UnixWare®

Porting Notes

Copyright 1990,1991,1992,1993 UNIX System Laboratories Europe, Ltd
Copyright 1994 Novell UK Ltd.
All Rights Reserved
Printed in UK
No part of this publication may be reproduced or transmitted in any form or by any means--graphic, electronic, electrical, mechanical, or chemical, including photocopying, recording in any medium, taping, by any computer or information storage and retrieval systems, etc., without prior permissions in writing from Novell UK.

This document was typeset using Documentors Workbench Release 3.1 on UNIX® System V Release 4.2.

IMPORTANT NOTE TO USERS

While every effort has been made to ensure the accuracy of all information in this document, Novell assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document, its updates, supplements, or special editions, whether such errors are omissions or statements resulting from negligence, accident, or any other cause.

Novell further assumes no liability arising out of the application or use of any product or system described herein; nor any liability for incidental or consequential damages arising from the use of this document.

Novell disclaims all warranties regarding the information contained herein, whether expressed, implied or statutory, *including implied warranties of merchantability or fitness for a particular purpose*.

Novell reserves the right to make changes without further notice to any products herein to improve reliability, function, or design.

Any comments on this document should be sent by electronic mail to *list-admin@novell.co.uk*.

2 Extending the UNIX Desktop

Introduction 2-1

Enhancing the File Class Database 2-2
Creating the Icons 2-2
Defining a File Class 2-5
Testing Your Metaphor 2-7

Integrating the Drag-and-drop Mechanism 2-10
Design Objectives 2-10
Creating the Utility Icons 2-11
Modifying the File Class Database 2-13
Testing Your Enhanced Metaphor 2-15
A Note about Minimize and Close 2-17

Describing the File Class Database 2-18

Determining the File Class of an Icon 2-21

Properties 2-22
Desktop Properties 2-23
File Class Properties 2-23

The Icon Definition File 2-26
Icon Definition File Location 2-27
Icon Definition File Syntax 2-27
- Help File Specifications 2-27
- Icon Specifications 2-28
- File Class Specifications 2-30

Example 2-30

Introduction

The UNIX desktop is a graphical implementation of the UNIX system where all resources are represented by folder windows and icons and can be manipulated using menus and drag-and-drop. Directories are represented as windows and files within directories are graphically shown as icons. Printing can be performed through menu selection or drag-and-drop. Standard tools are implemented via graphical interfaces and are menu-driven. The UNIX desktop provides several means to allow you to enhance the UNIX desktop by creating new icons and have these icons represent existing applications or even new applications.

By simply modifying the file class database, you can provide new icons to represent existing applications and specific data files. This chapter provides the steps for taking advantage of this database.

This chapter describes the File Class Database and the Icon Definition File, and presents an example of how to add new classes for the `compress` and `uncompress` commands.

Enhancing the File Class Database

The desktop manager partitions all files into a set of classes. Every file on the UNIX system belongs to a class and each class has its own graphical representation (icon) and a set of properties. The file class database contains the file class descriptions and defines the icons that represent each file class. This section provides a simple example on how you can manipulate the database to create your own images and cursors.

This example calls for the design of shirts representing data files that are candidate for compression via the `compress` command. Folded shirts represent the `compress`ed files and must have the capability to be uncompressed.

Creating the Icons

If you have read/write privilege to the `/usr/X/lib` directory, create bitmap versions of the icons using `bitmap` first. Then, use the `Pixmap_Editor` (or `olpixmap`) to create the color version of the icons. This simply requires retrieving the bitmap file from the `Pixmap_Editor`.

If you don't have the proper privileges, then simply create the icons using `bitmap` (for monochrome systems) or `Pixmap_Editor` (for color systems).

Figure 2-1 and Figure 2-2 show the icons for the data file (unfolded shirt) and the compressed file (folded shirt), respectively.

Figure 2-1: Data File Icon

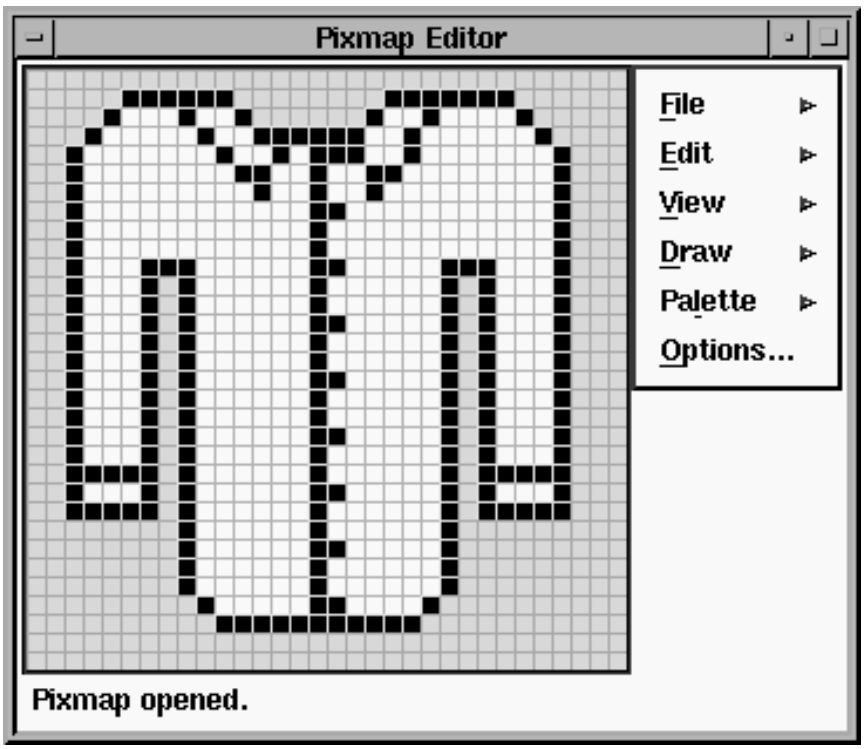
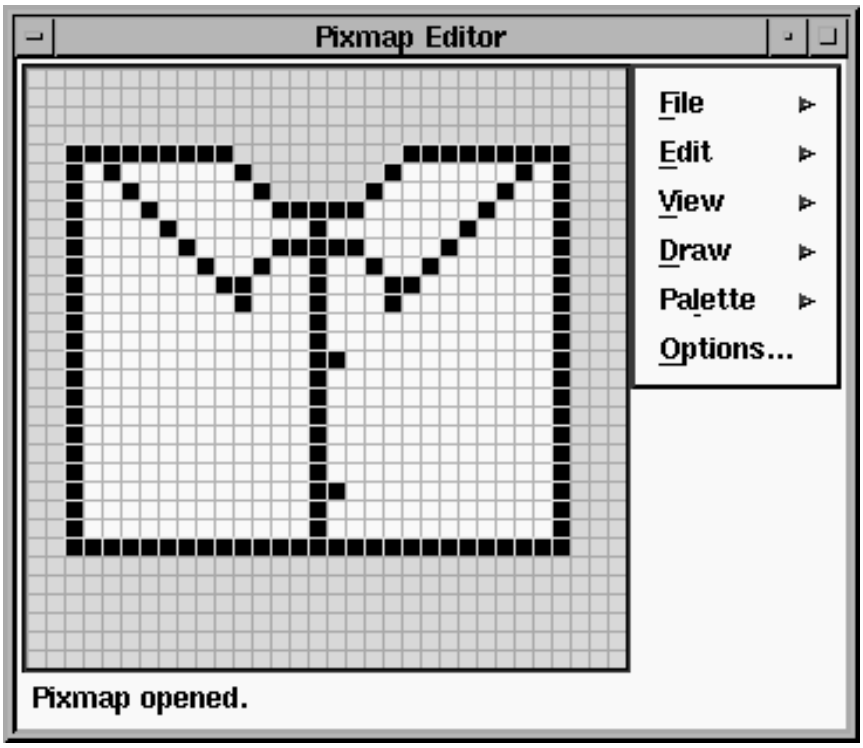


Figure 2-2: Compressed File Icon



For color systems, save these icons in the directory `/usr/X/lib/pixmaps` as:

```
/usr/X/lib/pixmaps/shirt.xpm
/usr/X/lib/pixmaps/fldshirt.xpm
```

For monochrome systems, save these icons in the directory `/usr/X/lib/bitmaps` as:

```
/usr/X/lib/bitmaps/shirt.xpm
/usr/X/lib/bitmaps/fldshirt.xpm
```

NOTE If you do not have the privilege to write in these directories, save these files in any directory. Just make sure you specify the full pathnames when defining the `_ICONFILE` property discussed in the next section.

Here are some additional guidelines that you need to know assuming that you have read/write privileges to the appropriate directories:

- If you intend to support both monochrome and color systems, you must have both the pixmap and bitmap versions of the icons located in the corresponding directories with the same file names.

- For color systems, you can also provide a pixmask version of the icon saved in the `/usr/X/lib/pixmasks` directory. This contains the ''mask'' for the color icon which may have a problem related to bleed-through of the window background color. If this pixmask file is not provided, then the pixel in the upper left hand corner is assumed to be transparent.
- To get a discernable ''cursor'' when doing a drag-and-drop on the icons, a corresponding mask should exist located in `/usr/X/lib/pixmasks`.
- If both a pixmap and bitmap are provided, every pixmask that exists must have a corresponding bitmask file saved in the `/usr/X/lib/bitmasks` directory.

All these files (specific to a particular icon) should have the same name.

Defining a File Class

A file class defines certain characteristics of a particular file type. Each definition is a named list of properties. The syntax for a file class definition is:

```
CLASS name
BEGIN
        [attributes]     property_name   property_value;
        .
        .
END
```

where

name is the name of the file class.

attributes are the attributes associated with a file class property.

property_name is the name of the property.

property_value is the value of the property.

NOTE

See the section ''File Class Database I/O'' for a full discussion on this database.

You can use the `Icon_Setup` utility to add entries in the file class database.

For the file compression and uncompression metaphor, the following file class database is appropriate:

```
CLASS pfile
BEGIN
      _PATTERN     "sample*.Z";
      _FILETYPE    "DATA";
      _ICONFILE    "fldshirt.xpm";
      MENU  Uncompress    "exec /usr/bin/uncompress %F &";
END

CLASS unpfile
BEGIN
      _FILETYPE    "DATA";
      _PATTERN     "sample*";
      _ICONFILE    "shirt.xpm";
      MENU  Compress      "exec /usr/bin/compress %F &";
END
```

Let's examine this database file.

The file defines two classes: `unpfile` (uncompressed files) and `pfile` (compressed files).

The `FILETYPE` property for both shows that files of these classes (uncompressed and compressed files) are data files. What distinguishes the uncompressed files from the compressed files is the extension `.Z`. Note that the compress command creates a compressed file with this extension. Thus, the use of the `PATTERN` property for `pfile`.

The `ICONFILE` property specifies the name of the icon file. Recall the icons you have created in the earlier section. If you saved the icons in a directory other than `/usr/X/lib/pixmaps`, you must specify the full pathnames of the icon files. Otherwise, it searches the directory `/usr/X/lib/pixmaps` by default.

The `MENU` attribute contains the Compress and Uncompress options for the corresponding file class. This attribute displays the specified property as a button in the icon menu. Thus, `Compress` and `Uncompress` appear in the icon menu for the shirt and folded shirt icons, respectively. The `%F` variable specifies the full path name of the file that is acted upon by the menu option.

NOTE

Property values may be enclosed in single or double quotes.

If you need to protect your icons (and the corresponding objects that they represent), use the `_SYSTEM` property with any value starting with either a "y" or a "Y." This property prevents the object from being dragged and dropped into the wastebasket.

Testing Your Metaphor

To recognize the effects of your newly created file class database, you must save its contents in `$HOME/.dtfclass` (this is the default database file path if the Desktop property `_FILEDB_PATH` is not set). Once you initialize the UNIX desktop, you should get the appropriate icons as in Figure 2-3.

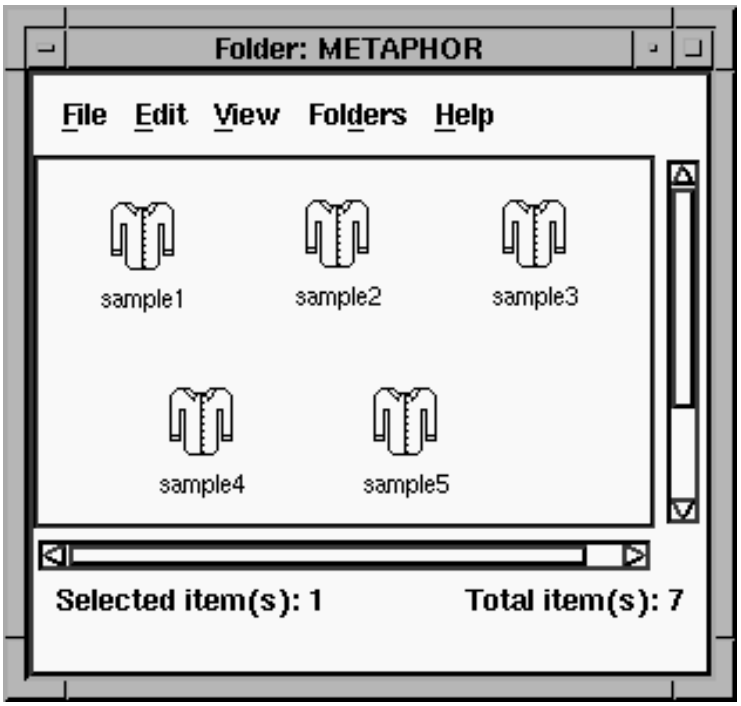
NOTE

Add the new file class entries before the existing DATA file class in the `.dtfclass` file provided with your system.

Make sure that your file class database does not have any syntax error. Otherwise, you will not be able to initialize the UNIX desktop.

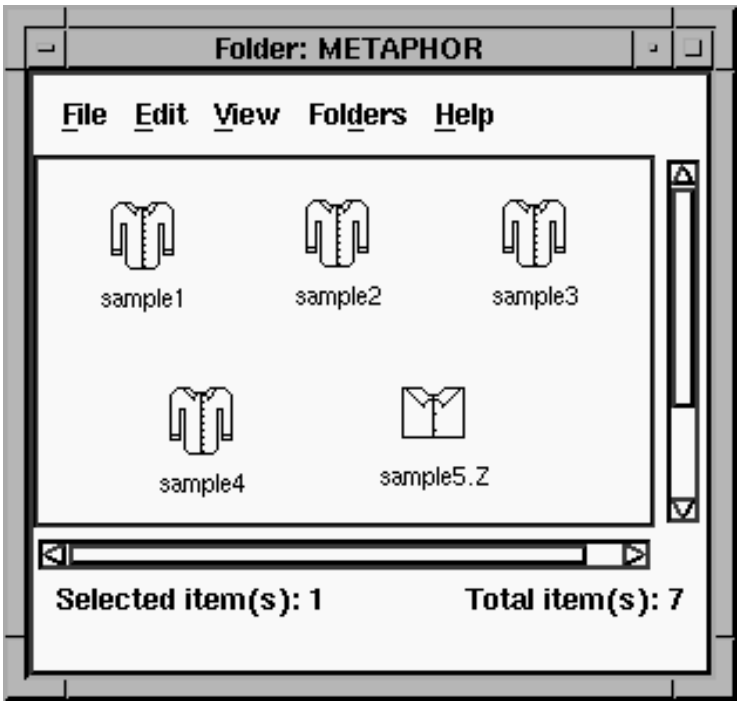
Changes made to the file class database (not using the `Icon_Setup` utility) while UNIX desktop is up will not be in effect until the next session.

Figure 2-3: Sample Folder



Click the MENU button/key on any shirt and get its icon menu. One of the options should be `Compress` (as specified by the `MENU` property). Choose this option. See how the shirt becomes a folded shirt as in Figure 2-4.

Figure 2-4: Compressed File Icon



You can uncompress the folded shirt by clicking on its menu again. Its menu should give you `Uncompress` as one of its options. Choose this option and notice how the shirt goes back to its original state.

Integrating the Drag-and-drop Mechanism

The Drag-and-drop mechanism allows the UNIX desktop user to pick up an object called the source with an input device (such as a mouse), drag the object to a desired location called the destination, and drop the object there. This operation initiates an operation or a transfer of data depending on the source and destination.

The following can be implemented using the Drag-and-drop programming interface:

- moving/copying a file into another directory
- moving a file into the wastebasket for deletion
- moving a file to a printer for printing

Building upon the file class database of the previous section, this section illustrates the Drag-and-drop mechanism by providing the interface for submitting files for `compress`ing and `uncompress`ing.

Design Objectives

The example of the previous section provides the `compress` and `uncompress` capability through the shirt icons. With the drag-and-drop mechanism, you can extend the capability that allows you to submit the shirts to destinations that provide the compress/uncompress functions. By designing a ''laundry room'' image, your can provide the interface for sending files (shirts) to the `compress` (a washer icon) or `uncompress` (a dryer icon) utility for processing. With the Drag-and-drop mechanism, the end-user should be able to point to a file (represented by the shirts), drag it to the dryer or washer, and drop it for processing. This mechanism handles the following situations:

- a compressed file is submitted for uncompressing, and
- an uncompressed file is submitted for compressing.

Creating the Utility Icons

As in the previous chapter, create the washer and dryer icons using the `Pixmap_Editor`.

NOTE

Same guidelines apply for monochrome and color systems as provided in the earlier section.

Figure 2-5 shows the `compress` command icon.

Figure 2-5: Compress Command Icon

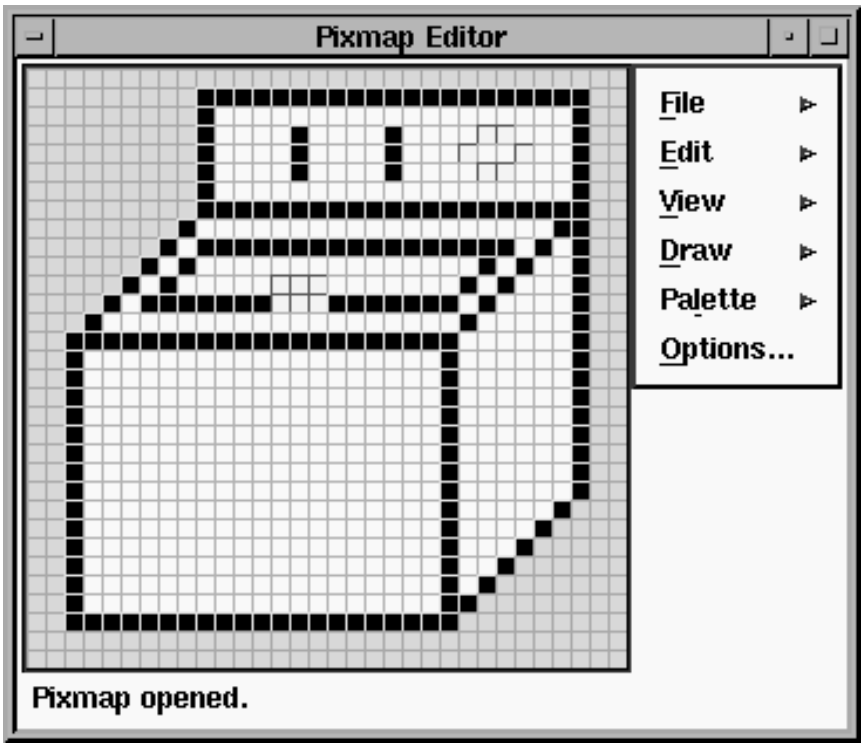
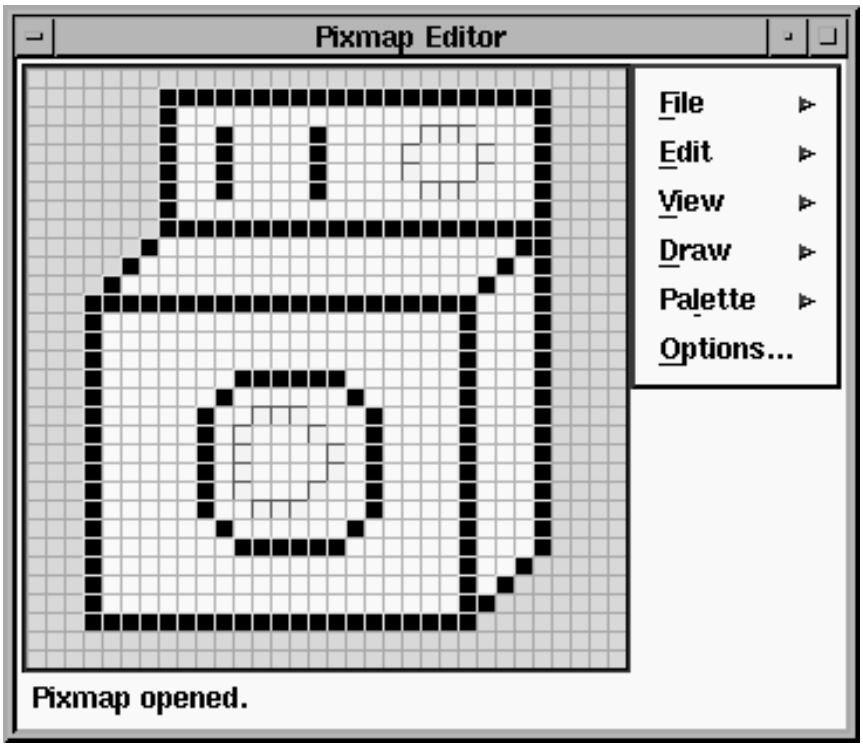


Figure 2-6 shows the `uncompress` command icon.

Figure 2-6: Uncompress Command Icon



Save these icons in the following files:

```
/usr/X/lib/pixmaps/washer.xpm
/usr/X/lib/pixmaps/dryer.xpm
```

Modifying the File Class Database

Implementing the Drag-and-drop mechanism requires additional entries in the file class database. Thus, to implement the washer and dryer metaphor, you need the following entries in the database:

```
CLASS compressing
BEGIN
      _PATTERN     "compress";
      _ICONFILE    "washer.xpm";
      _FILETYPE    "EXEC";
      _DROP        "compress %S";
END

CLASS uncompressing
BEGIN
      _PATTERN     "uncompress";
      _ICONFILE    "dryer.xpm";
      _FILETYPE    "EXEC";
      _DROP        "uncompress %S";
END
```

NOTE

As mentioned earlier, you can use the `Icon_setup` utility to add these entries to the database.

The `DROP` property provides the Drag-and-drop interface to these two classes (compressing and uncompressing). The value provided is a shell script that performs the necessary action when another icon is dropped to the icon provided by these classes.

The `ICONFILE`s created via the `Pixmap_Editor` (washer.xpm and dryer.xpm) are specified in the appropriate file classes. The `PATTERN` property specifies the file class qualifier that matches the filenames `compress` and `uncompress`.

The commands `compress` and `uncompress` have to be present in the `metaphor` directory. An alternative file class specification is:

```
CLASS compressing
BEGIN
      _LPATTERN    "compress";
        _LFILEPATH "/usr/bin";
      _ICONFILE    "washer.xpm";
      _FILETYPE    "EXEC";
      _DROP        "compress %S";
END

CLASS uncompressing
BEGIN
      _LPATTERN    "uncompress";
      _LFILEPATH   "/usr/bin";
      _ICONFILE    "dryer.xpm";
      _FILETYPE    "EXEC";
      _DROP        "uncompress %S";
END
```

If either `LPATTERN` or `LFILEPATH` is specified, the file must be a symbolic link in order to qualify.

The LPATTERN property specifies the symbolic link to the file class qualifier that matches the patterns compress and uncompress in the path specified by LFILEPATH. However, you need to invoke the following commands in the directory of interest:

```
ln -s /usr/bin/compress compress
ln -s /usr/bin/uncompress uncompress
```

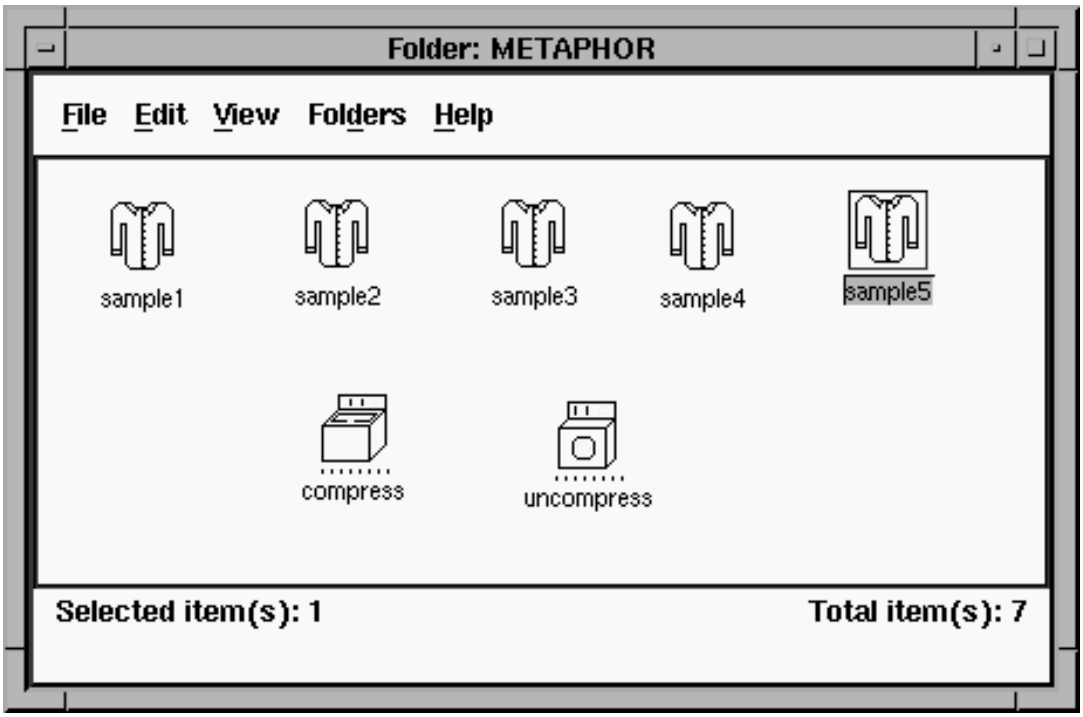
The %S specifies the full pathname of the source file that is being dropped on to the icon. This variable is used only in the DROP property value.

Testing Your Enhanced Metaphor

As in the previous chapter, save your newly created file class in $HOME/.dtfclass. Then, reinitialize the UNIX desktop and test your utility room metaphor.

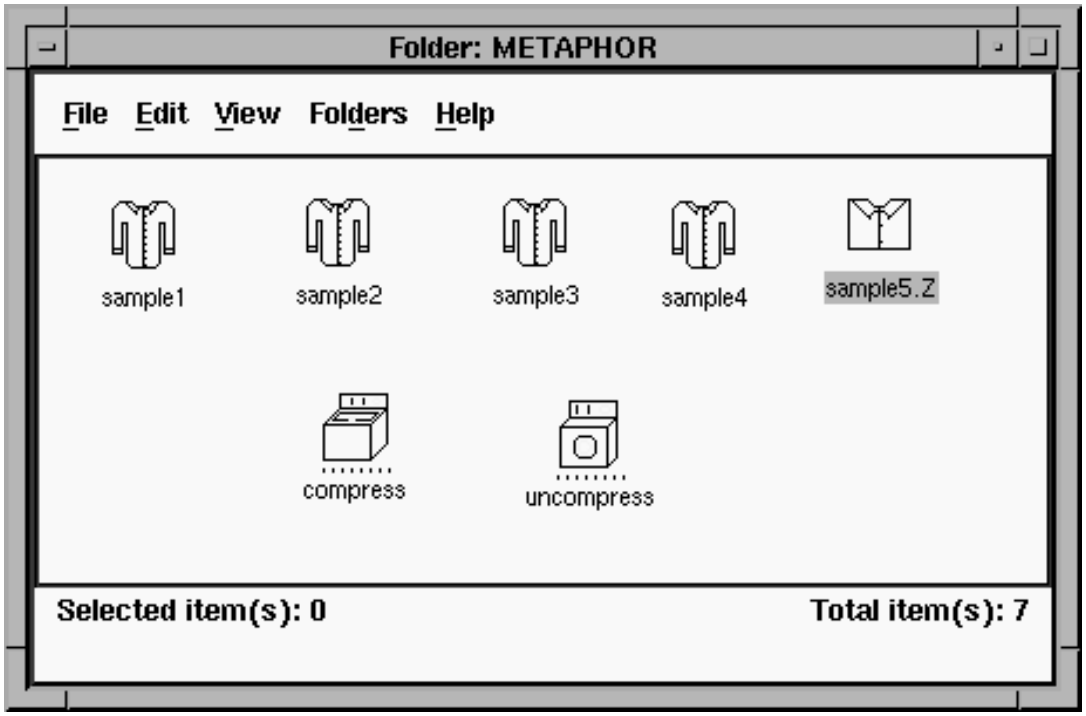
Assuming that the directory metaphor is the directory of interest containing the symbolic links to compress and uncompress, and has the files sample1, sample2, sample3, sample4 and sample5, opening a window for this folder will appear as shown in Figure 2-7.

Figure 2-7: The Folder: METAPHOR



Drag the file sample5 into the washer (compress) and drop it. This results in Figure 2-8.

Figure 2-8: Compressing a File



You can uncompress this file by dragging it to the dryer icon this time and then dropping it. This file should revert to its original icon.

A Note about Minimize and Close

For the desktop functions `Minimize` (MOTIF-mode) and `Close` (OPEN LOOK-mode), in order to get a process icon other than the system default for a particular process or folder, one needs to do the following:

- Create a 48x48 pixel version of the application (or folder) icon.
- Incorporate this icon in the **source** code using the `XtNiconPixmap` and `XtNiconName` resources for the application's toplevel shell.

Describing the File Class Database

The file class database is used to store file class descriptions. The file path to the file class database is specified by the desktop property FILEDB_PATH. Each description comprises a list of properties. The syntax for each description is:

```
CLASS name
BEGIN
        [attribute]      property_name   property_value;
        .
        .
END
```

where

name	is the name of the file class. It must be a valid C symbol.
attribute	are the attributes associated with a file class property.
property_name	is the name of the property.
property_value	is the value of the property.

Lines that begin with '#' (comments) and blank lines are ignored. Note that once the class database is updated by the desktop manager, comments and blank lines in the original file may be lost.

Five class properties are used to qualify files of a particular file class. They are PATTERN, FILETYPE, FILEPATH, LPATTERN and LFILEPATH. If all these properties are defined for a file class, a file must have all these properties to be considered in that file class. A file class without using any of these properties will define all files.

PATTERN is a simple regular expression that matches the name of a file. Special characters are '*', '?', '[', '!', and ']'.

FILETYPE is the type of the file. Possible values are: DIR, FIFO, EXEC, CHRDEV, BLKDEV, DATA, SEM, SHDATA and UNK. See the section ''Determining the File Class of an Icon'' for a brief description of each.

FILEPATH is a simple regular expression that matches the full path of the file, excluding the base name portion.

There is a list of predefined properties with special meanings. See the section ''File Class Properties'' for more detail.

The following is a list of attributes:

MENU	Use the property name as the button label in the icon menu. The prefix '_' of the property name will not be part of the button label. When the user selects the button, the property value is interpreted as a shell command.

EXAMPLE:

```
CLASS dashboard
BEGIN
      _PATTERN     "dashboard";
      _ICONFILE    "dash.xpm";
      MENU _Open   "exec '%F'&";
END
```

```
CLASS olpixmap
BEGIN
      _PATTERN     "olpixmap";
      _FILETYPE    EXEC;
      _ICONFILE    "x.xpm";
      MENU _Open   "exec %F &";
      _DROP        "exec %F %S &";
END

CLASS csource
BEGIN
      _PATTERN     "*.c";       # this is a comment
      _ICONFILE    "csource.xpm";
      MENU _Open   "exec xterm -e ${EDITOR} %F &";
      MENU _Compile"exec xterm -e ${CC} %F &";
END
```

A class database file can include another class database file as follows:

```
INCLUDE system;
INCLUDE dtadmin;

CLASS datafile
BEGIN
     _FILETYPE       "DATA";
     _ICONFILE       "datafile.icon";
     MENU _Open      "exec dtedit %F &";
     MENU _Print     "exec /usr/X/bin/PrtMgr -p %_DEFAULT_PRINTER %F &";
END

INCLUDE system.post;
```

`INCLUDE` is used to include another file. If the specified file is a relative path, then `XtResolvePathName` is called to find the file based on the default search path. Because locale specific paths are part of the default search path, applications that need to internationalize strings in the class database should use relative paths. The type specified for the `XtResolvePathName` call is ''classdb.'' Thus, locale specific files are stored in `/usr/X/lib/locale/%L/classdb`, where `%L` is the current locale. Locale independent files can be placed in `/usr/X/lib/classdb` directory. The nesting level limit is 16.

NOTE

You may define your own properties. However, these properties will not be interpreted by the desktop manager. User-defined properties should not have the prefix ''_.'' Names having this prefix are reserved for system-defined properties.

Determining the File Class of an Icon

There is a built-in list of internal file classes that are used to set the default file class of a file based on the information returned from `stat(2)` system call. The list contains the following file class entries:

`DIR`	a directory
`DATA`	a data file
`EXEC`	an executable file, a file is of this type if it has an execute permission set for owner, group and/or others
`FIFO`	a pipe file
`CHRDEV`	a character device
`BLKDEV`	a block device
`SEM`	a semaphore file (a XENIX file type)
`SHDATA`	a shared memory file (a XENIX file type)
`UNK`	none of the above

The following is the sequence of operations to determine the file class for a given file:

- Assign the built-in unknown file class. This is the ultimate default for files that the desktop manager knows nothing about.
- Go through the list of internal file classes to determine the file type.
- Go through the list of file classes from the file database in the order they are read in, and use the file class qualifiers (see the section ''File Class Properties'') to check if the file belongs to each file class. If a match is found, then the file class is determined.

Properties

Properties are used throughout the UNIX desktop as a method of relating an arbitrary list of attributes with an entity. There are three types of properties: desktop properties, file class properties, and file instance properties. Some property values allow variable substitutions. There are two types of variables. One type is environment variables. They are represented as $*name* or ${*name*}. The other type is property values. They are represented as %*name* or %{*name*}. Recursive substitutions are possible.

NOTE

File instance properties are stored in `.dtinfo` in each directory. Instance properties are not fully supported in the initial release of UNIX desktop.

The precedence for substituting these property variables are instance property, class property, and then desktop property. Certain properties are only recognized in a specific context. If no such properties are defined, an empty string is substituted. For example, if `%COMMENT` is defined as a class property as well as an instance property, then a query on the `%COMMENT` value will return the instance property value.

Other predefined variables are:

%F Full path name of the file.

%f Base name of the file.

%S Full path name of the source file that was dropped on the icon. This is usable only with the `DROP` property value.

%s Base name of the source file that was dropped on the icon. This is usable only with the `DROP` property value.

All property names prefixed with ''_'' and all single letter property names are reserved for future use by the desktop manager.

Desktop Properties

Desktop properties are global to the desktop. Some of the standard desktop properties are:

`DESKTOPDIR`	The directory that represents user's home window. By default, it is `$HOME`.
`WBDIR`	The wastebasket directory path. The default is `$HOME/wastebasket`.
`FILEDB_PATH`	The file path of the file class database. The default is `$HOME/.dtfclass`.
`HDPATH`	The Help Desk database file. It is set to `/usr/X/desktop/Help_Desk`.
`TEMPLATEDIR`	The default directory where templates for file classes reside.

File Class Properties

This section lists the standard file class properties.

_CLASSNAME	Provides a way to specify a label for a file class. This label is used in the File Property Sheet and Icon Setup instead of what is defined by CLASS. This is used to localize a class name or to provide a more descriptive name since spaces can be used.
_DFLTICONFILE	This is used only if the property _ICONFILE is defined and has variable substitutions in it. This is the default icon file to use in case the _ICONFILE value, after substituting variables, is not an existing icon file name. Naturally, the value of this property must not have variable substitutions.
_DROP	This is the shell command to invoke when the icon is being dropped on by another icon. The value may have variable substitutions. Special variables %S and %s can be used here. The special syntax %{S.*pname*} or %{s.*pname*} provides flexibility in specifying a value for the _DROP property. *pname* is a property name associated with the object being dropped (source). When *pname* is expanded, the name of the source object (derived from %s or %S) is used in the expansion of any %f or %F variables found in the definition of the property *pname*. For example, suppose the _DROP property for the printer class is defined to be %{s._Print} and the _Print property for the datafile class of the source object foo is defined as exec lp %f. Drag-and-drop of foo to the printer icon will cause the _DROP property for the printer to be expanded as foo._Print. When this property is evaluated, by looking up the definition of the _Print property associated with foo, foo._Print will be expanded to exec lp foo. The evaluation of this property would then cause foo to be sent to the printer.
_FILEPATH	This is a file class qualifier used to determine what files belong to the class. Only files found in this path are considered.
_FILETYPE	This is a file class qualifier used to determine what files belong to this class. Only files of this type as determined by the stat(2) system call are considered.
_ICONFILE	This specifies the name of the icon file. If a simple name is specified, /usr/X/lib/pixmaps is assumed. Variable substitutions are allowed in its property value.
_ICONLABEL	If this property is specified, the desktop uses its value to identify the icon instead of the name of the file or object represented by the icon.
_LFILEPATH	This is a file class qualifier used to determine what symbolically link files belong to the class. Only links that point to a file whose path matches this entry are considered.
_LPATTERN	This is a file class qualifier used to determine what symbolically linked files belong to this class. Only links that point to a file whose name matches this pattern are considered.

_Open	This is the shell command for opening the icon. The value may have variable substitutions. The name is in lower case because this property has the MENU attribute set and thus will appear in the icon menu.
_PATTERN	This is a file class qualifier used to determine what files belong to this class. Only files with names that match this pattern are considered. It can be a comma separated list of shell regular expressions (see sh(1)).
_SYSTEM	If the first character of the property value is a ''y'' or ''Y,'' then the object is considered a system object and cannot be deleted.
_TEMPLATE	This is the name of one or a comma-separated list of template files. Template files are used to fill in the content of a newly created file. With the exception of the DIR and DATA file classes, only files with one or more templates can be created using the New menu option in a folder's File menu. The default location for template files is /usr/X/lib/template.

Property names are not restricted to what is in the above list. Users or applications can use any other property names in the description, but they will not have any standard semantics to the desktop manager.

The Icon Definition File

All icons provided with the UNIX desktop are defined by entries in the File Class Database (`$HOME/.dtfclass`). When the desktop manager opens a folder, it attempts to match each of the objects in the folder against entries in the database. If the desktop manager cannot match an object with a particular class, it matches it with a class entry that is more generic, such as a standard file or an executable.

When installing a package (that is in System V Release 4 `pkgadd` format), the PackageMgr (`Application Setup` icon) first looks for the presence of an Icon Definition File. This file includes a variety of Desktop-related information that includes the following:

- a list of help files that should be ''registered'' with the Help Desk. These help files appear as icons in the Help Desk.
- a list of icons that should be presented to the user at the time of package installation. Application icons are displayed in the `Application Setup` ''Show Contents'' window so that a user can selectively install one or more icons to a folder window.
- new file-class entries that should be added to the File Class Database. Note that each object/icon in the Database has a variety of properties associated with it, including an *Open* action (`_Open`) and a *Drag-and-Drop* action (`_DROP`). See the section ''Describing the File Class Database'' found in this chapter.

Without an Icon Definition File, `Application Setup` presents the user with all of the files that are installed by the package. Since these files are likely to be foreign to the File Class Database, `Application Setup` must use generic file and/or executable icons to represent these files. The user may use the Icon Setup facility of UNIX desktop to build entries that better represent the types of files that are being installed.

For example, since the *Drop* action associated with a (generic) executable icon is ''exec %F %S,'' where `%S` is the object that is being dropped, this may not be sufficient for any given executable. The user would then have to use the Icon Setup to modify the invocation string.

However, rather than putting the burden on the end-users, packages that are targeted for the Desktop environment should be updated so that their packaging scripts include an Icon Definition File. The following section details the format of this file.

Icon Definition File Location

All Icon Definition files should be installed under `/usr/X/desktop/PackageMgr/`*pkgname* where *pkgname* is defined by `pkgadd`.

Icon Definition File Syntax

The Icon Definition File serves 3 purposes:

- to list those help files that should be added to the Help Desk,
- to indicate which icons should be registered with UNIX desktop, and
- to define new File Class entries.

Help File Specifications

The following syntax pertains to Help File entries:

HELP=*HelpPath*[:*Locale*] <tab> *IconFile*[:*Locale*] <tab> *Label* <tab> *AppName*

If *HelpPath* is a full path, then *Locale* is ignored. If *HelpPath* is a relative path and *Locale* is specified, the PackageMgr constructs the full path of the help file as follows:

/usr/X/lib/locale/*Locale*/help/*PkgName/HelpPath*

If *HelpPath* is a relative path and *Locale* is not specified, the PackageMgr assumes the C locale Thus, the full path of the help file is:

/usr/X/lib/locale/C/help/*PkgName/HelpPath*.

The *IconFile* field tells the PackageMgr the name of the pixmap/bitmap file to register with the Help Desk. If *IconFile* is a full path, then *Locale* is ignored. If *IconFile* is a relative path, then *Locale* is used to construct the full path of the iconfile. Finally, if *Locale* is not specified for a relative path, then the ''C'' locale is assumed. The pixmaps/bitmaps should be installed under the following directories:

/usr/X/lib/locale/*Locale*/pixmaps
: location of the pixmap files for *Locale*.

/usr/X/lib/locale/*Locale*/pixmasks
: location of the ''masks'' to use with the pixmap files in the pixmaps sub-directory.

/usr/X/lib/locale/*Locale*/bitmaps
: location of the bitmap files for *Locale*; to be used as cursors or icon bitmaps on monochrome displays.

/usr/X/lib/locale/*Locale*/bitmasks
: location of the ''masks'' to use with the bitmap files in the bitmaps sub-directory.

For each icon that is registered with the Help Desk, it is expected that an application installs both a pixmap and a bitmap file, though the corresponding mask files are optional.

NOTE

It is important that all pixmap, bitmap, and mask files for a given icon have the same file name.

Label is the string to be displayed under the icon in the Help_Desk. This is only used as a default if the Help File does not specify the label field.

The *AppName* field is an internationalized string of the form *AppString[CatalogFile:Index]*. If *AppName* is specified, it is used by the HelpMgr to construct the title of the corresponding Help Window. If *AppName* is not specified, then *PkgName* (from pkgadd) is used.

Icon Specifications

Icon entries should appear in the following format:

ICON=*IconFile*[:*Locale*] <tab> *IconLabel* <tab> *ObjectType*

The location of *IconFile* depends on the (optional) *Locale* suffix. If *IconFile* is a full path, then *Locale* is ignored. If *IconFile* is a relative path, then *Locale* is used to construct the full path of the iconfile. Finally, if *Locale* is not specified for a relative path, then the ''C'' locale is assumed. Once again, all pixmaps/bitmaps that define an icon specification should appear in the following directories:

`/usr/X/lib/locale/`*Locale*`/pixmaps`
: location of the pixmap files for *Locale*.

`/usr/X/lib/locale/`*Locale*`/pixmasks`
: location of the ''masks'' to use with the pixmap files in the `pixmaps` sub-directory.

`/usr/X/lib/locale/`*Locale*`/bitmaps`
: location of the bitmap files for *Locale*; to be used as cursors or icon bitmaps on monochrome displays.

`/usr/X/lib/locale/`*Locale*`/bitmasks`
: location of the ''masks'' to use with the bitmap files in the `bitmaps` sub-directory.

NOTE Icons that do not have locale variants can go in the `/usr/X/lib/pixmaps`, `/usr/X/lib/pixmasks`, `/usr/X/lib/bitmaps`, `/usr/X/lib/bitmasks` directories as illustrated earlier in this chapter.

For each icon that is listed, it is expected that an application installs both a pixmap and a bitmap file, though the corresponding mask files are optional. It is important that all pixmap, bitmap, and mask files associated with a given icon have the same file name.

The *IconLabel* field is an internationalized field of the form *LabelString[CatalogFile:Index]*. This indicates the label to use for the corresponding icon. For icons representing directories (folders), this field corresponds to the directory name.

Finally, the *ObjType* field indicates the type of object that the icon represents. The syntax for the *ObjType* field is "*type=value*, where *type* can be `FILE` or `DIR`. The *value* associated with an object of type `FILE` is the full path of the object. For example, if a package defines an icon for an executable called ''`spreadsheet`,'' the *ObjType* field might appear as `FILE=/usr/appdir/bin/spreadsheet`. `DIR` does not accept a *value* and takes the directory name specified by the *IconLabel* field.

File Class Specifications

All File Class specifications should appear in the following format:

CLASS=*ClassFile*[:*Locale*]

If *Locale* is not present, `ApplicationSetup` assumes that the location of the Class File (whose entries are to be added to the File Class Database) is:

`/usr/X/lib/locale/C/classdb/`*PkgName/ClassFile*`.`

If *Locale* is specified, then the class file should be located in the following directory (where *PkgName* is

extracted from *pkgadd*):

`/usr/X/lib/locale/`*Locale*`/classdb/`*PkgName*`/`*ClassFile*`.`

ClassFile is the name of the file class database you have provided.

NOTE

Any reserved property names (those beginning with "_" such as `_Print` and `_Open`) cannot be localized. It is the responsibility of the desktop manager to display the localized version of the name if necessary.

Example

Consider an application whose *PkgName* is `ZCalc`, and whose package format is `pkgadd`. Assume also that this application is written for the "C" locale only, has only one icon that can be installed and has a single help file to be registered with the Help Desk. Given this scenario, the Icon Definition File for `ZCalc` might appear as follows:

```
HELP=ZHelpFile <tab> ZCalc.icon <tab> ZCalc
ICON=ZCalc.icon <tab> ZCalc <tab> FILE=/usr/xyz/bin/ZCalc
CLASS=ClassFile
```

The package installation script for `ZCalc` must perform the following tasks:

1. install `ZHelpFile` under `/usr/X/lib/locale/C/help/ZCalc`.
2. install both a pixmap and a bitmap file, each named `ZCalc.icon`, under `/usr/X/lib/locale/C/pixmaps` and `/usr/X/lib/locale/C/bitmaps`, respectively.
3. (optionally) install a pixmap mask and a bitmap mask file, each named `ZCalc.icon`, under `/usr/X/lib/locale/C/pixmasks` and `/usr/X/lib/locale/C/bitmasks`, respectively.
4. install an executable file named `ZCalc` under `/usr/xyz/bin`.
5. install `ClassFile` under `/usr/X/lib/locale/C/classdb/ZCalc`. A file class entry corresponding to `ZCalc` must appear in the file class database so that it is typed properly by the desktop manager.

   The contents of the class file appear as follows:

   ```
   CLASS Z_calc
   BEGIN
         _PATTERN     "ZCalc";
         _FILETYPE    EXEC;
         _ICONFILE    ZCalc.icon;
         MENU Open    "exec %F &";
   END
   ```

6. install the Icon Definition File under `/usr/X/desktop/PackageMgr`. Assuming that the package name is `Zcalc`, the icon definition file should also be called `ZCalc`.
7. If the `ZCalc` executable is to be executed by double clicking on the icon `Zcalc` from an `Applications` folder, the postinstall script should have the following code:

```
pkginfo -q desktop >/dev/null 2>&1
if [ $? = 0 ]
then
   ln -s /usr/xyz/bin/ZCalc /usr/X/desktop/Applications/ZCalc
   if [ -d $HOME/Applications ]
   then
           ln -s /usr/xyz/bin/ZCalc $HOME/Applications/ZCalc
   fi
```