navigating to a new archive causes an additional view to be created.
initial_folder	SINIX filename	/	The archive to be viewed when this view is first displayed.
initial_poll_rat e	constant	120	poll rate in seconds in the range 0-600. A value of 0 disables polling.
font	X font name	value of the fontlist resource	Font used for standard icon labels.
small_font	X font name	value of the fontlist resource	Font used for small icon labels.
archive_entry_ types	group	null	A group of *archive_*

			entry_type instances to be used in conjunction with this object.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window _characteristics)	class *window _characteristics*	default	
(popup_menu)	class *popup_menu*	no popup	Defines the pop-up menu invoked when the host interacts with the window background.

Table 44: class archivemanager_view

Tip:

By default, an object class archivemanager_view recognizes only the following method:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 45: class archivemanager_view: methods

9.2 archive_entry_type

Archive manager class
This class defines match characteristics for an archive entry and associates an iconic object representation for archive entries that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `archive_entry_type`. However, for an archive to be associated with an `archive_entry_type`, the archive must match all the specified characterstics.
Instances of class `archive_entry_type` can be grouped within instance `archive_entry_types` of class `group` and belong to the `archivemanager_view` class.

Element	**Type**	**Defaul t**	**Notes**
(iconic object)	class *iconic_object*	no default	Must always be specified.
name_pattern	shell regexp	*	match characteristic on the entry's `%name`.
path pattern	shell regexp	not defined	match characteristic on the entry's `%pathname`.
owner	string	not defined	match characteristic on the entry's `%owner`.
group	string	not defined	match characteristic on the entry's `%owner_group`.
permissions_ pattern	shell regexp	not defined	match characteristic on the entry's `%permissions`.
size	comparative constant	not defined	match characteristic on the entry's

			`%size` in bytes.

Table 46: class archive_entry_type

9.3 button_group

General class

Element	**Type**	**Default**	**Notes**
foregrou nd	X color name	value of the foreground resource	
backgrou nd	X color name	value of the background resource	
font	X font name	value of the fontList resource	
(checkbu tton)	class *checkbutton*		0 - *n* of these elements.
(radiobutt on)	class *radiobutton*		0 - *n* of these elements.

Table 47: class button_group

9.4 cascadebutton

General class
This class attaches a cascade menu to a button, rather than a selection method. Cascade menus can be attached to other cascade menus. Class `cascadebutton` can belong to the following classes:

- button_group
- menu
- menubar
- popup_menu
- toolbar

Element	Type	Default	Notes
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
description	class *nls_string* or a string	null	A string that defines a description.
image	class *image*	null	An icon that defines an image.
link	class *nls_string* or a string	null	Specifies a button to which the selection status is linked.
label	class *nls_string* or a string	value of the labelString resource	
mnemonic	class *nls_string* or a string	value of the mnemonic resource	A single character that matches a character in the label.
justify_left	boolean	True	False justifies to the right.

			This value has no meaning when this class is used with a pop-up menu.
sensitivity	sensitivity kind	always	Evaluated prior to display of the menu to enable or disable the button.
help	method	null	Invoked when you request help on this object.
(menu)	class *menu*	default	Defines the menu to cascade from this button.

Table 48: class cascadebutton

9.5 checkbutton

General class
The following elements are used for the class **checkbutton**. Check buttons are used to set a feature of the object they define, for example, file permissions. A check button can have a value of a simple on/off or yes/no. A method can be assigned to the button depending on what the setting relates to.
The `set` or `unset` method is called on initialization of the menu containing this button, and also when the user explicitly changes the state of the button. The `set` method is called if the `initially_set` element is 'True'. The `unset` method is called if the `initially_set` element is 'False'.
Class "checkbutton" can belong to the following classes:
- button_group
- menu
- popup_menu
- toolbar

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
description	class *nls_string* or a string	null	A string that defines a description.
image	class *image*	null	An icon that defines an image.
link	class *nls_string* or a string	null	Specifies a button to which the selection status is linked.
label	class *nls_string* or a string	value of the labelString resource	
accelerator	class *nls_string* or a string	value of the accelerator resource	A string defining an accelerator.

acceler ator_te xt	class *nls_string* or a string	value of the acceleratorTex t resource	A string defining the label text of an accelerator.
mnemo nic	class *nls_string* or a string	value of the mnemonic resource	A single character that matches a character in the label.
initially _set	boolean	False	Determines the initial state of a button.
set	method	null	Called when you select this button.
unset	method	null	Called when you deselect this button.
sensitiv ity	sensitivity kind	always	Evaluated prior to display of the menu owning this button to enable or disable the button.
help	method	null	Invoked when you request help on this object.

Table 49: class checkbutton

Tip:

- If `accelerator` is defined, `accelerator_text` must also be defined, and vice versa.
- If `initially_set` has the value false, the `unset` method is called during initialization of a menu.
- If `initially_set` has the value true, the `set` method is called during initialization of a menu.

9.6 choice_dialog

General class
This class defines a collection of methods (held in radio buttons), one of which you are to choose from a dialog box. This class can only be used in place of the drop methods, `dropped_on_A`, `dropped_on_B` or `dropped_on_C`.

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
(radiobutton)	class *radiobutton*		0 - *n* of these elements.

Table 50: class choice_dialog

The following example shows how to use this class:

```
dropped_on_A = choice_dialog two_options_choice {
radiobutton option1 {
label = "option 1";
set = "external_method command_1";
}
radiobutton option2 {
label = "option 2";
set = "external_method command_2";
}
}
```

If the `dropped_on_A` method is invoked, a dialog box is displayed with two alternatives of "option 1" and "option 2", giving the user the choice of executing the external methods command_1 or command_2.

9.7 desktool

Desktop manager class
This class defines the location and appearance of desktools. Class `desktool` can be grouped within the following classes:
- desktop_as_root
- desktop_as_window

Element	Type	Default	Notes
geometry	geometry	auto	Position of the desktool.
deletability	boolean	True	If True the desktool may be deleted.
unique	boolean	True	If True, any object can occur only once in the toolbox. This is used in dragging and dropping.
(iconic_object)	class *iconic_object*	default	

Table 51: class desktool

9.8 desktop_as_root

Desktop manager class
This class defines a desktop manager view where the desktop is displayed as the background of the screen with the desktools placed along the side. A file manager view is placed in a window.

Element	**Type**	**Default**	**Notes**
initial_poll_ rate	number	10	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
file_types	class *group*	null	A group of *file_type* instances to be used in conjunction with this object.
new_windo w_on _navigate	boolea n	False	If True, navigating to a new object of class `toolbox` causes an additional view to be created.
initial_folde r	XDCL object referen ce	null	Defines the "toolbox" displayed when the view is opened for the first time.
import_targ ets	X atom list	_SNI_DES KTOOL _SNI_TOO LBOX _SNI_FILE FILE_NAM	Valid target list for importing to the window background by means of dragging and

		E	dropping.
(toolbox)	class *toolbox*	default	Defines the parent object of class *toolbox*.
menu_sensitivity	boolean	True	Defines whether menu items are sensitive in this view.
(methods)	class *methods*	See note	

Table 52: class desktop_as_root

Tip:

By default, an object of class `desktop_as_root` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.

Table 53: class desktop_as_root: methods

In the default configuration these methods enable files and desktools to be dropped on the desktop. Other methods can be defined, but will not be actioned.

- If no `toolbox` object is defined as a child of class `desktop_as_root`, the tree structure of the "toolbox" objects cannot be generated.

9.9 desktop_as_window

Desktop manager class
This class defines a desktop manager view in a decorated window.

Element	**Type**	**Default**	**Notes**
initial_poll_rate	number	10	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
file_types	class *group*	null	A group of *file_type* instances to be used in conjunction with this object.
new_window_on_navigate	boolean	False	If True, navigating to a new object of class `toolbox` causes an additional view to be created.
initial_folder	XDCL object reference	null	Defines which "toolbox" is displayed when the view is opened for the first time.
initial_folder_ depth	constant	2	Defines the number of levels of the "toolbox" hierarchy to be displayed when the desktop view is displayed.
import_targets	X atom list	_SNI_DESKTOOL _SNI_TOO	Valid target list for importing to the window

		LBOX _SNI_FILE FILE_NAME	background by means of dragging and dropping.
menu_sensitivity	boolean	True	Defines whether menu items are sensitive in this view.
(toolbox)	class *toolbox*	default	Defines the parent object of class `toolbox`.
(menubar)	class *menubar*	default	Defines the menus for the view.
(methods)	class *methods*	See note	
(window _characteristics)	class *window _characteristics*	default	
(popup_menu)	class *popup_menu*	default	Defines the pop-up menu invoked when you interact with the window background.

Table 54: class desktop_as_window

Tip:

By default, an object of class `desktop_as_window` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.

Table 55: class desktop_as_window: methods

In the default configuration these methods enable files and desktools to be dropped on the desktop. Other methods can be defined, but will not be actioned.

- If no `toolbox` object is defined as a child of class `desktop_as_window`, the tree structure of the "toolbox" objects cannot be generated.

9.10 devicemanager_view

Device manager class
This class defines the device manager view.

Element	Type	Default	Notes
initial_poll_rate	number	10	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontlist resource	Font used for standard icon labels.
small_font	X font name	value of the fontlist resource	Font used for small icon labels.
device_types	group	null	A group of *device_type* instances to be used in conjunction with this object.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_menu*	no popup	Defines the pop-up menu invoked when the host interacts with the window background.

Table 56: class devicemanager_view

Tip:

By default, an object class `devicemanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 57: class devicemanager_view: methods

9.11 device_type

Device manager class
This class defines match characteristics for a device and associates an iconic object representation for devices that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `device_type`. However, for a device to be associated with a `device_type`, the device must match all the specified characterstics.
Instances of class `device_type` can be grouped within instance `device_types` of class `group` and belong to the `devicemanager_view` class.

Element	**Type**	**Default**	**Notes**
(iconic object)	class *iconic_object*	no default	Must always be specified.
name_pattern	shell regexp	not defined	Match characteristic on the device's `%name`.
default_capacity	number	no default	Match characteristic on the device's `%default_capacity`.
device_kind	shell regexp	not defined	Match characteristic on the device's `%type`.
device_use_with_fs	boolean	not defined	Match characteristic on the device's files system state.

Table 58: class device_type

9.12 filemanager_view

File manager class
This class defines the file manager view.

Element	Type	Default	Notes
new_window_on_navigate	boolean	False	If True, navigating to a new folder causes an additional view to be created.
navigate_up_past_initial_folder	boolean	True	If True, the parent folder is displayed when viewing the specified *initial_folder*.
initial_folder	SINIX pathname	$HOME	The folder to be viewed when this view is first displayed.
initial_folder_depth	constant	2	Defines the number of levels of the folder hierarchy to be displayed when the file manager overview is opened.
initial_poll_rate	number	10	Poll rate in seconds in the range 0 - 600. A value of 0 disables

			polling.
file_types	class *group*	null	A group of *file_type* instances to be with this object.
font	X font name	value of the fontlist resource	Font used for standard icon labels.
small_font	X font name	value of the fontlist resource	Font used for small icon labels.
import_targe ts	X atom list	_SNI_FILE FILE_NAM E _SNI_ARC HIVE	Valid target list for importing to the window background by means of dragging and dropping.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window _characteris tics)	class *window _characteri stics*	default	
(popup_men u)	class *popup_men u*	default	Defines the pop-up menu invoked when you interact with the window background .

Table 59: class filemanager_view

Tip:

By default, an object of class `filemanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 60: class filemanager_view: methods

9.13 file_object

Desktop manager class
This class defines the location of file objects.

Element	**Type**	**Default**	**Notes**
aliasname	string	""	Free assignable name.
filename	SINIX filename	""	
pathname	SINIX pathname	""	
geometry	geometry	auto	Position of the file object.

Table 61: class file_object

9.14 file_type

Desktop manager class
This class defines match characteristics for a file and associates an iconic representation for files that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `file_type`. However, for a file to be associated with a `file_type`, the file must match all the specified characteristics.
Instances of class `file_type` can be grouped within instance `file_types` of class `group` and belong to the following classes:

- desktop_as_root
- desktop_as_window
- filemanager_view

Element	Type	Default	Notes
name_pattern	shell regexp		Match characteristic for `%filename`.
path_pattern	shell regexp	not defined	Match characteristic for `%pathname`.
magic	magic	not defined	Match characteristic for file type in accordance with the entries in `/etc/magic`.
owner	string	not defined	Match characteristic for %owner.
group	string	not defined	Match characteristic for `%owners_group`.
file_kind	file kind	not defined	Match characteristic for file's SINIX type.
readable_by_you	boolean	not defined	Match characteristic for access permissions for file.
writable_by_you	boolean	not defined	Match characteristic for access permissions for file.

executable_by_you	boolean	not defined	Match characteristic for access permissions for file.
alias_pattern	string	""	Match characteristic for file's free assignable name.
permissions_pattern	shell regexp	not defined	Alternative to the above for characteristics. This pattern matches the permissions string returned by the command ls -l. Cannot be defined if any of `file_kind`, `readable_by_you`, `writable_by_you`, or `executable_by_you`, are defined.
size	comparative number	not defined	Match characteristic for file's size in bytes.
age_since_last _modification	comparative age	not defined	Match characteristic for file's last modification time.
age_since_last _access	comparative age	not defined	Match characteristic for file's last access time.
(iconic_object)	class *iconic_object*	default	

Table 62: class file_type

9.15 group

General class
This class enables you to define any number of heterogeneous class instances as a group.
This class is used to collect instances for attachment to another object. For example, attaching desktools to a `desktop_as_root` or `desktop_as_window` object.
In a skeleton form, a typical `desktop_as_window` object uses class `group` in the following manner:

```
desktop_as_window daw1 {
// Child objects
menubar m1 {
...
}
window_characteristics w1 {
...
}
group file_types {
...
}
popup_menu p1 {                  // on click on background
...
}
group desktop_objects {          // desktools and file refs
...
}
methods m1 {                     // drop methods for desktop
...
}
// Attributes
initial_poll_rate = 20;          // poll every 20 seconds
...
}
```

9.16 group_type

User manager class
This class defines match characteristics for a user group and associates an iconic representation for groups that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `group_type`. However, for a user group to be associated with a `group_type`, the group must match all the specified characteristics.
Instances of class `group_type` can be grouped within instance `group_types` of class `group` and belong to class `usermanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified
name_pattern	shell regexp	not defined	Match characteristic for `%group_name`.
users_pattern	shell regexp	not defined	Match characteristic for `%users_in_group`.
gid	comparative number	not defined	Match characteristic for ID of the user group.
gtype	Possible values for this type: global, local, all	all	Match characteristic for global or local group

Table 63: class group_type

9.17 hostmanager_view

Host manager class
This class defines the host manager view.

Element	Type	Default	Notes
new_window_on_navigate	boolean	False	If True, navigating to a new network causes an additional view to be created.
initial_folder	network name	all_hosts	The network displayed when this view is opened for the first time.
initial_poll_rate	number	120	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
host_types	class *group*	null	Instance *host_types* of class *group*, which groups instances of class *host_type*.
network_types	class *group*	null	Instance *network_types* of class *group*, which groups instances of class *network_type*.

import_targets	X atom list	_SNI_NETWORK	Valid target list for importing by means of dragging and dropping
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_menu*	no pop-up menu	Specifies which pop-up menu is called when you click the window background using <BMenu>.

Table 64: class hostmanager_view

Tip:

By default, an object of class `hostmanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 65: class hostmanager_view: methods

9.18 host_type

Host manager class
This class defines match characteristics for a host and associates an iconic representation for hosts that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `host_type`. However, for a host to be associated with a `host_type`, the host must match all the specified characteristics.
Instances of class `host_type` can be grouped within instance `host_types` of class `group` and belong to class `hostmanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified.
address_pattern	shell regexp	not defined	Match characteristic for `%host_address`.
name_pattern	shell regexp	not defined	Match characteristic for `%host_name`.
alias_pattern	shell regexp	not defined	Match characteristic for `%host_alias`.
comment_pattern	shell regexp	not defined	Match characteristic for `%comment`.

Table 66: class host_type

9.19 icon

General class
This class defines the icon to be associated with a particular file type. Class `icon` can belong to the following class:
- iconic_object

Element	**Type**	**Default**	**Notes**
label	class *nls_string* or string	dynamic	
long_label	class *nls_string* or string	label	Displayed when you choose the extended format view.
max_label_length	constant	unrestricted	Maximum length of the label. Not used with `long_label`.
truncate_from_left	boolean	True	
(image)	class *image*	default	0 to *n* objects of this class.

Table 67: class icon

9.20 iconic_object

General class
This class defines the iconic representation of files, folders, desktools, hosts, networks, users and user groups. Class `iconic_object` can belong to the following classes:
- archive_entry_type
- device_type
- desktool
- file_type
- group_type
- host_type
- log_type
- message_type
- mount_type
- MIME_body_part_type
- network_type
- pkg_type
- share_type
- toolbox
- user_type

Element	**Type**	**Default**	**Notes**
name	class *nls_string* or string	object	
visibility	boolean	True	If True, the icon is visible.
selectability	boolean	True	If True, the icon can be selected.
dragabilityA	boolean	dynamic	If True, the icon can be dragged using <MCtrl> + <BTransfer>.
dragabilityB	boolean	dynamic	If True, the icon can be dragged using <MShift> + <BTransfe

			r>.
dragabilityC	boolean	True	If True, the icon can be dragged using <MShift> <MCtrl> + <BTransfer>.
panel	panel ID	dynamic	Defines the panel in which the icon is stored.
import_targets	X atom list	context-dependent (see note 1)	Target list for importing by means of dragging and dropping.
export_targets	X atom list	context-dependent (see note 1)	Target list exported by the object in dragging.
associated_program	string	""	See `%associated_program`.
(@user_attribute)	class *nls_string* or string	""	See note 2
(popup_menu)	class *popup_menu*	no pop-up menu	Defines the pop-up menu invoked when you interact with iconic object.
(icon)	class *icon*	default	
(methods)	class *methods*	See note 1	

Table 68: class iconic_object

Notes

1. Default values for target lists
   The export and import lists used depend on the object class of the object for which `iconic_object` is defined.

Object class	**Export Targets**	**Import targets**
desktool	_SNI_DESKTOOL _SNI_METHOD	_SNI_FILE FILE_NAME
toolbox	_SNI_TOOLBOX _SNI_METHOD	_SNI_DESK TOOL _SNI_TOOL BOX _SNI_FILE FILE_NAME
archive_entry_ty pe	_SNI_ARCHIVE _SNI_METHOD STRING TEXT	None
device_type	_SNI_ARCHIVE _SNI_METHOD STRING TEXT _SNI_DEVICE	_SNI_FILE
file_type	_SNI_FILE FILE_NAME _SNI_METHOD STRING TEXT	None
group_type	_SNI_GROUP _SNI_METHOD STRING TEXT	_SNI_USER USER
host_type	_SNI_HOST HOST _SNI_METHOD STRING TEXT	None
log_type	_SNI_METHOD STRING TEXT	None
message_type	_SNI_MAIL_MSG SNI_FILE_REG FILE_NAME _SNI_METHOD STRING TEXT	None
mount_type	_SNI_MOUNT _SNI_METHOD STRING TEXT	None
network_type	_SNI_NETWORK _SNI_METHOD	None

	STRING TEXT	
share_type	_SNI_SHARE _SNI_METHOD STRING TEXT	None
user_type	_SNI_USER USER _SNI_METHOD STRING TEXT	None
folder_types (Element of class group)	_SNI_MAIL_FOLDER _SNI_FILE_REG FILE_NAME_SNI_METHOD STRING TEXT	_SNI_MAIL_MSG
MIME_body_part_type	_SNI_MIME_BODY_PART _SNI_FILE FILE_NAME	_SNI_FILE FILE_NAME _SNI_MAIL_MSG _SNI_MIME_BODY_PART

Table 69: class iconic_object: export and import targets

2. Methods
   The methods of an iconic object can include any method, either internal or external, defined by name. The following method names have special meaning with respect to user interaction with an iconic object:

`default_action`	Executed when you double-click on the iconic object.
`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	Executed when you request context help on this object.

Table 70: class iconic_object: methods

   Other named methods can also be defined, such as `edit` or `print`. These methods can also be accessed via method-based desktools, or via the `xdtell -exec` option.
3. Attributes set by the user must begin with @. These attributes are expanded as variables in methods and other values.
   Example:

```
iconic_object fred {
@username="bloggs";
methods methods {
default_action="external_method xdeskbox
\"Fred's surname is {@username}\"";
}
}
```

9.21 image

General class
The elements below are used to define the icon images used by the object managers. Class `image` can be used in the following classes:
- icon
- window characteristics

Eleme nt	**Type**	**Default**	**Notes**
foregro und	X color name	value of the foreground resource	
backgr ound	X color name	value of the background resource	
name	SINIX filename	default.64	
mappin g	mapping	no mapping defined	Defines a mapping pattern for name

Table 71: class image

9.22 logmanager_view

Log manager class
This class defines the log manager view.

Element	Type	Default	Notes
initial_poll_rate	constant	120	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
log_types	class *group*	null	Instance *log_types* of class *group*, which groups instances of class *log_type*.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_menu*	no pop-up menu	Specifies which pop-up menu is called when you click the window background using <BMenu>.

Table 72: class logmanager_view

Tip:

By default, an object of class `logmanager_view` recognizes only the following methods:

`help`	executed when you request context help on this object.

Table 73: class logmanager_view: methods

9.23 log_type

Log manager class
This class defines match characteristics for a log entry and associates an iconic object representation for log entries that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `log_type.` However, for a log entry to be associated with a `log_type`, the log entry must match all the specified characterstics. Instances of class `log_type` can be grouped within instance `log_types` of class `group` and belong to the `logmanager_view` class.

Element	**Type**	**Default**	**Notes**
(iconic_ob ject)	class *iconic_object*	no default	Must always be specified.

Table 74: class log_type

9.24 mailmanager_view

Communcation manager class
This class defines the mail manager view.

Element	Type	Default	Notes
new_window_on_navigate	boolean	False	If True, navigating to a new mailbox causes an additional view to be created.
bell_volume	constant	50	Specifies the volume of the audible signal emitted when new mail arrives.
initial_folder	SINIX pathname	system mailbox	On System VR4 systems /var/mail/$USER
import_targets	X atom list	_SNI_FILE FILE_NAME _SNI_ARCHIVE	Defines the objects that can be dropped on the representation of this instance of class *filemanager_view*.
initial_poll_rate	number	60	Poll rate in seconds in the range 0 - 600. A value of 0 disables

			polling.
folders_directory	SINIX pathname	$HOME/Mail	Folder for storing the mailboxes of the user.
folder_types	class *group*	null	Instance *file_types* of class `group`, which groups together instances of class *file_type*.
message_types	class *group*	null	Instance *message_types* of class `group`, which groups together instances of class *message_type*.
mail_full_image	class *image*	null	Defines the icon which indicates there is mail in the system mailbox.
mail_empty_image	class *image*	null	Defines the icon which indicates there is no mail in the system mailbox.
signature_file	SINIX pathname	$HOME/.signature	Signature file of the user,

			which is appended to all outgoing mail.
reply_address	mail address	login name of the user	Sender address of the user, which is used for all outgoing mail.
outbox_folder	SINIX pathname	null	Default folder in which copies of outgoing mail are stored.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
(methods)	class *methods*	null (See also note)	A group of methods to be used in this view.
(popup_menu)	class *popup_menu*	no pop-up menu	Specifies which pop-up menu is called when you click on the window background using <BMenu>.
(menubar)	class	default	Defines

	menubar		the menus for the view.
(window _characteristics)	class *window _characteristics*	default	

Table 75: class mailmanager_view

Tip:

- The `message_types` group must be in the view for message input otherwise an error occurs at startup.
- The elements `mail_full_image` and `mail_empty_image` are only evaluated if the currently displayed mailbox is the system mailbox in which your host stores incoming mail. The image is also used as an icon defined in the view of `window_characteristics`.
- By default, an object of class `mailmanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 76: class mailmanager_view: methods

9.25 menu

General class
This class is used to define menus.

Tip:

A menu may not contain a mixture of checkbuttons and radiobuttons. Otherwise any number of instances of `pushbutton`, `cascadebutton`, `checkbutton`, `radiobutton` and `separator` classes may appear in a single definition.

Class `menu` can belong to the following class:
- cascadebutton

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
(pushbutton)	class *pushbutton*		0 - *n* of these elements.
(cascadebutton)	class *cascadebutton*		0 - *n* of these elements.
(checkbutton)	class *checkbutton*		0 - *n* of these elements.
(radiobutton)	class *radiobutton*		0 - *n* of these elements.
(separator)	class *separator*		0 - *n* of these elements.

Table 77: class menu

9.26 menubar

General class
This class is used to define the menubar at the top of a window.
Class "menubar" can belong to the following classes:

- desktop_as_window
- archivemanager_view
- devicemanager_view
- filemanager_view
- hostmanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
(cascadebutton)	class *cascadebutton*		0 - *n* of these elements.

Table 78: class menubar

9.27 message_type

Communication manager class
This class defines match characteristics for a message and associates an iconic representation for messages that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `message_type`. However, for a message to be associated with a `message_type`, the message must match all the specified characteristics.
Instances of class `message_type` can be grouped within instance `message_types` of class `group` and belong to class `mailmanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified.
content_type	shell regexp	not defined	Match characteristic for *content_type* according to RFC 822.
sender_pattern	shell regexp	not defined	Match characteristic for `%sender`.
subject_pattern	shell regexp	not defined	Match characteristic for `%subject`.
date_since_sent	comparative date	not defined	Match characteristic for the date on which the message was sent.
size	comparative number	not defined	Match characteristic for the message size in lines.
status	shell regexp	not defined	Match characteristic for the status field (see note).

Table 79: class message_type

Tip:

- The status field in a message header output by the communication manager can have one of the following entries:

N	new unread message
O	old unread message
R	read message
U	urgent or important message
D	message marked for deletion. It is deleted from the current folder at the next update or when the view is closed.

Table 80: class message_type: status field in message header output of communication manager

- Only messages with the content type "Text" are currently supported.
- An object of class `message_type` must define `(iconic_object)` and at least one attribute.

9.28 methods

General class
This class defines what methods are associated with an object.
Class `methods` can belong to the following classes:

- desktop_as_root
- desktop_as_window
- iconic_object
- archivemanager_view
- devicemanager_view
- filemanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- hostmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
(method)	method	null	0 - *n* of these elements.

Table 81: class methods

9.29 MIME_body_part_type

MIME manager class
This class defines match characteristics for a message and associates an iconic representation for messages that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `MIME_body_part_type`. However, for a message to be associated with a `MIME_body_part_type`, the message must match all the specified characteristics.
Instances of class `MIME_body_part_type` can be grouped within instance `MIME_body_part_types` of class `group` and belong to class `mimemanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	null	Description of the appearance of the object icon. Must be defined.
MIME_type_pattern	shell regexp	null	The MIME type of a body part corresponding to RFC 1521.
MIME_subtype_pattern	shell regexp	null	The MIME subtype of a body part corresponding to RFC 1521.
view_command	string	null	The name of a program used as external viewer for a MIME body part.
edit_command	string	null	The name of a program used as external editor for a MIME body part.
compose_command	string	null	The name of a program used as external compose editor for a MIME body part.
print_command	string	null	
show_deiconified	boolean	dynamic	Defines if a MIME body part is showed

			deiconified. Only text and some image body parts can be viewed deiconified.

Table 82: class MIME_body_part_type

9.30 mimemanager_view

MIME manager class
This class defines the MIME manager view.

Element	Type	Default	Notes
(menubar)	class *menubar*	null	Defines the menus for the view.
pushbutton	class *pushbutton*	default	Defines additional push buttons in the action area.
(methods)	class *methods*		
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_nenu*	no popup	Defines the popup menu when the user interacts with the window background.
initial_poll_rate	constant	0	Poll rate in seconds in the range 0-600. A value of 0 disables polling. It should be 0 for a MIME Manager View.
new_window_on_navigate	boolean	false	If true, navigation to a new MIME container opens a new view.
font	X font name	default	Font used for standard icon labels.

small_font	X font name	def ault	Font used for small icon labels.
MIME_body_part_t ypes	class *group*	null	The group of MIME_body_p art_type objects recognized by a MIME Manager View.
mimemanager_mo de	constant	0	Defines the mode of a MIME Manager View. Currently the values 0 (view mode) and 1 (compose/send mode) are supported.

Table 83: class mimemanager_view

9.31 mount_type

NFS manager class
The class `mount_type` defines match characteristics for a mount and associates an iconic object representation for mounts that match the specified characterstics. Not all match characteristics have to be defined in a given instance of `mount_type`. However, for a mount to be associated with a `mount_type`, the mount must match all the specified characteristics. Instances of class `mount_type` can be grouped within instance `mount_type` of class `group` and belong to class `nfs_manager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified.
special_pattern	shell regexp	not defined	Match characteristic for `%mount_special`.
mountp_pattern	shell regexp	not defined	Match characteristic for `%mount_point`.
fstype_pattern	shell regexp	not defined	Match characteristic for `%mount_fstype`.
automnt_pattern	shell regexp	not defined	Match characteristic for `%mount_automnt`.
mntopts_pattern	shell regexp	not defined	Match characteristic for `%mount_options`.
vfstab_entry	boolean	not defined	Match characteristic. If true the object has a mount tab entry.
currently_mounted	boolean	not defined	Match characteristic, If true the object is currently mounted.

Table 84: class mount_type

9.32 network_type

Host manager class
This class defines match characteristics for a network and associates an iconic representation for networks that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `network_type`. However, for a network to be associated with a `network_type`, the network must match all the specified characteristics.
Instances of class `network_type` can be grouped within instance `network_types` of class `group` and belong to class `hostmanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified.
address_pattern	shell regexp	not defined	Match characteristic for `%network_address`.
name_pattern	shell regexp	*	Match characteristic for `%network_name`. A valid network name consists of letters, numbers, '-'s, and '_'s.
alias_pattern	shell regexp	not defined	Match characteristic for `%network_alias`.
comment_pattern	shell regexp	not defined	Match characteristic for `%comment`.

Table 85: class network_type

9.33 nfsmanager_view

NFS manager class
This class defines the NFS manager view.

Element	**Type**	**Default**	**Notes**
font	X font name	value of the fontlist resource	Font used for standard icon labels.
small_font	X font name	value of the fontlist resource	Font used for small icon labels.
mount_types	group	null	A group of *mount_types* instances to be used in conjunctions with this objects.
share_types	group	null	A group of *mount_types* instances to be used in conjunctions with this objects.
initial_poll_rate	number	120	Update rate in seconds.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_menu*	no popup	Defines the pop-up menu invoked when you interact with the window background.

Table 86: class nfsmanager_view

Tip:

By default, an object class `nfsmanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	Executed when you request context help on this object.

Table 87: class nfsmanager_view: methods

9.34 nls_string

General class
Strings that are language-dependent, such as menu button labels and window titles, may be internationalized using this class in place of a literal string. NLS is a standard message file format for the internationalization of SINIX programs. For information on how to generate NLS message catalogs please refer to the *X/Open Portability Guide*.
Class `nls_string` can belong to the following classes:

- cascadebutton
- checkbutton
- icon
- iconic_object
- pushbutton
- radiobutton
- window_characteristics
- xdesk

Element	**Type**	**Default**	**Notes**
filename	SINIX filename	""	Name of the NLS file containing the message.
set_number	number	-1	Defines the NLS set number containing the message.
msg_number	number	-1	Defines the NLS message number.
default_string	string	"nls_message"	Used if the NLS message cannot be retrieved.

Table 88: class nls_string

9.35 pkgmanager_view

Package manager class
This class defines the package manager view.

Element	Type	Default	Notes
initial_folder	product name	system:all_installed	
initial_poll_rate	number	120	Update rate in seconds
new_window_on_navigate	boolean	False	If True, navigating to a new network causes an additional view to be created.
font	X font name	value of the fontList resource	Font used for standard icon labels.
pkg_types	group	null	A group of package type instances
prod_types	group	null	A group of product type instances
small_font	X font name	value of the fontList resource	Font used for small icon labels.
(methods)	class *methods*	See note	
(menubar)	class *menubar*	default	Defines the menus for the view.
(window_characteristics)	class *window_characteristics*	default	
(popup_menu)	class *popup_menu*	no pop-up menu	Specifies which pop-up menu is called when you click the window

			background using <BMenu>.

Table 89: class pkgtmanager_view

Tip:

By default, an object of class `pkgmanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.
`help`	executed when you request context help on this object.

Table 90: class pkgmanager_view: methods

9.36 pkg_type

Package manager class
This class defines match characteristics for a software package and associates an iconic representation for packages that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `pkg_type`. However, for a package to be associated with a `pkg_type`, the package must match all the specified characteristics. Instances of class `pkg_type` can be grouped within instance `pkg_types` of class `group` and belong to class `pkgmanager_view`.

Element	Type	Default	Notes
(iconic_object)	class *iconic_object*	no default	Must always be specified
name_pattern	shell regexp	*	Match characteristic for `%pkgname`.
inst_state	Possible values for this type: complete, partial, dif_vers, not	no default	Match characteristic for installation status of `%package`
medium	Possible values for this type: yes, no	no default	Match characteristic for installed or spooled software

Table 91: class pkg_type

9.37 popup_menu

General class
This class is used to define the pop-up menu associated with an object type.

Tip:

A menu may not contain a mixture of checkbuttons and radiobuttons.
Otherwise any number of instances of `pushbutton`, `cascadebutton`, `checkbutton`, `radiobutton` and `separator` classes may appear in a single definition.

Class "popup_menu" can belong to the following classes:
- filemanager_view
- iconic_object
- desktop_as_window
- archivemanager_view
- devicemanager_view
- hostmanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
(pushbutton)	class *pushbutton*		0 - *n* of these elements.
(cascadebutton)	class *cascadebutton*		0 - *n* of these elements.
(checkbutton)	class *checkbutton*		0 - *n* of these elements.
(radiobutt	class		0 - *n* of

on)	*radiobutton*		these elements.
(separator)	class *separator*		0 - *n* of these elements.

Table 92: class popup_menu

9.38 prod_type

Package manager class
This class defines match characteristics for a software product and associates an iconic representation for products that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `prod_type`. However, for a product to be associated with a `prod_type`, the product must match all the specified characteristics. Instances of class `prod_type` can be grouped within instance `prod_types` of class `group` and belong to class `pkgmanager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified
name_pattern	shell regexp	not defined	Match characteristic for `%prodname`.

Table 93: class prod_type

9.39 pushbutton

General class
This class defines the pushbuttons which can be used in menus and dialog boxes.
Class "pushbutton" can belong to the following classes:
- button_group
- menu
- popup_menu
- toolbat

Element	Type	Default	Notes
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
label	class *nls_string* or a string	value of the labelString resource	
accelerator	class *nls_string* or a string	value of the accelerator resource	A string that defines an accelerator.
accelerator_text	class *nls_string* or a string	value of the acceleratorText resource	A string that defines the label text of an accelerator.
description	class *nls_string* or a string	null	A string that defines a description.
image	class *image*	null	An icon that defines an image.
mnemonic	class *nls_string* or a string	value of the mnemonic resource	A single character that matches the character in the label.
select	method	null	Invoked

			when you click on this button.
sensitivity	sensitivity kind	always	Evaluated prior to display of the menu to enable or disable the button.
help	method	null	Invoked when you request help on this object.

Table 94: class pushbutton

Tip:

If `accelerator` is defined, `accelerator_text` must also be defined, and vice versa.

9.40 radiobutton

General class
This class defines radiobuttons which can be used in menus and dialog boxes.
The `set` or `unset` method is called on initialization of the menu containing this button and also when the user explicitly changes the state of the button.
The `unset` method is called on initialization of a menu if the `initially_set` element is set to False. The `set` method is called on initialization of a menu if the `initial_set` element is set to True.
Class `radiobutton` can belong to the following classes:
- button_group
- choice_dialog
- menu
- popup_menu
- toolbar

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
description	class *nls_string* or a string	null	A string that defines a description.
image	class *image*	null	An icon that defines an image.
link	class *nls_string* or a string	null	Specifies a button to which the selection status is linked.
label	class *nls_string* or a string	value of the labelString resource	
accelerator	class *nls_string* or a	value of the accelerator resource	A string that defines an accelerator.

	string		
accelerator_text	class *nls_string* or a string	value of the acceleratorText resource	A string that defines the label text of an accelerator.
mnemonic	class *nls_string* or a string	value of the mnemonic resource	A single character that matches the character in the label.
initially_set	boolean	False	Determines the initial state of a button.
set	method	null	Invoked when you select this radiobutton.
unset	method	null	Invoked when you deselect this radiobutton.
sensitivity	sensitivity kind	always	Evaluated prior to display of the menu to enable or disable the button.
help	method	null	Invoked when you request help on this object.

Table 95: class radiobutton

Tip:

If `accelerator` is defined, `accelerator_text` must also be defined, and vice versa.

9.41 separator

General class
This class provides the separator bars in the toolbar and in menus.
Class `separator` can belong to the following classes:
- menu
- popup_menu
- toolbar

Element	Type	Default	Notes
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	

Table 96: class separator

9.42 share_type

NFS manager class
The class `share_type` defines match characteristics for a share and associates an iconic object representation for shares that match the specified characterstics. Not all match characteristics have to be defined in a given instance of `share_type`. However, for a share to be associated with a `share_type`, the share must match all the specified characteristics. Instances of class `share_type` can grouped within instance `share_types` of class `group` and belong to class `nfs_manager_view`.

Element	**Type**	**Default**	**Notes**
(iconic_object)	class *iconic_object*	no default	Must always be specified.
fstype_pattern	shell regexp	not defined	Match characteristic for `%share_fstype`.
options_pattern	shell regexp	not defined	Match characteristic for `%share_options`.
description_pattern	shell regexp	not defined	Match characteristic for `%share_description`.
pathname_pattern	shell regexp	not defined	Match characteristic for `%share_pathname`.
dfstab_entry	boolean	not defined	Match characteristic. If true the object has a share tab entry.
currently_shared	boolean	not defined	Match characteristic. If true the object is currently shared.

Table 97: class share_type

9.43 statusbar

General class
This class is used to define the statusbar at the bottom of a window.
Class "statusbar" can belong to the following classes:

- desktop_as_window
- archivemanager_view
- devicemanager_view
- filemanager_view
- hostmanager_view
- logmanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
font	X font name	value of the fontList resource	
label	class *nls_string* or a string	value of the label String resource	

Table 98: class statusbar

9.44 toolbar

General class
This class is used to define the toolbar at the top of a window.
Class "toolbar" can belong to the following classes:
- desktop_as_window
- archivemanager_view
- devicemanager_view
- filemanager_view
- hostmanager_view
- logmanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foregroun d resource	
background	X color name	value of the backgroun d resource	
link	class *nls_string* or a string	null	Specifies a button to which the selection status is linked.
(checkbutto n)	cla ss *checkbutton*		0 - *n* of these elements.
(pushbutton)	class *pushbutton*		0 - *n* of these elements.
(radiobutton)	class *radiobutton*		0 - *n* of these elements.
(separator)	class *separator*		0 - *n* of th se elements.

Table 99: class toolbar

9.45 toolbox

Desktop manager class
This class defines a desktop toolbox.

Element	Type	Default	Notes
geometry	geometry	auto	Position of the toolbox.
deletability	boolean	True	If True, you delete the toolbox.
unique	boolean	True	If True, each object can occur in the toolbox only once. This is used in dragging and dropping.
(iconic_obje ct)	class *iconic_object*	default	
desktop_obj ects	class *group*	null	Defines a group of objects of class *desktool*, *toolbox* or *file_object*, which are displayed in this toolbox.

Table 100: class toolbox

9.46 usermanager_view

User manager class
This class defines a user manager view.

Element	Type	Default	Notes
new_window_on_navigate	boolean	False	If True, navigating to a new group causes an additional view to be created.
initial_folder	SINIX group name	all_users	The user group displayed when this view is opened for the first time.
initial_poll_rate	number	10	Poll rate in seconds in the range 0 - 600. A value of 0 disables polling.
font	X font name	value of the fontList resource	Font used for standard icon labels.
small_font	X font name	value of the fontList resource	Font used for small icon labels.
user_types	class *group*	null	Instance *user_types* of class `group`, which groups together instances of class *user_type*.

group_types	class *group*	null	Instance *group_types* of class `group`, which groups together instances of class *group_type*.
import_targets	X atom list	_SNI_GROUP	Valid target list for importing by means of dragging and dropping.
(menubar)	class *menubar*	default	Defines the menus for the view.
(methods)	class *methods*	See note.	
(popup_menu)	class *popup_menu*	no pop-up menu	Specifies which pop-up menu is called when you click on the window background using <BMenu>.

Table 101: class usermanager_view

Tip:

By default, an object of class `usermanager_view` recognizes only the following methods:

`dropped_on_A`	executed when you drop an object on the desktop background using <MCtrl> + <BTransfer>.
`dropped_on_B`	executed when you drop an object on the desktop background using <Shift> + <BTransfer>.
`dropped_on_C`	executed when you drop an object on the desktop background using <MShift> <MCtrl> + <BTransfer>.

`help`	executed when you request context help on this object.

Table 102: class usermanager_view: methods

9.47 user_type

User manager class
This class defines match characteristics for a user and associates an iconic representation for users that match the specified characteristics. Not all match characteristics have to be defined in a given instance of `user_type`. However, for a user to be associated with a `user_type`, the user must match all the specified characteristics.
Instances of class `user_type` can be grouped within instance `user_types` of class `group` and belong to class `usermanager_view`.

Element	Type	Default	Notes
(iconic_object)	class *iconic_object*	no default	Must always be specified.
name_pattern	shell regexp	*	Match characteristic for `%user_name`.
uid	comparative number	not defined	Match characteristic for the user's user ID.
group_pattern	shell regexp	not defined	Match characteristic for `%group_name`.
gid	comparative number	not defined	Match characteristic for the user's group ID.
utype	Possible values for this type: global, local, all	all	Match characteristic for global or local user.
comment_pattern	shell regexp	not defined	Match characteristic for `%comment`.
home_directory _pattern	shell regexp	not defined	Match characteristic for `%home_directory`.

shell_pattern	shell regexp	not defined	Match characteristic for `%user_directory`.

Table 103: class user_type

9.48 window_characteristics

General class
This class defines the window characteristics for a particular window.
Class `window_characteristics` can belong to the following classes:
- desktop_as_window
- archivemanager_view
- devicemanager_view
- filemanager_view
- hostmanager_view
- mailmanager_view
- mimemanager_view
- nfsmanager_view
- pkgmanager_view
- usermanager_view

Element	**Type**	**Default**	**Notes**
foreground	X color name	value of the foreground resource	
background	X color name	value of the background resource	
title	class *nls_string* or string	value of the title resource	
geometry	geometry	auto	Position and size of the window, or "auto".
mwm_iconize	boolean	True	If True, the Motif window manager allows iconization.
mwm_close	boolea	True	If True, the

	n		Motif window manager allows closure.
fat_scrollbars	boolean	False	If True, the window scrollbars are fat.
smoothscroll	boolean	True	If True, scrolling with the 'thumb' causes immediate changes to the window's interior.
background_image	class *image*	default	The image used for the window background.
iconized_image	class *image*	default	The image used when the window is iconized.
_overwiew_panel_geometry	geometry	auto	Size hint for the overview panel.
_default_panel_geometry	geometry	auto	Size hint for the default panel.
*_panel_geometry	geometry	auto	Size hint for the named panel, where * stands for the panel name.

Table 104: class window_characteristics

9.49 xdesk

Desktop manager class
This class defines the file manager and desktop manager instances that are to be used.

Element	Type	Default	Notes
filemanager	class *nls_string* or string	default	
root_desktop	class *nls_string* or string	default	
decorated_desktop	class *nls_string* or string	default	

Table 105: class xdesk

Tip:

The elements `root_desktop` and `decorated_desktop` specify the name of the instance to be used when you change the desktop type interactively. Both must be defined in this class. If the `rootDesktop` resource has the value True at startup, the desktop manager tries to place and open the instance defined by `root_desktop`. If the value is False, the desktop manager tries to place and open the instance defined by `decorated_desktop`.

10 Installation utilities

Three installation utilities are provided to enable applications to install themselves into the desktop environment. It is envisaged that these utilities will be run as part of the installation and deinstallation scripts of an application package, and run with superuser permissions. If the utilities are not run as superuser it is likely that they will fail due to permissions problems in accessing and modifying desktop configuration files and directories.

10.1 xdaddftype - File type installation utility

This utility adds a file type to or removes a file type from the desktop configuration. It does this by editing the `ins_ftypes.cfg` file in the `/usr/lib/X11/xdesk/cfg` directory. For added file types to appear the next time a session is started, there must be an `#include` directive for this file in the user's desktop configuration files. The utility also allows you to copy files into the above-named directory or delete them from it.
The XDCL syntax of added definitions is not checked. The same applies to the uniqueness of the instance names of added XDCL objects. It may be useful to make instance names start with the name of the package in which the added XDCL objects are supplied. Even this does not guarantee uniqueness in every case, as users or system administrators may have freely assigned their own instance names.
A sample XDCL file type definition is supplied as `ftype.tmpl` in `/usr/lib/X11/xdesk/cfg`. This file also contains a checklist to help you write your own file type.
The following exit codes are returned:

0	Successful completion
1	Bad argument specification
2	Could not open system configuration files for reading
3	Could not write new configuration file
4	File type is already installed
5	Could not copy specified file to system configuration location
7	Could not write to system configuration files
8	Could not delete specified file
9	Could not deinstall specified file, as it was not installed

Table 106: xdaddftype: exit codes

xdaddftype[*option*]...[*filename*]...

Option	**Description**	**Default**
-?	outputs a usage line and exits.	

-d	deinstalls the specified file type and files *filenames*.	xdaddftype installs the specified file type and files *filenames*.
filena me	If the `-d` option is not specified, copies the file *filename* to `/usr/lib/X11/xdesk/cfg` and includes the XDCL file in the system configuration XDCL file `ins_ftypes.cfg` in the same directory. If the `-d` option is specified, deletes the file *filename* from `/usr/lib/X11/xdesk/cfg` and removes the #include directive from the system configuration XDCL file `ins_ftypes.cfg` in the same directory.	No default.
-h	outputs a usage line and exits.	
-v	verbose mode prints diagnostics to stdout.	xdaddftype does not print diagnostics to stdout.

Table 107: xdaddftype command line options

10.2 xdaddadmtool - Desktool installation utility

This utility adds a desktool to or removes a desktool from the desktop configuration. It does this by editing the `adm_tools.cfg` file in the `/usr/lib/X11/xdesk/cfg` directory. Desktools included in the `adm_tools.cfg` file appear in the administration toolbox. For added desktools to appear in the desktools catalog and be available for moving out onto the user-specific desktop the next time a session is started, there must be an `#include` directive for this file in the user's desktop configuration files. The utility also allows you to copy files into the above-named directory or delete them from it.
Desktool generation is not possible unless the `-filename` *filename* option is specified.
The XDCL syntax of added definitions is not checked. The same applies to the existence of files such as those specified in defining an XDCL object of XDCL class "image", as this depends on the value of the XDESK_ICON_PATH environment variable at the time when the client interpreting the XDCL description is started up. Similarly, the uniqueness of the instance names of added XDCL objects is not checked. It may be useful to make instance names start with the name of the package in which the added XDCL objects are supplied. Even this does not guarantee uniqueness in every case, as users or system administrators may have freely assigned their own instance names.
A sample XDCL desktool definition is supplied as `desktool.tmpl` in `/usr/lib/X11/xdesk/cfg`. This file also contains a checklist to help you write your own desktool.
The following exit codes are returned:

0	Successful completion
1	Bad argument specification
2	Could not open system configuration files for reading
3	Could not write new configuration file
4	Desktool is already installed
5	Could not copy specified file to system configuration location
6	Could not write generated XDCL description file
7	Could not write to system configuration files
8	Could not delete specified file
9	Could not deinstall specified file, as it was not installed

Table 108: xdaddadmtool: exit codes

Note that some options also take a file name as a parameter.
xdaddadmtool[*option*]...[*filename*]...

Option	**Description**	**Default**
-?	outputs a usage line and exits.	
-d	deinstalls the	xdaddadmt

	specified desktool and files *filenames*.	ool installs the specified desktool and files *filenames*
-default_action *method*	see `-filename` option specifies the method to be called when a user double-clicks on the generated desktool.	No default.
filename	If the `-d` option is not specified, copies the file *filename* to `/usr/lib/X11/xdesk/cfg` and includes the XDCL file in the system configuration XDCL file `adm_tools.cfg` in the same directory. If the `-d` option is specified, deletes the file *filename* from `/usr/lib/X11/xdesk/cfg` and removes the #include directive from the system configuration XDCL file `adm_tools.cfg` in the same directory.	No default
-filename *filename*	generates a desktool in the file *filename*. Desktool generation is not possible unless the `-filename` *filename* option is specified.	No default
-h	outputs a usage line and exits.	
-image *filename*	see `-filename` option specifies the name of the file containing the normal image for the generated desktool.	No default
-label *string*	see `-filename` option specifies the string	No default

	used as a label of the generated desktool.	
-mask *filename*	see `-filename` option specifies the name of the file containing the normal mask for the generated desktool.	No default
-small_image *filename*	see `-filename` option specifies the name of the file containing the small image for the generated desktool.	No default
-small_mask *filename*	see `-filename` option specifies the name of the file containing the small mask for the generated desktool.	No default
-v	verbose mode prints diagnostics to stdout.	xdaddadmtool does not print diagnostics to stdout

Table 109: xdaddadmtool command line options

10.3 xdaddtool - Desktool installation utility

This utility adds a desktool to or removes a desktool from the desktop configuration. It does this by editing the `ins_tools.cfg` file in the `/usr/lib/X11/xdesk/cfg` directory. Desktools included in the `ins_tools.cfg` file appear in the applications toolbox. For added desktools to appear in the desktools catalog and be available for moving out onto the user-specific desktop the next time a session is started, there must be an `#include` directive for this file in the user's desktop configuration files. The utility also allows you to copy files into the above-named directory or delete them from it.
Desktool generation is not possible unless the `-filename` *filename* option is specified.
The XDCL syntax of added definitions is not checked. The same applies to the existence of files such as those specified in defining an XDCL object of XDCL class "image", as this depends on the value of the XDESK_ICON_PATH environment variable at the time when the client interpreting the XDCL description is started up. Similarly, the uniqueness of the instance names of added XDCL objects is not checked. It may be useful to make instance names start with the name of the package in which the added XDCL objects are supplied. Even this does not guarantee uniqueness in every case, as users or system administrators may have freely assigned their own instance names.
A sample XDCL desktool definition is supplied as `desktool.tmpl` in `/usr/lib/X11/xdesk/cfg`. This file also contains a checklist to help you write your own desktool.
The following exit codes are returned:

0	Successful completion
1	Bad argument specification
2	Could not open system configuration files for reading
3	Could not write new configuration file
4	Desktool is already installed
5	Could not copy specified file to system configuration location
6	Could not write generated XDCL description file
7	Could not write to system configuration files
8	Could not delete specified file
9	Could not deinstall specified file, as it was not installed

Table 110: xdaddtool: exit codes

Note that some options also take a file name as a parameter.
xdaddtool[*option*]...[*filename*]...

Option	**Description**	**Default**
-?	outputs a usage line and exits.	
-d	deinstalls the	xdaddtool

	specified desktool and files *filenames*.	installs the specified desktool and files *filenames*
-default_action *method*	see `-filename` option specifies the method to be called when a user double-clicks on the generated desktool.	No default
filename	If the `-d` option is not specified, copies the file *filename* to `/usr/lib/X11/xdesk/cfg` and includes the XDCL file in the system configuration XDCL file `ins_tools.cfg` in the same directory. If the `-d` option is specified, deletes the file *filename* from `/usr/lib/X11/xdesk/cfg` and removes the #include directive from the system configuration XDCL file `ins_tools.cfg` in the same directory.	No default
-filename *filename*	generates a desktool in the file *filename*. Desktool generation is not possible unless the `-filename` *filename* option is specified.	No default
-h	outputs a usage line and exits.	
-image *filename*	see `-filename` option specifies the name of the file containing the normal image for the generated desktool.	No default
-label *string*	see `-filename` option specifies the string used as a label of the generated desktool.	No default

-mask *filename*	see `-filename` option specifies the name of the file containing the normal mask for the generated desktool.	No default
-small_image *filename*	see `-filename` option specifies the name of the file containing the small image for the generated desktool.	No default
-small_mask *filename*	see `-filename` option specifies the name of the file containing the small mask for the generated desktool.	No default
-v	verbose mode prints diagnostics to stdout.	xdaddtool does not print diagnostics to stdout

Table 111: xdaddtool command line options

11 Related publications

[1] SINIX/windows
Documentation Overview

[2] SINIX/windows User Environment
Introduction to Handling and Configuration
(SINIX Desktop)
User Guide

[3] SINIX/windows User Environment
Guide for Experts and System Administrators
(SINIX Desktop)

[4] SINIX/windows User Environment
Clients Reference Manual
(SINIX Desktop)

[5] SINIX/windows User Environment
Desktop Kornshell User's Guide
(Common Desktop Environment)

[6] SINIX/windows User Environment
User's Guide
(Common Desktop Environment)

[7] SINIX/windows User Environment
TED Enhancements
(Common Desktop Environment)

[8] SINIX/windows User Environment
Advanced User and System Administrator Guide
(Common Desktop Environment)

[9] SINIX/windows Development
Product Overview

[10] SINIX/windows Development
OSF/Motif
Style Guide Release 1.2

[11] SINIX/windows Development
OSF/Motif
Programmer's Reference, Release 1.2

[12] SINIX/windows Development
OSF/Motif
Programmer's Guide, Release 1.2

[13] SINIX/windows Development

OSF/Motif
X Server Reference Manual

[14] SINIX/windows Development
X Window System, Xt Toolkit Intrinsics
Reference Manual

[15] SINIX/windows Development
X Window System X11, Xlib
Programming Manual

[16] SINIX/windows Development
X Window System, Xlib
Reference Manual

[17] SINIX/windows Development
X Window System X11 Rel.5 (SINIX), X Input Extension
Programmer's Reference

[18] SINIX/windows Development
X Window System, X Protocol
Reference Manual

[19] SINIX/windows Development
Widget Set
Programmer's Reference

[20] SINIX/windows Development
Programmer's Guide
(Common Desktop Environment)

[21] SINIX/windows Development
Programmer's Overview
(Common Desktop Environment)

[22] SINIX/windows Development
Application Builder User's Guide
(Common Desktop Environment)

[23] SINIX/windows Development
Help System Author's and Programmer's Guide
(Common Desktop Environment)

[24] SINIX/windows Development
Internationalization Programmer's Guide
(Common Desktop Environment)

[25] SINIX/windows Development
Style Guide and Certification Checklist
(Common Desktop Environment)

[26] SINIX/windows Development

Tooltalk Messaging Overview
(Common Desktop Environment)

[27] SINIX/windows Development
Tooltalk Programmer's Guide
(Common Desktop Environment)

Ordering manuals

The manuals listed above and the corresponding order numbers can be found in the Siemens Nixdorf *List of Publications*. New publications are described in the *Druckschriften Neuerscheinungen (New Publications)*.
You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Please apply to your local office, where you can also order the manuals.

Glossary

A

access permissions
The attributes of a file that are used to control who can read, write to, or run the file.

application
A program, such as a word processor, a shell command or a shell script.

B

bitmap Browser
A window containing a collection of all the pictures that can be used as *icons* to represent objects.

C

cascade button
A pushbutton that is used to display a cascading menu.

checkbutton
A pushbutton that is used to let you choose a number of selections from a number of possible choices.

container
Holds groups of an object manager's objects. Containers are displayed in object manager views and contain individual objects. The purpose of containers varies from one object manager to another. For the file manager they are folders, for the mail manager mailboxes, for the user manager user groups, and for the host manager networks.

container object
An *object* representing a *container* in a *window* or on the *desktop*. Every container object is symbolized by an *icon*.

D

default action
The action executed when you double-click on an *object*.

desktools
Applications that are controlled by the *desktop manager*. Each desktool is represented by an object on the desktop.

desktools catalog
A window containing copies of all the *desktools*. Desktools from this window can be added to those already on the *desktop*.

desktop
The background on which the windows and other objects are placed. See also *Desktop-as-root* and *Desktop-as-window*.

desktop manager
The client that controls the desktop and the objects that appear on it.

desktop-as-root
The desktop manager setting in which the *desktop* covers the screen and the *desktools* are arranged down the right-hand side.

desktop-as-window
The desktop manager setting in which the desktop is confined to a window. The *desktools* are displayed in this window.

dialog box
A window requesting, or giving, information, or requiring you to select from a number of possible choices.

drag and drop
Data transfer between clients in which a user drags an object across the screen using the mouse, and then drops it on another object, thereby executing an action.

E **element**
A building block of an *object class*. For example, the class "image" can be constructed from elements defining aspects such as foreground and background color.

executable file object
An object, either on the desktop or in a file manager window, that represents an application.

expansion variables
Variables that are preceded by a '%' character, for example `{%filename}`. The string assigned to the variable is used in place of the variable name.

external program
An *application* available as a shell command. External program and *internal method* are opposites.

F **file manager**
The client that controls the file manager windows and the objects that appear within them.

file manager view
The contents of each file manager *window*. Windows can be used to display different views of the same *folder*.

file type
A group of similar files; all folders, for example, are of the same file type, and all files with the extension `doc` can be treated as a file type.

folder
An area of the SINIX file system that can contain files and other folders. A folder is where your work, and the work of others, will be stored on the filesystem.

I **icon**
Small graphical images that represent a *file*, *folder*, or *desktool*. When they are minimized,

windows are also represented by icons. Icons representing windows can be kept in an icon box of the window manager mwm.

internal method
A method to act on an object only certain objects know of. The client xdesk, for example, knows internal methods to act on *desktools*, *files* and *folders*.

L **label**
The title or description of a button or *icon*.

location cursor
The equivalent to the mouse pointer when using the keyboard. The location cursor is a box drawn around the *object* in a container that will be affected by the next input.

M **menu**
A list of options. You can click on a menu option using the mouse, thus invoking the function defined for that option.

menubar
A rectangular area below the *title bar* that contains the titles of the pull-down *menus*.

method
An action that is associated with an object; that is, it defines what the object does in response to a particular event taking place. For example, a pushbutton object has a method that defines what happens when you select that pushbutton from a menu.

mwm window manager
The application that enables windows to be manipulated, such as by moving, minimizing, or maximizing.

O **object**
A basic unit that performs its own *methods* and usually has its own visual appearance as an icon, for example *files*, *folders* and *desktools*.

object class
A group of *objects* of the same kind that have common *elements* defined by the desktop configuration language XDCL. For example, *menus* form an object class.

P **parent object**
The *object* in a container window with the label `parent`, representing the container on the level above the container displayed in the window.

pushbutton
A button in menus and dialog boxes that can be pushed with the mouse or keyboard causing the action specified by the button label.

R **radiobutton**

A button that lets you make a single selection from a number of possible choices represented by a group of radiobuttons.

regular file object
An *object* in a file manager *window*, or on the *desktop*, that represents a file such as a text file.

remote file manager
A file manager running on a host machine other than the one managing your display.

S **statusbar**
A rectangular area at the bottom of a window that provides instant information about a current operation, or other context.

string
A series of letters and/or numbers bounded by double quote characters. For example, "This is a string".

T **This (object)**
The *object* in a container window representing the container displayed in the window.

toolbar
A rectangular area below the *menu bar* that contains toolbar buttons representing their operation picorially. A toolbar provides quick access to often used program functions.

top-level object
An *object* whose description in an XDCL file is not enclosed in braces "{}".

W **window menu**
The menu that is displayed when you click on the window menu button. This menu enables you to manipulate the window in various ways. The window menu is part of the *mwm window manager*.

Index