# Programming the IRIS WorkSpace



IRIS-4D Series

Document number: 007-2006-010

# Programming the IRIS WorkSpace

Version 1.0

Document Number 007-2006-010

#### **Technical Publications:**

Robert Reimann Claudia Lohnes

#### Engineering:

John Eisenman Bruce Karsh Eva Manolis Rob Myers Betsy Zeller

#### © Copyright 1989, Silicon Graphics, Inc. - All rights reserved

This document contains proprietary information of Silicon Graphics, Inc. The contents of this document may not be disclosed to third parties, copied or duplicated in any form, in whole or in part, without the prior written permission of Silicon Graphics, Inc.

## **Restricted Rights Legend**

Use, duplication or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013, and/or in similar or successor clauses in the FAR, or the DOD or NASA FAR Supplement. Unpublished rights reserved under the Copyright Laws of the United States. Contractor/manufacturer is Silicon Graphics Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.

Programming the IRIS WorkSpace Version 1.0 Document Number 007-2006-010

Silicon Graphics, Inc. Mountain View, California

IRIS, IRIS, IRIS Graphics Library, and WorkSpace are trademarks of Silicon Graphics, Inc. PostScript is a registered trademark of Adobe Systems, Inc.

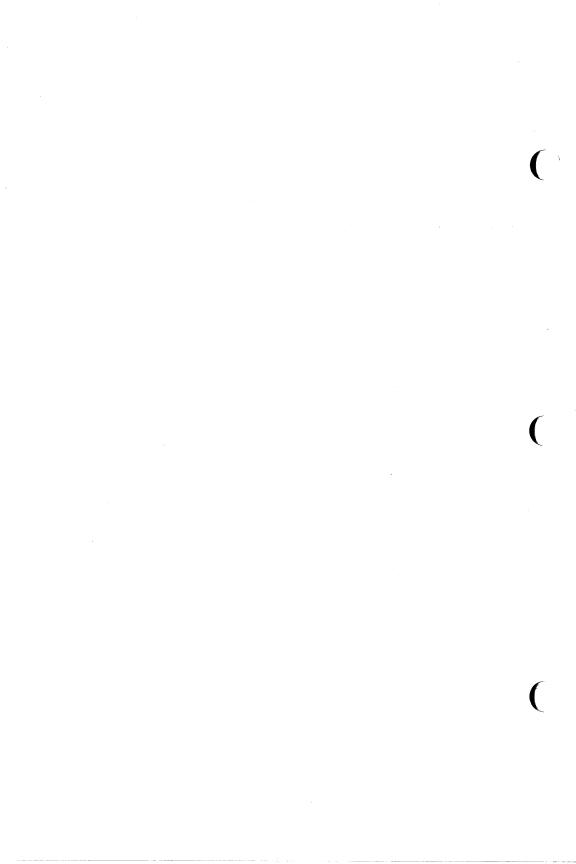
## Contents

1. Intr	oduction	•	•		•	•	1-1
1.1	What Does WorkSpace Do?	•	•	•	٠	•	1-1
1.2	What Is File Typing?		•	٠		•	1-1
1.3	FTR Files		•		•	•	1-2
1.4	WorkSpace Templating				•		1-2
	Transfer Devices						1-2
2. Wri	ting File Typing Rules	•		•	•		2-1
	The TYPE Declaration						2-3
	The MATCH Rule						2-3
	2.2.1 Valid Match-Expressions						2-4
	2.2.2 Building Effective MATCH Rules						2-6
	2.2.3 Tagging Executables						2-6
	2.2.4 Using Scripts without Tagging						2-9
2.3	The LEGEND Rule						2-9
2.4							2-10
2.5	The SPECIALFILE Rule						2-10
	The Command (CMD) Rules						2-11
	2.6.1 The CMD OPEN Rule						2-11
	2.6.2 The CMD ALTOPEN Rule						2-12
	2.6.3 The CMD DROP Rule						2-12
	2.6.4 The CMD PRINT Rule						2-13
2.7	The MENUCMD Rule						2-13
	The BOUNDS Rule						2-14
	The ICON Rule						2-15
	2.9.1 The Icon Description Language						2-15
	2.9.2 Drawing Icons						2-18
	2.9.3 Style Conventions for Workspace Icons						2-19
2.1	0 Compiling FTR Rules						
	2.10.1 Order of Precedence of FTR Files						
	2.10.2 Placement of FTR Rules						

3. Writing Print Conversion Rules	3-1	
3.1 The Print Conversion Pipeline	3-1	4
3.2 The CONVERT Rule	3-2	
3.3 The COST Rule	3-2	
3.4 The FILTER Rule	3-3	
3.5 Printer Types	3-4	
3.6 The Current Printer	3-5	
olo Tilo Guilorit Tilitor	00	
4. Creating WorkSpace Templates	4-1	
4.1 Using WorkSpace in Template Mode	4-1	
4.2 Setting Up an Application Environment	4-3	
5. Creating Transfer Devices	5-1	
5.1 The Tranfer Device Interface	5-1	
5.1.1 Transfer Device Typing	5-1	
5.1.2 Transfer Device I/O	5-2	_
5.1.3 Using WorkSpace Environment Variables	5-3	1
5.1.4 Local Transfer Devices	5-3	•
5.2 Example: shellDevice	5-3	
	• •	
A: WorkSpace Environment Variables	A-1	
R: WorkSnace Man Pages	R-1	
D: WOLKSONCE WAN PAGES	H-1	

## **List of Tables**

Table 2-1.	File Typing Rules	•	•	•	•	•	•	•	2-2
Table 2-2.	Numerical Representations in Match Expressions						•	•	2-4
Table 2-3.	Match-Expression Functions	•	•	•	•	•	•	•	2-5
Table 2-4.	Tag Numbers for IRIX Executables		•	•	•		•	•	2-7
Table 2-5.	Icon Description Functions						•	•	2-17
Table B-1.	WorkSpace Man Page Summary							•	В-



# **List of Figures**

Figure 2-1.	Mapping Icons to Pixels Using BOUNDS	2-14
Figure 2-2.	Splitting a Concave Polygon	2-21
Figure 2-3.	3-D Icon Axes	2-21
Figure 2-4.	GenericExecutable Icon	2-22



## 1. Introduction

This manual describes how to program the appearance and functionality of file icons in the IRIS™ WorkSpace™ environment. This manual should enable you to customize existing icons, as well as add your own unique icons to the IRIS WorkSpace.

## 1.1 What Does WorkSpace Do?

The IRIS WorkSpace presents a sophisticated icon-based interface to the IRIX<sup>TM</sup> file system and operating system. Icons in the IRIS WorkSpace represent files; the various ways that you can act upon these icons, such as double-clicking on them with the mouse cursor, invoke IRIX command lines as if they had been typed in a shell. Thus, the WorkSpace allows you to perform sophisticated operations on IRIX files entirely through its graphical interface.

## 1.2 What Is File Typing?

The IRIS WorkSpace manages the numerous varieties of IRIX files and the actions that are associated with them using a process called *file typing*. There a potentially unlimited number of possible file types that can exist within the IRIX file system: plain text files, formatted documents, directories, shell scripts, images, and binary executables, to name only a few. Every type of file has an associated set of operations, often unique, that a user would most often want to perform on the files. The file type declarations and associated rules that give each type of file a unique appearance and behavior under the WorkSpace are collectively called *file* 

Version 1.0 Introduction 1-1

typing rules. The syntax of these rules is discussed in Chapter 2. An additional set of rules define the preferred method of sending a file of a particular type to a user-specified printer. These rules are called *print conversion rules*, and are discussed in Chapter 3.

## 1.3 FTR Files

File typing rules determine how files behave within the IRIS WorkSpace. Print conversion rules determine how file types defined by the typing rules are converted to printed output. Both kinds of rules are read by WorkSpace from compiled versions of special text files called *FTR files*. An FTR source file ends with the suffix *ftr*, and must reside in one of four subdirectories found in */usr/lib/filetype*. Section 2.10, "Compiling FTR Rules", discusses which of these subdirectories is appropriate for your FTR files, and how to compile them into versions that can be understood by WorkSpace.

# 1.4 WorkSpace Templating

WorkSpace templating allows you to create configurations of icons in the WorkSpace window that will appear for all WorkSpace users on a system. WorkSpace templates are maintained in /usr/lib/workspace; each template installed in this directory has an additive effect on existing templates, so a user of several software packages written for use with WorkSpace can have the benefit of each package's template. WorkSpace templating is described in Chapter 4.

## 1.5 Transfer Devices

This manual also describes how to create transfer devices. Transfer devices are special shell scripts that move data into and out of a directory, and are accessed as commands under the "Transfer" submenu in the WorkSpace and Directory View menus. End users can select transfer devices by selecting the Transfer Manager from the System toolchest. Possible transfer

devices include scripts that archive selected files to and from tape, copy selected files to and from remote machines, or mail selected files to other users. Transfer devices can move data locally or over the network, depending on how they are implemented. They are discussed in detail in Chapter 5.

Version 1.0 Introduction 1-3



# 2. Writing File Typing Rules

File typing rules determine how a file of a particular type will appear within the IRIS WorkSpace, and also define what functions you can perform on the file by double-clicking on it or choosing menu items that manipulate it. The IRIS WorkSpace uses the file typing rules to evaluate all files that are presented within the WorkSpace or any Directory View window.

There are several reasons why you might want to write file typing rules. You might wish to customize the look of WorkSpace icons or modify what happens to the files they represent when you manipulate them. Or, you might want to create unique WorkSpace icons when you are developing an application and its associated data files. In either case, you need to write file typing rules to accomplish your goal.

File typing rules are similar in some ways to shell scripts. In fact, several of the rules are simply sets of Bourne shell commands (see sh(1)). File typing rules consist of a type declaration, which identifies a unique file type, and a set of up to seven rules following the declaration. All rules, including the type declaration, consist of a *rule key* followed by the rule itself. Rules can be multi-line, but continuation lines cannot begin with any of the rule keys. Table 2-1 below gives a brief summary of the rule keys and what each associated rule is used for.

The remaining sections of this chapter describe each of the file typing rules in detail, and offer suggestions for good file typing style and strategies.

File Typing Rule Key	Function
TYPE	Declares a new type.
MATCH	Lets WorkSpace determine if a file is of the declared type.
LEGEND	Provides a text description of the file type.
SUPERTYPE	Tells WorkSpace to treat the file as a subset of a another type under certain circumstances.
SPECIALFILE	Tells WorkSpace that the file typing rule is to be used only on non-plain files.
CMD OPEN	Defines a series of actions that occur when the mouse is double-clicked on an icon.
CMD ALTOPEN	Defines a series of actions that occur when the mouse is alt-double-clicked on an icon.
CMD DROP	Defines a series of actions that occur when you "drop" an icon on top of another.
CMD PRINT	Defines a series of actions that occur when you choose "Print" from the WorkSpace or Directory View menus.
MENUCMD	Defines menu entries and actions that are inserted into the WorkSpace or Directory View menus when an icon is selected.
BOUNDS	Defines the coordinate space for the file type's icon.
ICON	Defines the appearance (geometry) of the file type's icon.

Table 2-1. File Typing Rules

## 2.1 The TYPE Declaration

Syntax: TYPE type-name

**Description:** type-name is a one word ASCII string. Legal type names can

be any legal C language variable name. You should ideally choose a name that is in some way descriptive of the file type it represents. All rules that follow a TYPE declaration apply to that type, until the next TYPE declaration is encountered in the FTR file. Each TYPE declaration must have a unique

type name.

Example: TYPE GenericExecutable

## 2.2 The MATCH Rule

**Syntax:** MATCH match-expression;

**Description:** match-expression is a logical expression that should evaluate

to true if and only if a file is of the type declared by TYPE. The match-expression must consist only of valid MATCH functions, as described Section 2.2.1. The match-expression can use multiple lines, but must terminate with a semicolon (;). Multiple match-expressions are not permitted for a given type. The MATCH rule is employed each time a file is encountered by the WorkSpace, to assign a type to that file.

Example: MATCH glob("\*.h") && ascii;

## 2.2.1 Valid Match-Expressions

This section describes the syntax and function of valid match-expressions.

## Operators, Constants, and Numerical Representation

The following C language operators can be used in a match-expression:

The following C language conditional operators can be used in a matchexpression:

The '==' operator works for string comparisons in addition to numerical comparisons.

The following constants can be used in a match-expression:

#### true false

Numbers in match-expressions can be represented in decimal, octal, or hexadecimal notation. Table 2-2 describes these representations.

Representation	Syntax
decimal	num
octal	0num
hexadecimal	0xnum

Table 2-2. Numerical Representations in Match-Expressions

## **Functions**

Table 2-3 describes the set of valid match-expression functions.

Function Syntax	Definition
ascii	Returns TRUE if the first 512 bytes of the file are all printable ASCII characters.
char(n)	Returns the $n$ th byte in the file as a signed character; range $-128$ to $127$ .
dircontains ("string")	Returns TRUE if the file is a directory and contains the file named by <i>string</i> .
glob("string")	Returns TRUE if the file's name matches <i>string</i> ; allows use of the following expansions in <i>string</i> for pattern matching: $\{\}\ [\ ] * ?$ and backslash (see $sh(1)$ file name expansion).
linkcount	Returns the number of hard links to the file.
long(n)	Returns the <i>n</i> th byte in the file as a signed long integer; range $-2^{31}$ to $2^{31} - 1$ .
mode	Returns the mode bits of the file (see $chmod(1)$ ).
<pre>print (expr or "string")</pre>	Prints the value of the expression <i>expr</i> or <i>string</i> to <i>stdout</i> each time the rule is evaluated; used for debugging. Always returns true.
short(n)	Returns the $n$ th byte of the file as a signed short integer; range $-32768$ to $32767$ .
size	Returns the size of the file in bytes.
string(n,m)	Returns a string from the file that is $m$ bytes (characters) long, beginning at the $n$ th byte of the file.
uchar (n)	Returns the <i>n</i> th byte of the file as an unsigned character; range 0 to 255.
tag	Returns the specific WorkSpace application tag injected into an executable file by the tag injection tool (see <i>tag</i> (1) in Appendix B, "WorkSpace Man Pages"). Returns –1 if the file is not a tagged file.
ushort(n)	Returns the <i>n</i> th byte of the file as an unsigned short integer; range 0 to 65535.

Table 2-3. Match-Expression Functions

## 2.2.2 Building Effective MATCH Rules

A match rule consists of a sequence of expressions, each of which checks a file for positive distinguishing characteristics. The most effective way to order these expressions in a single MATCH rule is to choose a set of expressions, each of which tests for a single characteristic, and conjoin them all using "and" conditionals (&&).

The order in which the expressions in a MATCH rule are conjoined may have an effect on the efficiency of the rule's evaluation. You should always try to order the expressions so that the maximum number of files are "weeded out" by the first expressions. The reason for this is that, as in the C language, && will stop evaluation as soon as one side of the conditional is found to be false. Therefore, as a rule of thumb, the more likely an expression is to be false, the further to the left of the MATCH rule you should place it.

For example, one possible way to match a C source file is with the following rule:

```
MATCH glob("*.c") && ascii;
```

Note that it is more efficient to place the glob("\*.c") expression first because there are many more files that do not end in c than there are files that are not ASCII text.

You should also make sure that your match rule is specific enough not to "catch" any unwanted files. FTR files are scanned sequentially (see Section 2.10, "Compiling FTR Rules"), so if you define a type with the following MATCH rule at the beginning of the FTR file,

```
TYPE foo
MATCH ascii;
```

every text file in your system will be defined as a file of type "foo".

## 2.2.3 Tagging Executables

The preferred way to match a specific executable to a file typing rule is to "tag" it with a unique 32-bit number. tag(1) allows you to inject a 32-bit tag safely into a shell script or MIPS executable, where it can be read by a MATCH rule using the tag match-expression function (see Table 2-2).

The upper 16 bits of the tag number are reserved for vendor ID, and are administered by Silicon Graphics, Inc. The lower 16 bits are for general use. Anyone planning to create file typing rules as part of a commercial software release is strongly encouraged to obtain a vendor ID.

Number	Semantics
0x00000000	windowed executable (no tty window needed)
0x0000100	generate a launch window for argument input
0x00000200	generate an output-only tty window
0x00000400	generate a tty window for input and output
0x0000000	no arguments
0x0000001	one argument
0x00000002	two arguments
0x0000004	three arguments
0x0000008	zero or one argument
0x0000010	zero or more arguments
0x00000020	one or more arguments

Table 2-4. Tag Numbers for IRIX Executables

Silicon Graphics, Inc., has defined a tag numbering scheme and a set of TYPE rules for standard IRIX executables that make use of these 32-bit values. Table 2-4 lists the various tag numbers and what actions they anticipate. These numbers are combined via bitwise-OR to create the tag numbers used in the standard IRIX executable TYPEs listed below with their associated MATCH rules.

```
TYPE GenericWindowedExecutable
   MATCH     tag == 0x00000000;

TYPE ttyOut1-NargExecutable
   MATCH     tag == 0x00000620;

TYPE ttyLaunchExecutable
   MATCH     tag == 0x00000500;

TYPE ttyOutExecutable
   MATCH     (tag == 0x00000600) || (tag == 0x00000610);

TYPE ttyLaunchOutExecutable
   MATCH     (tag == 0x00000700) || (tag == 0x00000710);

TYPE ttyLaunchOut1-NargExecutable
   MATCH     tag == 0x00000720;
```

```
TYPE ttyExecutable
                (tag == 0x00000400) \mid | (tag == 0x00000410);
    MATCH
TYPE LaunchExecutable
    MATCH
                tag == 0x00000100;
TYPE ttyLaunch1-NargExecutable
    MATCH
              tag == 0x00000520;
TYPE ttyOut2argExecutable
    MATCH
               tag == 0x00000602;
TYPE Generic1-NargExecutable
    MATCH
                tag == 0x00000020;
TYPE ttyLaunchOutlargExecutable
                tag == 0x00000701;
TYPE tty1-NargExecutable
    MATCH
                tag == 0x00000420;
TYPE GenericlargExecutable
   MATCH
                tag == 0x00000001;
TYPE ttyOutlargExecutable
    MATCH
                tag == 0x00000601;
TYPE ttyLaunchOut3argExecutable
                tag == 0x00000703;
TYPE tty2argExecutable
   MATCH
               tag == 0x00000402;
TYPE Generic2argExecutable
    MATCH
                tag == 0x00000002;
TYPE ttyLaunchOut2argExecutable
                tag == 0x00000702;
TYPE Generic3argExecutable
                tag == 0x00000003;
TYPE ttyLaunchlargExecutable
    MATCH
                tag == 0x00000501;
TYPE Launch1-NargExecutable
    MATCH
                tag == 0x00000120;
TYPE Launch2argExecutable
               tag == 0x00000102;
TYPE LaunchlargExecutable
   MATCH
```

tag == 0x00000101;

Software developers are free to make use of these predefined TYPEs for IRIX utilities that do not require a personalized look. For applications and other executables for which a personalized icon is desired, you should create your own tag numbers (taking care not to use any of those predefined by Silicon Graphics, Inc.) and appropriate MATCH and ICON rules. Other rules can be copied from appropriate TYPEs in the list above.

See the *tag*(1) man page in Appendix B, "WorkSpace Man Pages", for more information on tagging executables.

## 2.2.4 Using Scripts without Tagging

If you have a generic shell script that you would like to run in a tty window, but don't want to create a file typing rule for it, you can do so by making the second line of the script look as follows:

#### #winterm

You can also use any of the available flags to *winterm* (see *winterm*(1) in Appendix B) on this line.

## 2.3 The LEGEND Rule

Syntax: LEGEND text-string

**Description:** text-string is a string that describes the file type in plain

language a user can understand. The legend is used to describe the file in the Get File Info window. It is also used when a Directory View window is set to display as a list. Legends that are longer than 25 characters might be truncated

in some circumstances.

Example: LEGEND C program source file

## 2.4 The SUPERTYPE Rule

Syntax: SUPERTYPE type-name [type-name ... ]

Description: The SUPERTYPE rule is used to indicate that a particular

file type can be treated as a part of a more general file type under certain conditions. A special case in WorkSpace is directories; you might wish to create a directory with a custom icon, yet still have the OPEN and ALTOPEN rules work as a normal directory. (Note: WorkSpace automatically handles the OPEN and ALTOPEN rules for directories: the OPEN and ALTOPEN rules are place holders.). You could create your own directory TYPE with its own ICON rule, but if you use SUPERTYPE, it will work like a standard directory (see example). SUPERTYPE is also very useful for printing, where you might want to print a custom file type as, say, an ASCII file. You can also make use of SUPERTYPEs in your own OPEN, ALTOPEN, DROP, and PRINT rules by using isSuper(1) as part of those rules. See the isSuper(1)man page in Appendix B, "WorkSpace Man Pages". A given file typing rule may contain several different SUPERTYPE rules, and thus be considered a subset of several more general file types. The SUPERTYPE rule does not reflect true object-oriented typing, and hence, does not allow inheritance of rules; it is more a way of aliasing

Example:

TYPE MyDirectory

TYPEs.

SUPERTYPE Directory

## 2.5 The SPECIALFILE Rule

Syntax: SPECIALFILE

**Description:** SPECIALFILE is used to distinguish a file typing rule used

for matching non-plain files. Device files, and other nonplain files can cause damage to physical devices if they are matched using standard file typing rules. Special files are matched using only rules containing SPECIALFILE, which are written so as not to interfere with actual physical devices. Similarly, plain files are not matched using rules containing a SPECIALFILE rule.

Example:

SPECIALFILE

# 2.6 The Command (CMD) Rules

The CMD rules determine how an icon behaves when a user interacts with it, whether it is by clicking, dragging, or through menu selections.

#### 2.6.1 The CMD OPEN Rule

Syntax:

CMD OPEN sh-expression[; sh-expression; ...; sh-

expression]

**Description:** The OPEN rule is invoked when any file of the appropriate type is double-clicked, or selected and opened from the WorkSpace or Directory View menu via the "Open" menu item. The OPEN rule should reflect the most often used function that would be applied to a file of the given type. sh-expression can be any valid Bourne shell expression. Any

> expression can use multiple lines. Any number of expressions can be used, and must be separated by

semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix A, "WorkSpace Environment Variables", for a list of special

environment variables set by the WorkSpace. These environment variables can be used to refer to the currently selected icons within the WorkSpace or Directory View.

Example:

\$WINEDITOR \$FILES CMD OPEN

## 2.6.2 The CMD ALTOPEN Rule

Syntax:

CMD ALTOPEN sh-expression[; sh-expression; ...; sh-

expression]

Description:

The ALTOPEN rule is invoked when any file of the appropriate type is double-clicked while the ALT key is pressed. The ALTOPEN rule provides added functionality for power users. *sh-expression* can be any valid Bourne shell expression. Any expression can use multiple lines. Any number of expressions can be used, and must be separated by semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix A, "WorkSpace Environment Variables" for a list of special environment variables set by WorkSpace. These

environment variables can be used to refer to the currently selected icons within the WorkSpace or Directory View.

Example:

CMD ALTOPEN make

## 2.6.3 The CMD DROP Rule

Syntax:

CMD DROP sh-expression[; sh-expression; ...; sh-

expression]

**Description:** 

The DROP rule is invoked whenever a selected (file) icon is "dropped" onto another icon in the WorkSpace or Directory View windows. When this happens, WorkSpace checks to see if the file type that is being dropped upon has a DROP rule to handle the files being dropped. In this way, you can write rules that allow one icon to process the contents of other icons simply by dragging selected icons that you want processed on top of the target icon (i.e., the one with the

DROP rule).

Example:

2-12

CMD DROP STARGET SSELECTED

#### 2.6.4 The CMD PRINT Rule

**Syntax:** CMD PRINT sh-expression[; sh-expression; ...; sh-

expression]

Description: The PRINT rule is invoked whenever a file of the appropriate

type is selected and printed using the "Print" menu item from the WorkSpace or Directory View menu. sh-expression can be any valid Bourne shell expression. Any expression can use multiple lines. Any number of expressions can be used, and must be separated by semicolons (;). The final expression should not end with a semicolon. Variables can be defined and used as in a Bourne shell script, including environment variables. See Appendix A, "WorkSpace Environment Variables", for a list of special environment variables set by WorkSpace. These environment variables can be used to refer to the currently selected icons within the WorkSpace or Directory View. The recommended method of implementing the PRINT rule is to use the WorkSpace's print-job routing utility, *routeprint*. See the *routeprint*(1) man page in Appendix B, "WorkSpace Man Pages", for details on its syntax.

Example: CMD PRINT routeprint \$LEADER \$REST

## 2.7 The MENUCMD Rule

Syntax: MENUCMD "string" sh-expression[; sh-expression; ...; sh-

expression]

Description: MENUCMD inserts the menu entry string into the

WorkSpace or Directory View menu if a single file of the appropriate type is selected, or a group all of the same appropriate type is selected. If the menu entry is chosen, the actions described by the *sh-expression*s are performed on

each of the selected files.

Example: MENUCMD "Empty Dumpster" \rm -rf \$LEADER/\*

## 2.8 The BOUNDS Rule

Syntax: BOUNDS x0, y0, x1, y1

**Description:** x0, y0, x1, y1 define, respectively, the lower left and upper

right corners of the bounding rectangle of the coordinate space in which the icon is displayed. The values are separated by commas (,). When the WorkSpace paints the icon, it scales the icon so that its bounds fit just within the fixed layout area reserved for it. The aspect ratio of the bounding rectangle is preserved. If no BOUNDS rule is supplied for a file type's icon, the bounding rectangle

defaults to 0.0, 0.0, 100.0, 100.0.

Example: BOUNDS -20.0, -20.0, 50.0, 75.0

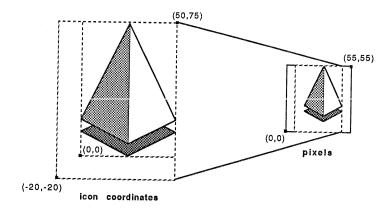


Figure 2-1. Mapping Icons to Pixels Using BOUNDS

2-14

## 2.9 The ICON Rule

Syntax: ICON icon-description-routine

**Description:** *icon-description-routine* is a routine written using the icon

description language, detailed below. The routine can continue for any number of lines. The ICON rule is invoked any time a file of the specified type needs to be represented in the WorkSpace or a Directory View. The rule is evaluated each time the icon is painted by the application that needs it.

## 2.9.1 The Icon Description Language

The icon description language is a restricted subset of the C programming language, including line and polygon drawing routines from the IRIS Graphics Library<sup>TM</sup>. The description routine for a given icon is similar in structure to a C subroutine, but lacks the subroutine and variable declarations and the outermost enclosing brackets. The valid symbols and functions in the icon description language are described below.

## **Operators**

The following C language operators can be used in an icon description routine:

```
+ - * / & | ^ ! \% = () {}
```

The following C language conditional operators can be used in an icon description routine:

```
&& || == != < > <= >=
```

#### Constants

The following logical constants can be used in an icon description routine:

#### true false

The following icon color constants can be used in an icon description routine:

#### iconcolor outlinecolor shadowcolor

Use of the icon color constants is described in Section 2.9.2, "Drawing Icons".

#### **Variables**

The following icon status variables are set by WorkSpace, and can be used in an icon description routine:

#### current disabled opened located selected

These variables have values of either **true** or **false**. They can be used in a conditional statement to alter the appearance of an icon when it has been manipulated in various ways from the WorkSpace (see Section 2.9.2, "Drawing Icons").

Other legal C variables can be used in an icon description routine without need of a declaration; all variables are represented as type float. Any variable name is acceptable, provided it does not collide with any of the predefined constants, variables, or function names in the icon description language.

## **Functions**

The icon description functions comprise, for the most part, a very restricted subset of the C language version of the IRIS Graphics Library, modified for 2-D drawing. Table 2-5 describes the set of valid icon description functions.

Function Syntax	Definition
arc(x, y, r, startang, endang)	Draw an arc starting at icon coordinates $x$ , $y$ , radius $r$ , starting at angle <i>startang</i> , ending at angle <i>endang</i> . Angle measures are in tenths of degrees.
arcf (x, y, r, startang, endang)	Like arc, but filled with the current pen color.
bclos(color)	Like pclos (see below) but uses <i>color</i> for the border (outline) color of the polygon.
bgnclosedline()	Begin drawing a closed, unfilled figure drawn in the current pen color. Used in conjunction with vertex and endclosedline.
bgnline()	Like bgnclosedline, except the figure is not closed. Used in conjunction with vertex and endline.
bgnoutlinepolygon	Begin drawing a polygon filled with the current pen color. The polygon is outlined with a color specified by endoutlinepolygon. Also used in conjuction with vertex.
bgnpoint()	Begin drawing a series of unconnected points defined using calls to vertex. Used in conjunction with vertex and endpoint.
bgnpolygon()	Like bgnoutlinepolygon except the polygon is not outlined. Used in conjunction with vertex and endpolygon.
color(n)	Set current pen color to color index $n$ .
draw(x, y)	Draw a line in the current color from the current pen location to $x$ , $y$ .
endclosedline()	Finish a closed, unfilled figure started with bgnclosedline.

Table 2-5. Icon Description Functions

Function Syntax	Definition
endline()	Finish an open, unfilled figure started with bgnline.
endoutlinepolygon(color)	Finish a filled polygon started with bgnoutlinepolygon and outline it with color.
endpoint()	Finish a series of points started with bgnpoint.
endpolygon()	Finish a filled, unoutlined polygon started with bgnpolygon.
for	Standard C for-loop.
if (expr) expr [ else expr ]	Standard C language if-statement.
move(x, y)	Move current pen location to $x$ , $y$ .
pclos()	Draw a line in the current pen color that closes the current polygon, and fill the polygon with the current color.
pdr(x, y)	Draw the side of a filled polygon in the current pen color, from the current pen location to x,y.
pmv(x, y)	Begin a filled polygon at location $x$ , $y$ .
<pre>print (expr or "string")</pre>	print the value of the expression <i>expr</i> or <i>string</i> to <i>stdout</i> ; used for debugging.
vertex(x,y)	Specify a coordinate used for drawing points, lines, and polygons by bgnpoint, bgnline, bgnpolygon, etc.

Table 2-5. (continued) Icon Description Functions

## 2.9.2 Drawing Icons

This section describes several concepts that should help you design your icons.

Any points, lines, or polygons you draw will "stack" in the order they are drawn, with the most recently drawn polygon on top. You can use this concept to easily achieve drop-shadow effects, by drawing the same polygon twice, using different pen colors, and offset.

Three icon color constants are recommended for standard icon use: iconcolor for drawing polygons that make up the "foreground" of the icon, outlinecolor for outlining and linework, and shadowcolor for contrasting drop shadows. iconcolor is particularly useful, because it automatically changes to the calling application's preferred color conventions when a given icon is located, disabled, or selected.

## 2.9.3 Style Conventions for Workspace Icons

The standard set of WorkSpace icons have been designed to establish a clear, predictable visual language for end users. As you extend the WorkSpace by adding your own application-specific icons, it is important to make sure that your extensions fit the overall look of the WorkSpace and operate in a manner consistent with the rest of the WorkSpace. The following set of conventions should be followed by developers creating new WorkSpace icons.

## **Using World Coordinate Space**

All WorkSpace icons developed by Silicon Graphics, Inc., lie completely within the default world coordinate BOUNDS, a square area 100 units on a side. Icons you design can lie within any convenient coordinate space, but it is crucial that all points in your icon lie completely within whatever BOUNDS you specify. Failure to follow this convention can result in icon clipping and other display anomalies.

In the default configuration of WorkSpace, any geometry defined for an icon within the specified BOUNDS region is mapped into a square screen region 55 pixels on a side. You should be aware that in other contexts, your icon might be mapped onto larger or smaller screen areas, the idea being to maintain resolution-independence, which allows an icon to be arbitrarily magnified or reduced. Keep in mind that in a 55x55 pixel area, fine details are not visible.

## **Using Icon Colors and Animation**

Use the **iconcolor**, **outlinecolor**, and **shadowcolor** as your icons' typical colors; that way, when the icons are selected or located, they will indicate that state in a manner consistent with the rest of the WorkSpace. Be sparing with the use of other accenting colors. This helps preserve the impact of color when it is needed.

Some WorkSpace icons animate when they are double-clicked; directory folders open, ASCII text files glow, and executables have the "flying carpet" whisked out from under them. If you wish to animate your icon when it is opened, you can easily do this by having a set of geometry for the open state defined within a conditional:

Since a redraw of each icon's bounding area is done only when it changes between its closed and opened states, you should refrain from animating an icon for anything other than the "opened" state.

## **Drawing Hints**

When designing your icons, do not use concave polygons; they are not currently supported by the icon description language, and using them will yield unpredictable results. If your icon does not display as you planned, check for concave polygons. You will need to break any such polygons into two or more convex polygons (see Figure 2-2). In addition, no single polygon can contain more than 255 vertices.

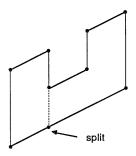


Figure 2-2. Splitting a Concave Polygon

To help keep icon descriptions simple, you are encouraged to use bclos (outlinecolor) when possible, in place of a pclos() filled area outlined by further calls to move() and draw().

bclos (outlinecolor) automatically draws outlined polygons, thus shortening icon code and improving performance. Remember to comment your icon code generously, in case you want to alter parts of it later.

## Keeping the 3-D Look

Icons created by Silicon Graphics, Inc., were designed in an isometric space, which provides an illusion of 3D, even though the polygons composing the icons are 2-D. To generate this effect, draw "horizontal" lines so that they move up 1 unit in the y-axis for every 2 units they extend along the true x-axis (see Figure 2-3).

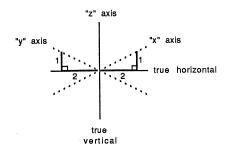


Figure 2-3. 3-D Icon Axes

## **Designing Icons for Applications**

If you are designing an icon for an application, use the ICON rule for GenericExecutable as a base to extend from, adding iconography to the basic "flying carpet" look. The icon used by *jot* is a good example of such an extension. You can find the ICON rule for GenericExecutable in the file */usr/lib/filetype/default/sgidefault.ftr*. See Figure 2-4 for a representation of the GenericExecutable icon. Similarly, your application's data files should be modeled using the ICON rule for AsciiTextFile as a base. This rule can also be found in the file *sgidefault.ftr*.



Figure 2-4. GenericExecutable Icon

## 2.10 Compiling FTR Rules

New FTR rules must be compiled from the FTR source files located in a set of directories in /usr/lib/filetype. Any time you add or change FTR rules (or print conversion rules) within these subdirectories, you must recompile the complete set of ftr files. This is done by performing the following command line sequence:

su cd /usr/lib/filetype make

2-22

**Note:** When distributing and installing your application and its FTR files on a new machine, be sure your installation process copies the FTR files into /usr/lib/filetype/install and then remakes the FTR file set using the procedure described above.

Be sure to quit any currently running version of WorkSpace, and restart it afterwards to activate the new FTR rules.

#### 2.10.1 Order of Precedence of FTR Files

FTR source files in the following four directories are compiled in the order listed here:

- 1. /usr/lib/filetype/local
- 2. /usr/lib/filetype/install
- 3. /usr/lib/filetype/system
- 4. /usr/lib/filetype/default

Since compiled rules are scanned sequentially by WorkSpace, a TYPE defined in *local* will override any subsequently defined TYPE with an identical type-name. Care should be taken so as not to so override important system or default TYPE declarations.

Within each directory, separate FTR source files are compiled alphabetically.

## 2.10.2 Placement of FTR Rules

The *default* and *system* directories in */usr/lib/filetype* are reserved for systemwide standards and maintained by Silicon Graphics, Inc. Developers and users should not place their *.ftr* files in these directories.

The *install* directory should be used by applications developers and site maintainers for integrating their extensions. The standard naming convention for application vendors' file typing rules is:

vendor-name[.application-name].ftr

The *local* directory may be used by power end-users for personal customizations.

.*			

# 3. Writing Print Conversion Rules

Print conversion rules are similar in structure to file typing rules, and in fact both can occupy the same FTR files. While file typing rules declare and define valid types of files, print conversion rules provide a means of generating command pipelines that process a file for printing. Using *routeprint*(1) in a file type's CMD PRINT rule (see Section 2.6.4, "The CMD PRINT Rule") causes WorkSpace to scan each existing FTR file in its rule path for a set of one or more rules that, when strung together will result in a printed copy of the file. If the rules are written properly, the end result is a printed copy of the file.

# 3.1 The Print Conversion Pipeline

Sometimes, converting a file to printable form may take more than one step. For example, an *nroff* file must first be converted to a PostScript® file before it can be printed by a laser printer. You could write a single rule that would print your *nroff* file directly, but since PostScript files can originate from sources other than *nroff*, you would want instead to write two separate rules—one that converts from *nroff* to PostScript format, and one that converts from PostScript to the printed file. This idea of multi-step conversion to printable form is called the *print conversion pipeline*. The print conversion rules are designed to take advantage of this method of processing printable files.

Note that when a file is processed along the print conversion pipeline, the original file remains intact. The print conversion pipeline is really just a series of IRIX commands that process a copy of the file's data in modular increments.

# 3.2 The CONVERT Rule

Syntax: CONVERT source-type-name destination-type-name

**Description:** source-type-name is the file type you are converting from.

destination-type-name is the file type you are converting to. If the CONVERT rule is the last in the print conversion pipeline (i.e., the rule that actually converts the file to printed form), then destination-type-name should be one of the printer types defined in the FTR rules (see Section 3.5, "Printer Types"). All print conversion rules following a CONVERT rule apply to that conversion, until another

CONVERT rule is encountered.

Example: CONVERT NroffFile PostScriptFile

# 3.3 The COST Rule

Syntax: COST non-negative-integer

**Description:** non-negative-integer represents the arc cost, or incremental

cost of the conversion. This is an abstract notion that can be used to determine the preferred printer for a particular document. This rule is primarily for use in a network in which there are many different printers of various quality. When *routeprint* selects a printer, it takes the arc costs into account, choosing the print conversion pipeline (see Section 3.1 "The Print Conversion Pipeline") with the least total cost. The COST rule is optional; if it is omitted, the cost of

the conversion is assumed to be zero.

Example: cost 1

# 3.4 The FILTER Rule

Syntax: FILTER filter-expression

**Description:** The FILTER rule represents part of an IRIX pipeline that

prepares a file for printing. *filter-expression* can be any single IRIX command line, and generally takes the form of a number of piped commands. In the general case, the first command within a single FILTER rule receives input from *stdin*; the last command in the rule send its output to *stdout*. *routeprint* concatenates all the FILTER rules in the print conversion pipeline to form one continuous command that sends the selected file to its destination printer.

There are three special cases in creating FILTER rules:

- first case
- last case
- setvar case

In the first case, the FILTER rule is the very first rule in the print conversion pipeline. In this case, *routeprint* passes the list of selected files to the first command in the FILTER rule as arguments. If a first case FILTER rule begins with a command that does not accept the files in this fashion, you should prepend the *cat* command to your rule:

FILTER cat | tbl - | psroff -d\$CURRENTPRINTER

The files will then be piped to the next command's *stdin*.

In the last case, the FILTER rule is the very last rule in the print conversion pipeline. At the end of this rule will be a command that sends output to a printer (such as lp). routeprint sets the environment variable \$CURRENTPRINTER to the currently selected printer. Use this variable in FILTER rules that require a print destination (see example below). See Section 3.6, "The Current Printer" for more information on how \$CURRENTPRINTER is set.

Note that a one-step conversion pipeline is an example of both a first case and a last case FILTER rule.

In the setvar case, the FILTER rule is used to set an environment variable used later in the print conversion pipeline. The first CONVERT rule in the example below sets a variable that defines an *nroff* macro used in the second rule. In all setvar cases, *stdin* is passed to *stdout* transparently. Thus, you can include setvar as part of the pipeline in a single FILTER rule.

## Example:

CONVERT mmNroffFile NroffFIle COST

FILTER setvar MACRO=mm

CONVERT NroffFile PostScriptPrinter COST

FILTER eqn | tbl | psroff -\$MACRO -d\$CURRENTPRINTER

# 3.5 Printer Types

Three printer types are currently supported by the IRIS system. They are specified in the FTR file /usr/lib/filetype/sytem/sgisystem.ftr:

TYPE ColorPrinter

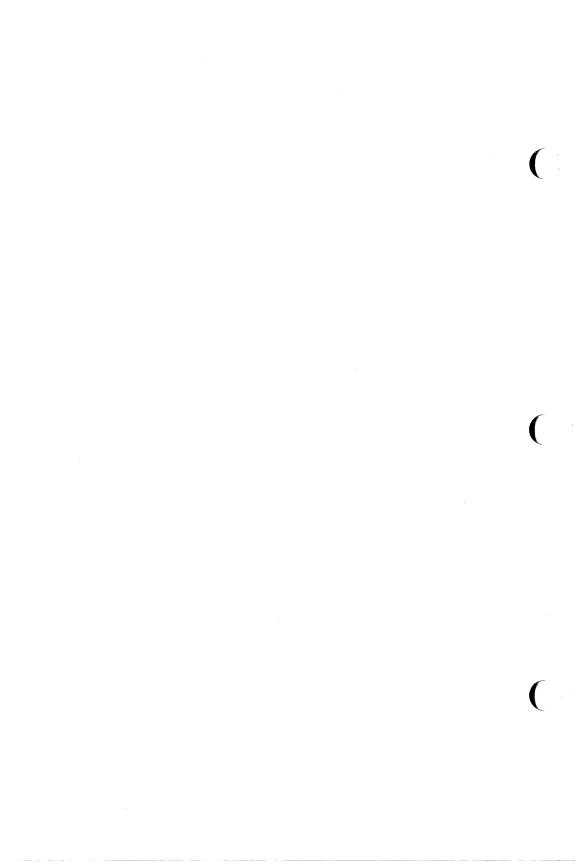
TYPE PostScriptPrinter

TYPE DumbPrinter

In most cases you will want to use either PostScriptPrinter (for text or images) or ColorPrinter (for images only) in your CONVERT rules. DumbPrinter supports standard line printer text-only output.

# 3.6 The Current Printer

The current printer is the system default printer set by the user with the Print Manager, or, alternatively, with the  $-\mathbf{p}$  option to *routeprint*. If no default is set and  $-\mathbf{p}$  is not used, an error message will be returned by *routeprint* to either stdout or a notifer window (if the  $-\mathbf{g}$  option to *routeprint* was set).



# 4. Creating WorkSpace Templates

WorkSpace templating provides a means of making sure that all the applications and/or files a user will need appear in the WorkSpace window whenever WorkSpace is started up. It also provides a way of limiting naive users' access to sensitive directories by "locking" those directories out of the WorkSpace.

WorkSpace templates are set interactively by invoking a special flag to workspace from the shell.

# 4.1 Using WorkSpace in Template Mode

To use the WorkSpace in template mode, you must first quit WorkSpace if you are using it. Open a shell window, and type the following commands:

su workspace -t

You may optionally specify a name after the -t option (see *workspace*(1)). This name determines the name of the template file; by default it is set to your login name (*\$USER*). The name option is useful for developers who might want to identify their WorkSpace templates by the name of their product or company.

You should run WorkSpace in template mode when logged in as a typical user in order to get a template that represents an environment in which a typical user would work. You should not run WorkSpace in template mode when logged in as "root", for example, because its home directory is in an unusual location. Note that you must, however, invoke root privileges to run WorkSpace in template mode, but in this case, the original login environment is preserved by WorkSpace.

When WorkSpace comes up, it will look identical to your non-template mode WorkSpace. The only difference is the addition of four items to the WorkSpace menu:

- Save Template
- Get Root
- Set Lock On
- · Set Lock Off

These functions are explained below.

# **Save Template**

The Save Template menu item saves your WorkSpace template to a file with the following format:

name.wsrc

name is either the name you specified when invoking WorkSpace from the command line (described above), or \$USER by default. When you use the Save Template command, WorkSpace will first attempt to write the template file to the directory /usr/lib/workspace. If you do not have permission to write to that directory, WorkSpace will offer you the option of creating that file in your own WorkSpace directory. In this case, you will need to install your new template file in /usr/lib/workspace before other users can access it. If more than one WorkSpace template exists in this directory, WorkSpace will use all of the templates, in an additive manner. Developers should plan on installing their templates into this directory on a user's machine as part of their installation process.

# **Get Root**

Under normal circumstances, the root (/) directory cannot be removed from the WorkSpace. WorkSpace in template mode allows you to remove the root directory and limit access from the WorkSpace to other directories (see "Set Lock On/Off", below). The Get Root menu item lets you retrieve the root directory and place it back in the WorkSpace window once it has been put away.

## Set Lock On/Off

The Set Lock On and Set Lock Off menu entries allow you to limit the user's access to the IRIX file system from the WorkSpace. When a directory is locked using the Set Lock On menu item, a user cannot access any file or directory that is in a parent directory, or is itself a parent directory of the locked directory, unless it is already in the WorkSpace window. To lock a directory, select it and choose the Set Lock On item from the WorkSpace menu.

The Set Lock Off menu item unlocks a locked directory. Locked directories cannot be put away from the WorkSpace using the Put Away menu item; they must first be unlocked. The root (/) directory is, by default, locked; all other directories are initially unlocked. If you unlock and put away root, you should choose another directory to be the head of the subtree, and set a lock on it. If you choose to lock a directory other than root (/), you must put away any ancestors of that directory before you save the template, or they will appear on the user's WorkSpace.

# 4.2 Setting Up an Application Environment

By using the four menu items described in the last section along with the standard workspace features, you can create a template for the WorkSpace window that provides your users with all the necessary icons (files) to use whatever application you are developing. You can place the application directly in the WorkSpace window so that your users will not have to rummage through directory views to find it. Likewise, you can place utilities and directories customized with your company logo there. Using the Set Lock... items and the Put Away item, you can limit naive-user access to the root directory and other directories that might contain data to which end users should not have direct access.

# 5. Creating Transfer Devices

Transfer devices are special executable shell scripts that allow users to import and export data from a directory. Transfer devices selected using the Transfer Manager tool install themselves in the WorkSpace and Directory View menus as items in the Transfer submenu. Once installed using the Transfer Manager, a transfer device is accessible from the WorkSpace or any Directory View menu.

# 5.1 The Tranfer Device Interface

Transfer Devices are shell scripts that communicate with the WorkSpace and the Transfer Manager through a simple set of protocols. They are installed by placing them in the directory /etc/transferDevice; any script in that directory that follows the protocol described below will appear as an icon in the Transfer Manager.

# 5.1.1 Transfer Device Typing

The second line of a transfer device script (the line after the shell invocation) must have the following form:

#transferDevname

name can be any name addition to the "transferDev" prefix. The complete name (transferDevname) should correspond to a TYPE declaration in an FTR file. The Transfer Manager scans the set of file typing rules for this TYPE, and uses the associated ICON rule to draw the transfer device's icon within the Transfer Manager window.

When creating new transfer devices, you should write your own FTR rules, using the rules for standard shipped transfer devices as templates. If you do not do so, a generic transfer device icon will be assigned to the transfer device.

## 5.1.2 Transfer Device I/O

A transfer device must be able to handle the following strings as arguments.

- menu
- versionsOK

The meaning of these strings, and the output that is associated with them, is explained in the following two sections.

#### menu

5-2

The transfer device is called with this argument by both the WorkSpace and the Transfer Manager. The transfer device must respond by echoing one or more character strings in the following format:

```
echo "token1 menu-item-string1" echo "token2 menu-item-string2"
```

menu-item-string is the text of the menu item that you wish to appear in the Transfer submenu of the WorkSpace and Directory View menus. These items will only appear once the transfer device as been set by the user with the Transfer Manager. A transfer device may insert any number of menu items.

token is a one-word, unique token that WorkSpace will use as an argument in calling the transfer device when its associated menu item is chosen from the Transfer submenu. Thus, your transfer device must also accept each token string as an argument, and respond with the appropriate action described in its associated menu item.

## versionsOK

When a user tries to set your transfer device using the Transfer Manager, the device is called with "versionsOK" as an argument. Your device must echo one of the following strings:

```
echo "local"
echo "remote"
echo "remote local"
```

If it echoes the first string, the Transfer Manager will allow only the local version of the device to be set. If it echoes the second string, only remote versions of the device can be set. If it echoes the last string, either version can be set.

# 5.1.3 Using WorkSpace Environment Variables

Currently, only the WorkSpace environment variable *\$SELECTED* is supported for use within transfer devices. This variable contains the list of selected icons from the WorkSpace. See Appendix A, "WorkSpace Environment Variables", for more information.

# 5.1.4 Local Transfer Devices

You can make a transfer device available to a particular user, but not all users by placing the transfer device in the user's \$HOME/.workspace/localTransferLinks directory. It will appear in the Transfer Manager window just like any other transfer device, but only when the window system is started from that particular user's login.

# 5.2 Example: shellDevice

The following is an example transfer device that allows you to open a shell window on any selected directory. Note that this transfer device inserts only one menu item in the Transfer submenu. Transfer devices can insert any number of items into the Transfer submenu.

## Make sure to set the device's execute permissions.

```
#!/bin/sh
#transferDevShell
 User Preference Variables:
    # true or false
    openMultiple=false
case $1
   in
        "menu")
           echo "open Open a Shell Window"
        "versionsOK")
            echo "local"
        "open")
            if [ -z "$SELECTED" ]; then
                # Workspace puts us in the correct directory.
                title='pwd'
                winterm -t $title
            else
                # Check that the user has made a valid
                # number of selections.
                set -- $SELECTED
                if [ $openMultiple = false -a $# -gt 1 ]; then
                    inform "Too many selections; cannot open shell."
                    exit 1
                fi
                # Go to the proper directory and open the window.
                for selection in $SELECTED; do
                    if [ -d $selection ]; then
                        cd $selection
                    else
                        dir='dirname $selection'
                       cd $dir
                    fi
                    title='pwd'
                    winterm -t $title
                done
            fi
        ;;
esac
exit 0
```

# Appendix A: WorkSpace Environment Variables

The following is a list of environment variables used by WorkSpace. Any of these variables can be used as part of the OPEN, ALTOPEN, or PRINT file typing rules, or as part of the FILTER print conversion rule.

\$LEADER If one or more icons are currently selected from

the WorkSpace, *LEADER* is set to the icon whose text field is highlighted. If no icons are

selected, it is set to null.

\$REST If more than one icon is currently selected from

the WorkSpace, *REST* contains the list of names of all selected icons *except* the highlighted icon (see *LEADER* above). Otherwise, it is set to

null.

\$LEADERTYPE If one or more icons are currently selected from

the WorkSpace, *LEADERTYPE* is set to the TYPE of the icon whose text field is highlighted.

If no icons are selected, it is set to null.

\$RESTTYPE When more than one icon is currently selected

from the WorkSpace, RESTTYPE contains the

TYPE for all selected icons except the

highlighted icon, if the remainder of the selected icons are all the same TYPE. If they are not the

same TYPE, or only one icon is selected,

RESTTYPE is set to null.

\$RESTTYPELIST Contains the list of TYPEs corresponding to the

arguments in REST. If only one icon is selected,

RESTTYPELIST is set to null.

*\$ARGC* Contains the number of selected icons.

*\$TARGET* Set only for the CMD DROP rule, *TARGET* 

contains the name of the icon being dropped

upon; otherwise it is set to null.

*\$TARGETTYPE* Set only for the CMD DROP rule,

TARGETTYPE contains the TYPE of the icon being dropped upon; otherwise it is set to null.

*\$SELECTED* Contains the names of the icons being dropped

on TARGET, or null, if none are being dropped.

\$SELECTEDTYPE If all the icons named in SELECTED are of the

same TYPE, SELECTEDTYPE contains that

TYPE; otherwise it is set to null.

*\$SELECTEDTYPELIST* Contains a list of TYPEs corresponding to the

TYPEs of the selected icons named in

SELECTED. If only one icon is selected, it is set

to null.

\$WINEDITOR Contains the name for the text editor invoked

from WorkSpace. The default editor is *jot*. To use an editor that does not generate its own window by default, you must set *WINEDITOR* to the appropriate *winterm* command line sequence.

Thus, for vi, you would set WINEDITOR as

follows:

setenv WINEDITOR 'winterm -c vi'

*\$WINTERM* Contains the name of the window terminal

invoked from WorkSpace using *winterm*(1). Currently supported window terminals are *wsh*, *psterm*, and *xterm*. The default window terminal

is wsh.

A-2

# **Appendix B: WorkSpace Man Pages**

The following table lists the IRIX man pages found in this manual.

Man Page	Synopsis			
confirm(1G)	displays a message in a window with a choice of responses			
dirview(1G)	graphical interface to a directory			
inform(1G)	displays a message in a window			
isSuper(1)	checks if a given TYPE is of a given SUPERTYPE			
jot(1G)	a simple mouse-based editor; the default editor accessible from WorkSpace			
launch(1)	puts up a window that prompts to complete the command line of an executable			
routeprint(1)	routes files to printers using WorkSpace print conversion rules			
tag(1)	tags a MIPS executable with a magic number; for use in file typing			
winterm(1)	helps launch applications that require a terminal emulator			
workspace(1G)	the WorkSpace visual interface			

Table B-1. WorkSpace Man Page Summary

confirm - display a message in a window and request a response

#### **SYNOPSIS**

```
confirm [ -b button-name ... ] [ -t "title-string" ... ]
```

#### DESCRIPTION

confirm displays a window containing a line of test for each  $-\mathbf{t}$  argument specified, and a button for each  $-\mathbf{b}$  argument specified. When one of the buttons is pressed, the label of that button is written to confirm's standard output. This allows shell scripts to ask questions.

#### **EXAMPLE**

The following shell script will display a window, asking the user a yes or no question.

```
#! /bin/sh
case `confirm -t "Really power down the computer?" -b No -b Yes`
in
    Yes) shutdown ;;
    No) ;;
esac
```

#### BUGS

There can be at most three lines of titles specified with  $-\mathbf{t}$  and three buttons specified with  $-\mathbf{b}$ .

The window which appears is fixed size, and the text is not wrapped, so a lengthy message can be truncated.

The buttons are of fixed size, so lengthy button titles will not fit on the buttons.

#### SEE ALSO

inform(1G)

dirview – graphical interface to file system

#### **SYNOPSIS**

dirview pathnamesdirview -o sourcename targetname

#### DESCRIPTION

dirview is a tool which allows quick access to a directory view not currently open on the WorkSpace. When invoked with a directory path name, it will request an existing WorkSpace process to open a directory view of that directory. If no WorkSpace process exists for that user, dirview will start one.

dirview accepts the following options.

## pathname [pathname...]

If *dirview* is run with one or more directory pathnames as arguments, a view will be opened for each of those directories.

#### -o sourcename targetname

If a view is open for *sourcename*, replace it with a view of *targetname*. *sourcename* and *targetname* can be either full or relative path names.

#### SEE ALSO

workspace(1G)

inform – display a message in a window

#### SYNOPSIS

inform [text]

#### DESCRIPTION

*inform* opens up a window on the graphics console containing the *text* and a Continue button. If the mouse is clicked in the Continue button, the window will go away.

#### BUGS

The window which appears is fixed size, and the text is not wrapped, so a lengthy message can be truncated.

#### SEE ALSO

confirm(1G)

isSuper – supertype checking utility for use with file type rules

#### SYNOPSIS

isSuper supertype testtype [file.ctr]

#### DESCRIPTION

isSuper is used to check if testtype has a supertype of type supertype defined in the .ctr file file.ctr or a default of /usr/lib/filetype/workspace.ctr.

### SEE ALSO

jot – a simple mouse-based text editor

SYNOPSIS

jot [ -f fontname] [files...]

#### DESCRIPTION

*jot* is a simple editor that uses the mouse to cut, copy, and paste text, and to position the cursor. *jot* also allows you to perform simple searches.

If you specify the  $-\mathbf{f}$  option followed by a font specification, jot will use the specified font to display the text. For example:

jot -f Courier14

opens a jot window with a 14-point Courier font.

If you specify a set of text *files* separated by spaces on the command line, jot will open that file for editing; otherwise it will open a new file.

The jot menu provides the following facilities:

Cut Remove selected text to the cut buffer. This text can be sent to a wsh window or to another jot window, as well as the window it was cut from. This function can also be accessed using the F2 function key on the keyboard.

Copy Copy selected text to the cut buffer. This text can be sent to a *wsh* window or to another *jot* window, as well as the window it was copied from. This function can also be accessed using the F3 function key on the keyboard.

Paste Transfer text from the cut buffer to the current text cursor location (the *current point*). If text is selected when the paste is made, the selected text is replaced with the text from the buffer. This text could have been cut/copied from any *jot* or *wsh* window. This function can also be accessed using the F4 function key on the keyboard.

Search Search the document for the first instance of a string. Choosing 'Search' generates a notifier window that requests a string. *jot* searches sequentially from the current point through the rest of the file for that string, selecting the first instance it comes across. To find the next instance, you must choose 'Select' again.

Open Open a text file. Choosing 'Open' generates a notifier window that requests a file name. *jot* opens a new window with the new file in it, keeping the old one open as well.

#### SELECTING TEXT

Select text with the left mouse button by performing the following actions:

click Sets the new current point to wherever the

mouse (arrow) cursor is pointing in the

text.

click-hold-release This action selects all text between the the

point where the left mouse button is first clicked down and the point where it is

released.

click-click Double-clicking the left mouse causes the

word over which the mouse cursor is clicked to be selected. Double-clicking in the margin, or past the end of a line selects

the whole line.

shift&click-hold-release If you have already selected a block of text,

you can extend your selection either forwards or backwards by holding down the shift key while clicking the left mouse.

alt&click-hold-release Same as click-hold-release, except that an

implicit copy is performed when the left mouse button is released. You can use

alt&shift&click as well.

#### SCROLL BARS

jot uses horizontal and vertical scroll bars with proportional thumbs.

Clicking on the arrows with the left mouse button causes the *jot* window to scroll one character up, down, left, or right. Holding down the left button over any arrow causes the scrolling to auto-repeat. Holding down the shift key while clicking on any arrow causes the window to scroll one full page of text.

The scroll bars' thumbs can also be used to scroll through the *jot* window by clicking and holding down the left mouse button while dragging the thumb with the mouse cursor. The size of the thumbs are variable, and indicate the percentage of total text that is visible in the window.

#### **BUGS**

The horizontal scrollbar can only handle lines a maximum of 200 characters across.

#### SEE ALSO

wsh(1G)

launch – graphical utility to enter arguments and invoke commands

#### SYNOPSIS

launch [-h header] [-m message ] [-t trailer ] [-c command]

#### DESCRIPTION

*launch* is used to invoke commands through a window that contains a text edit field to allow for command completion. The options to launch are as follows:

-h header puts the header header on the command that is used to invoke the command but is not displayed.

-t trailer puts the trailer trailer on the command that is used to invoke the command but is not displayed.

-m message displays the message specified by message in the window.

-c command sets the command invoked to *command*.

When no arguments are specified, *launch* comes up with an empty text field waiting for input.

If the launched program requires a tty, you must call *launch* with the  $-\mathbf{h}$  option, followed by an appropriate invocation of *winterm*. The following is an example of launch as used with *mail*.

launch -h winterm -c mail

#### SEE ALSO

winterm(1)

route print – route file to printer

**SYNOPSIS** 

routeprint [-g] [-p printer] [-t type] files

#### DESCRIPTION

routeprint is a utility used by WorkSpace and accessible from the IRIX command line to route files of various types to a set of desired printers. routeprint uses file types specified on the command line to look up print conversion rules for each file to be printed. The conversion rules are located in compiled .ctr files in /usr/lib/filetype. The source .ftr files can be found in the local, install, system, and default subdirectories is /usr/lib/filetype. If no file types are specified on the command line, routeprint looks up the appropriate type for each file. routeprint uses the print conversion rules to process the files into a form printable by the target printer.

printer is the name of a printer to which the output may be sent.

type is a file-type name.

files is one or more file names, separated by spaces.

The  $-\mathbf{g}$  option should be used when routprint is defined as part of a file typing rule. This option puts error messages in a notifier window (instead of sending them to stdout) and supresses warnings.

The  $-\mathbf{p}$  or  $-\mathbf{t}$  options may appear multiple times on the command line, and are used in the following way:

- printer is added to the collection of printers on which the output -p may appear. Each instance of the -p option on the command line adds one printer to this collection. If more than one printer is specified, routeprint uses the print conversion rules to determine the best printer to use. If no printer names are given via the  $-\mathbf{p}$ flag, the destination printer is the system default printer. Using the  $-\mathbf{p}$  option overrules the system default printer.
- type sets the file-type for the files that follow it on the command -tline until another type is specified. If no type is given via the -tflag, or files appear on the command line before the first  $-\mathbf{t}$ , the files are typed by routeprint. (routeprint does not currently support the use of multiple file-types.) routeprint examines all of the specified files' types. If they are identical, a single print job will be initiated. If the types are varied, routeprint generates an error message.

The system default printer is the printer or printer class on which a print job appears if no printer is specified with the -p option. The system default printer is normally specified using the Print Manager in the System toolchest.

#### USAGE

A typical call from WorkSpace would be from a .ftr file type rule entry such as:

PRINT routeprint -t \$ARGTYPE \$FIRSTFILE \$RESTFILES

A typical call from the command line might look like the following:

routeprint -p myprinter file1 file2 file3

#### JOB ORDERING

The ordering of files handed to *routeprint* determines the ordering of files within the resultant print job. The ordering of files handed to routeprint from WorkSpace is constructed in the following manner:

- If an icon is selected individually, its name is appended to the current pending selection list.
- If an area selection is made, each of the icons within that area selection is added to the pending selection list in geographic order, left-to-right, top-to-bottom.

### PRINT CONVERSION RULES

The .ftr file used by routeprint contains both file type rules and print conversion rules.

The following is a typical set of print conversion rules:

CONVERT troff text postscript COST 1 FILTER psroff -t \$file

CONVERT postscript mylaserprintertype COST 1 FILTER lp -d \$CURRENTPRINTER

The CONVERT item specifies the file type of the input file followed by the file type of the converted file.

The COST item specifies an arbitrary number between 0 and 100 (inclusive) that represents the image degradation in printing. The higher the COST value, the more *routeprint* will try to avoid printing by that specific conversion method, if it is given a choice.

The FILTER item contains the shell command that performs the conversion.

Given the conversion rules above, the command:

```
routeprint -p mylaserprinter -t troff_text myfile.troff
```

would cause the file mytroff.t to be printed on the printer named "mylaser-printer" via the *psroff* and *lp* commands. Note that more than one conversion rule may be used to actually get the files into a printable form.

#### **FILES**

/usr/lib/filetype/local/\*.ftr /usr/lib/filetype/install/\*.ftr /usr/lib/filetype/system/\*.ftr /usr/lib/filetype/default/\*.ftr \$HOME/.workspace/print

#### SEE ALSO

tag – tag a MIPS executable or shell script with an identifying number

#### **SYNOPSIS**

tag number filename[s]

tag -c filename[s]

tag filename

tag -q filenames

#### DESCRIPTION

tag is used to set, clear or query the tag number in a MIPS executable, or shell script that follows the convention of #!/bin/sh or #!/bin/csh on the first line. The tag number is used by the SGI WorkSpace to determine the type of a file.

tag number filename[s] sets the tag number of a MIPS executable or

script. ( Many executables or scripts can be specified to be tagged with the same number.) The number must be non-negative and less than

prints out the tag number of a MIPS executable

4294967296.

tag -c filename[s] clears the tags on the specified file or files.

tag filename

or script.

prints out the tag numbers of a list of MIPS exetag –q filenames

cutables or scripts.

Tag numbers are administered by the Silicon Graphics user interface group. Contact them in order to get a block of tag numbers.

The tag number is stored as a longword in bytes 68 through 71 (numbering from zero) of the MIPS executable. The most significant bit of byte 18 is set when the file has been tag and will not be set otherwise. For shell scripts, the line '#Tag <number>' will be inserted as the second line of the shell script.

#### **BUGS**

Only shell scripts whose first line is exactly "#!/bin/sh" or "!#/bin/csh", with no trailing flags on the first line are recognized by the tag command.

#### SEE ALSO

winterm – utility to launch applications that require a terminal emulator.

#### **SYNOPSIS**

winterm [-H|-f font|-t title|-p x,y|-s cols,lines|-c command]

#### DESCRIPTION

winterm is a shell script that presents an abstract command line syntax for the user's own terminal emulator. Terminal emulators supported include wsh, psterm, and xterm. The user can preset their preferred termulator (with preferred options) by setting the environment variable \$WINTERM. If WINTERM is unset, winterm provides wsh as a default.

**–H** holds the winterm open.

**-f** font sets the font used by the winterm to *font*.

-t title sets the title used by the winterm to title

 $-\mathbf{p} \mathbf{x}$ ,y sets the position of lower left corner of the winterm to  $\mathbf{x}$ ,y.

-s cols,lines sets the size of the winterm to cols,lines.

-c command feeds the rest of the line as the command to execute. Must be the last flag set when winterm is invoked.

#### **NOTES**

The default *WINTERM* is: WINTERM='wsh -fScreen11 -C54,96,3,2,0,50'.

The **–H** option is not supported by *xterm*.

The  $-\mathbf{f}$  option is not supported by *psterm*.

#### SEE ALSO

WorkSpace – graphical interface to file system

**SYNOPSIS** 

workspace [-w] [-t [name]]

#### DESCRIPTION

WorkSpace provides a graphical, interactive interface to the IRIX file system. This interface is provided via two kinds of views. When invoked with no arguments, WorkSpace opens a window displaying a portion of the IRIX file tree, which can be pruned and expanded on a per user basis. The second type of view, instantiated by opening a directory icon, provides an constantly up-to-date representation of that IRIX directory.

WorkSpace accepts the following options.

-w Open only the WorkSpace view (Note that otherwise WorkSpace will start up with whatever views were open the last time it was used).

#### -t [name]

Allow the user to create a sample WorkSpace view, and install it in /usr/lib/workspace. If a *name* is provided after this option, the file will be installed in the form *name*.wsrc. Otherwise, the file will be called \$USER.wsrc. Each time a new template is added, or one is altered, all users on the system will automatically load it the next time they start their WorkSpace. Note that the template file will have to be installed manually if this option is run by a user without privelege to write into /usr/lib/workspace.

Note that only one WorkSpace process can be run for each user. If the process is invoked again while it is already running, the WorkSpace window will be either opened or popped to the top.

#### SEE ALSO

dirview(1G)

## Index

#### Α

ALTOPEN rule, 2-12 ALTOPEN, 2-2 application environment, 4-3 arc cost, 3-2 arc, 2-17 arcf, 2-17 ascii, 2-5

### В

bclos, 2-17
bgnclosedline, 2-17
bgnline, 2-17
bgnoutlinepolygon, 2-17
bgnpoint, 2-17
bgnpolygon, 2-17
bounding rectangle, 2-14
bounding values, 2-14
bounding values, default, 2-14
BOUNDS rule, 2-14

## C

char, 2-5 color, 2-17 compiling FTR files, 2-22 CONVERT rule, 3-2 COST rule, 3-2 current printer, 3-5 current, 2-16 CURRENTPRINTER, 3-5 CURRENTPRINTER, 3-3

#### D

decimal representation, 2-4 dircontains, 2-5 disabled, 2-16 draw, 2-17 drawing icons, 2-18 DROP rule, 2-12 DROP, 2-2

#### 

endclosedline, 2-17
endline, 2-18
endoutlinepolygon, 2-18
endpoint, 2-18
endpolygon, 2-18
environment variables,
WorkSpace, 2-11, 2-12, 2-13

## F

file icons, 1-1 file typing rules, 1-1, 1-2 file typing rules, summary, 2-2 file typing, 1-1 FILTER rule, 3-3 for-loop, 2-18 FTR files, 1-2

### G

Get Root, 4-2 glob, 2-5

## Н

hexadecimal representation, 2-4

move, 2-18

mode, 2-5

#### Ì

icon description conditionals, 2-15 icon description constants, 2-16 icon description functions, 2-16 icon description language, 2-15 icon description operators, 2-15 icon description variables, 2-16 ICON rule, 2-15 icon status variables, 2-16 ICON, 2-2 iconcolor, 2-16, 2-19 if-statement, 2-18 IRIX file system, 1-1

#### L

LEGEND rule, 2-9 LEGEND, 2-2 linkcount, 2-5 located, 2-16 long, 2-5

isSuper(1), 2-10

### M

MATCH rule, 2-3
MATCH rules, effective, 2-6
MATCH, 2-2
match-expression conditionals, 2-4
match-expression functions, 2-4
match-expression functions, 2-4
match-expression numerical
representation, 2-4
match-expression operators, 2-4
MENUCMD rule, 2-13
MENUCMD, 2-2

Index-2

## 0

octal representation, 2-4 OPEN rule, 2-11 OPEN, 2-2 opened, 2-16 order of precedence, FTR rules, 2-23 outlinecolor, 2-16, 2-19

## P

pclos, 2-18 pdr, 2-18 placement of FTR rules, 2-23 pmv, 2-18 print conversion pipeline, 3-1 print conversion rules, 1-2, 3-1 PRINT rule, 2-13 print, 2-2, 2-5, 2-18 printer types, 3-4

# R

routeprint(1), 2-13, 3-1, 3-3, 3-5 rule keys, 2-1

# S

Save Template, 4-2 Set Lock Off, 4-3 Set Lock On, 4-3 shadowcolor, 2-16, 2-19 short, 2-5 size, 2-5 SPECIALFILE rule, 2-10 SPECIALFILE, 2-2

Version 1.0

string, 2-4 style conventions, 2-19 SUPERTYPE rule, 2-10 SUPERTYPE, 2-2

### T

tag, 2-5 transfer device I/O, 5-2 transfer devices, 1-2, 5-1 Transfer Manager, 1-2, 5-1 transfer scripts, 5-1 Transfer submenu, 5-1 TYPE declaration, 2-3 TYPE, 2-2

#### U

uchar, 2-5 ushort, 2-5

#### ν

valid match-expressions, 2-4 vertex, 2-18

### W

WorkSpace Directory View, 1-2 WorkSpace templates, 1-2, 4-1