
 24th May, 1994

GARDEN Version 1 Revision 4
==========

==========

GARDEN Version 1 release 4 (January 1994)

CONTENTS

==========

==========

1. INTRODUCTION

==========

1.1. GARDEN FEATURES

==========

* GARDEN provides multiple users with precisely controlled and monitored access to large-scale textual and pictorial information resources.

* GARDEN provides developers with an efficient mechanism for organising large numbers of text and image files into high quality interactive applications.

* GARDEN provides system administrators with security mechanisms, time-stamped logging of all user actions and a flexible network distribution model.

* GARDEN supports the special needs of training and educational applications with sophisticated navigation aids and a 'prerequisite relation' allowing access-order constraints to be defined between sets of information.

* GARDEN features an advanced Graphical User Interface capable of transparent adaptation to practically any display resolution, enabling a wide range of machines to act as deliver platforms for GARDEN applications. (The only major requirement is support for 256 colours.)

* GARDEN delivers high levels of portability, expandability and reliablity by employing Open Systems and Object-Oriented technology; in the form of UNIX (SVR4), C, Eiffel and X11 Windows.

1.2. AUTOMATION POLICY

==========

Much of the design of GARDEN was informed by observing how people developed applications using Media Language [ref]. The most important conclusion from this study was that the cost of developing a large-scale information resource is mainly a function of the average time required to insert or amend an item.

GARDEN attempts to minimise development costs by automating as many tasks as possible. Unlike many systems that rely on manual page layout by direct manipulation, GARDEN performs automatic page layout. The benefits of the automatic layout facilities provided by GARDEN include:

1) Very low per screen production cost when compared with manual page layout.

2) Real-time adaptation to different screen/window sizes.

3) Easy insertion/deletion of material in multi-page sequences - no page overflow/underflow problems.

4) Choice of image sizes becomes function of desired image quality rather than incidental details of page layout.

5) Increased independence of the resource from the delivery platform.

6) Increased uniformity of presentation style, this becomes particular helpful when multiple developers are working on the same project.

2. SYSTEM DESCRIPTION

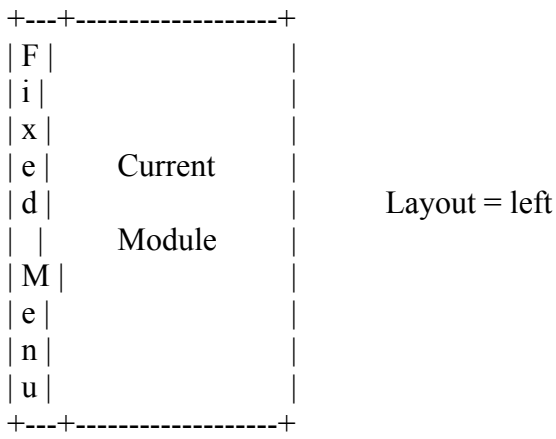
2.1. PRESENTATION AND INTERACTION MODEL

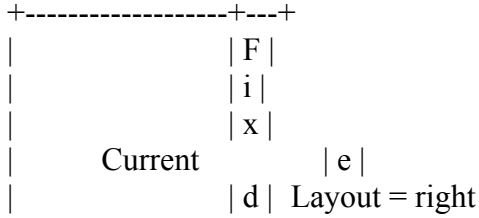
2.1.1. DISPLAY MODEL

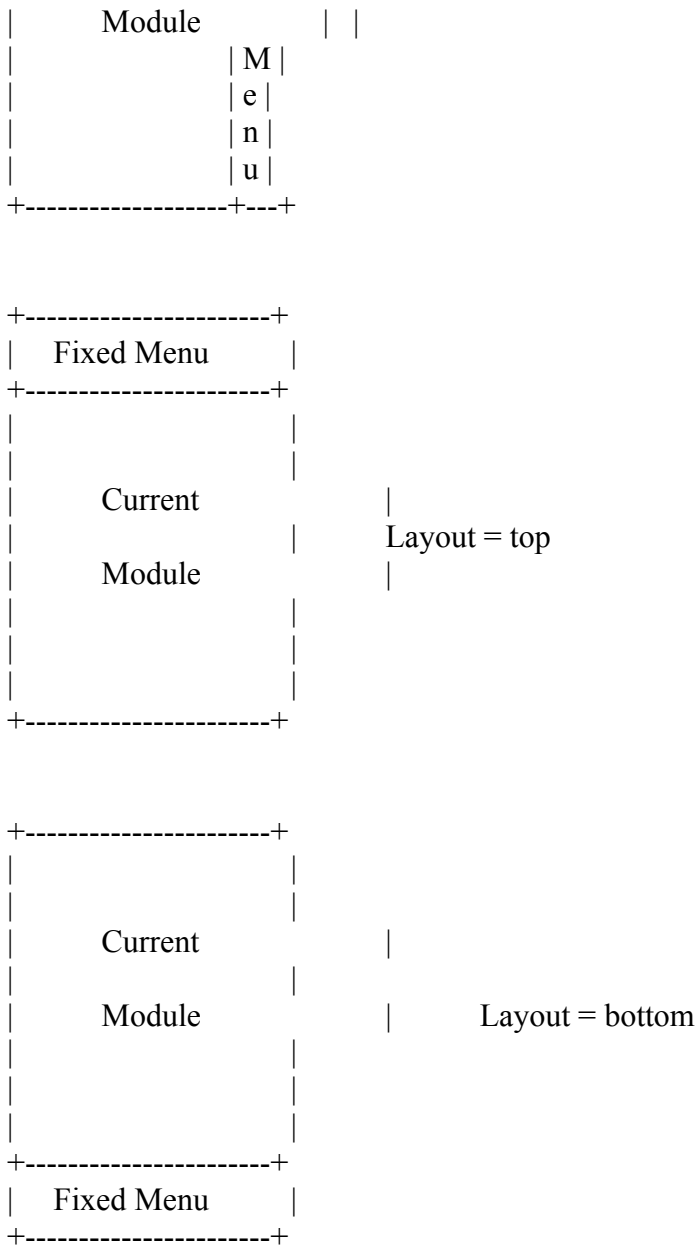
GARDEN displays information in a "window". By default the window occupies the entire screen and the user is prevented from resizing or moving the window - the underlying window manager is effectively hidden from the user. This mode is suitable for dedicated applications where users do not require access to (or knowledge of) the window manager.

An alternative mode allows the width and height of the window to be specified when an application is run, this mode also allows the user to interactively resize and move the window using the conventions supported by the underlying window manager. This enables GARDEN applications to be run alongside other applications, but requires that users learn how to operate the window manager.

A GARDEN application's window is divided into two areas. One area, usually the smallest, is occupied by a "fixed icon menu" located on the left, right, top or bottom of the window (see diagram below, and the "layout" Presentation Control File). The fixed icon menu typically contains icons for help, main menu, index and quit functions. The other area displays the "current module" which changes as a user interacts with an application.



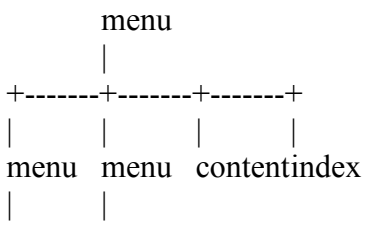


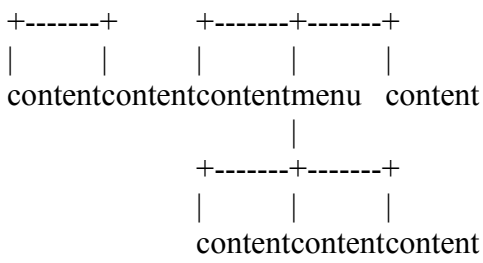


2.1.2. MODULES AND RESOURCE TREES

The "current module" area can display three types of module - menu, content, and index modules.

A GARDEN application consists of a (possibly very large) number of modules organised into a hierarchy called a "resource tree". A resource tree has multiple menu and content modules but only a single index module, eg:

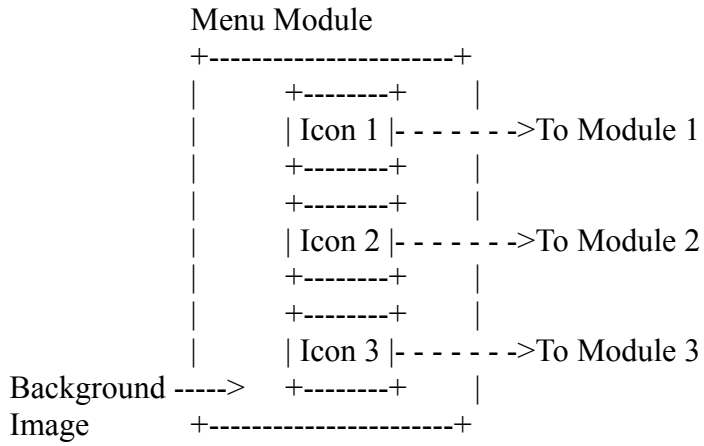




Note that an application's resource tree is structured exactly like the file system hierarchy of a computer - with directories containing files and further sub-directories. This is no coincidence as GARDEN resource trees are actually nothing more than UNIX files and directories organised by a particular naming scheme.

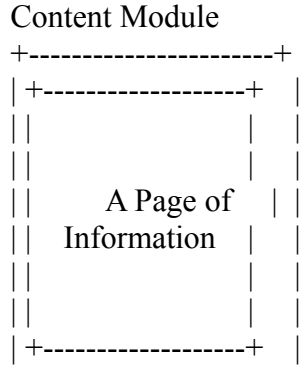
Menu Modules

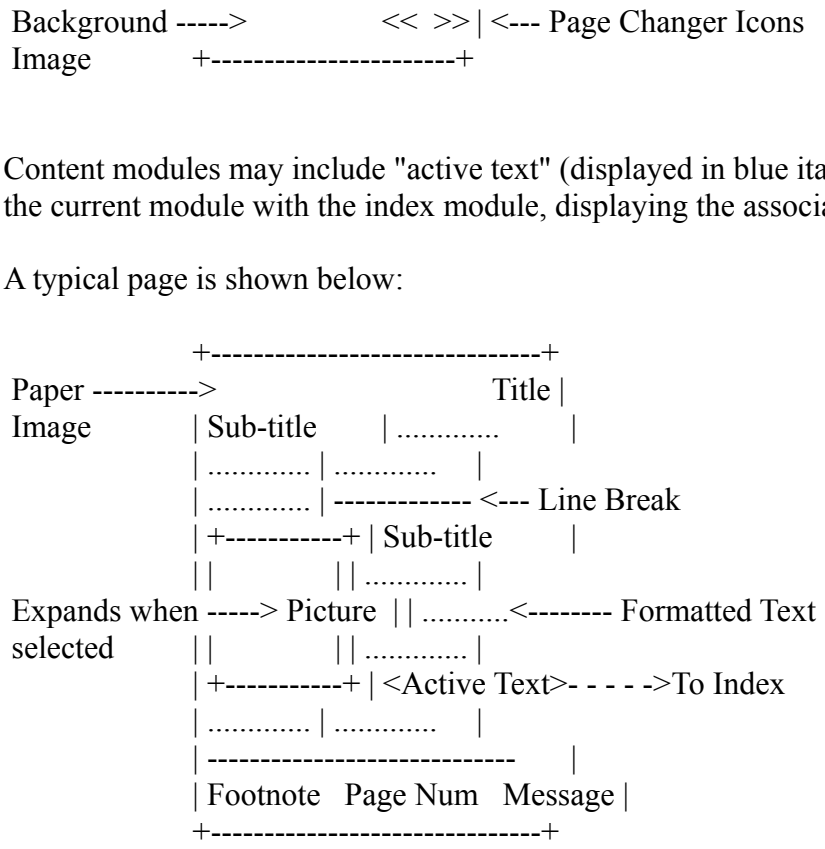
A "menu module" contains an icon for each content or menu module beneath it in the "resource tree". When an icon is selected the corresponding module replaces the current module.



Content Modules

A "content module" contains sequenced textual and pictorial information that GARDEN organises into pages - the exact number and layout of the pages depends upon the size of display area available.



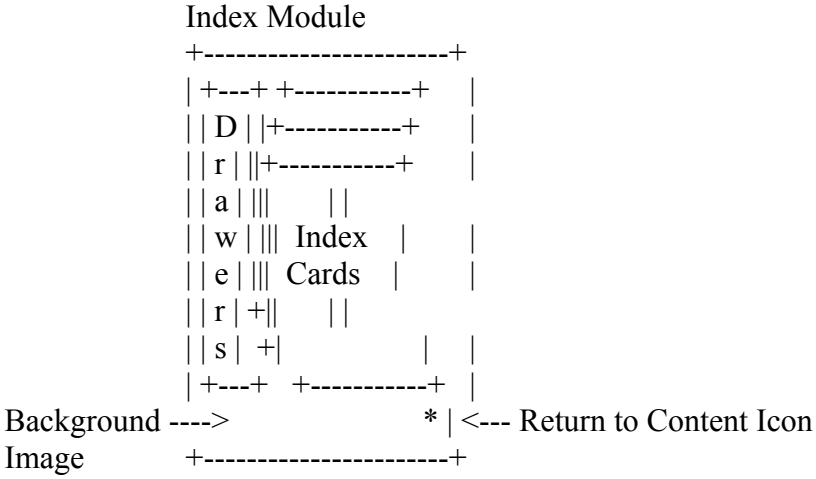


Content modules may include "active text" (displayed in blue italics) that, when selected, replaces the current module with the index module, displaying the associated index entry.

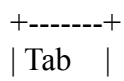
A typical page is shown below:

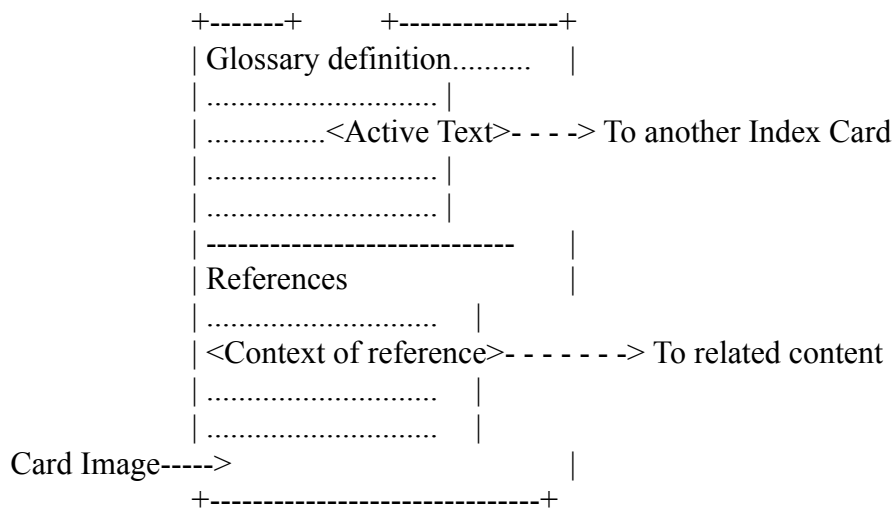
The Index Module

The "index module" simulates an old-fashioned card index. The index entries are stored one per "card" and are alphabetically sorted and divided into a number of "drawers". One drawer full of cards can be inspected at a time. Selecting a closed drawer causes it to replace the currently open one. Selecting the "tab" on a card brings it into full view. Therefore any card can be viewed by the user making a maximum of two selections.



Index entries consist of a glossary definition and a list of references to related content:





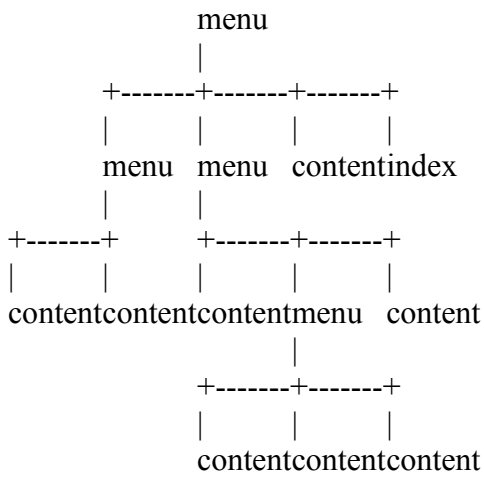
Glossary definitions may contain "active text" that when selected causes another index card to be displayed. Selecting a reference causes the related content module to be displayed.

2.1.2. RELATIONSHIPS BETWEEN MODULES

A number of relationships link modules together:-

Hierarchy Relation

Menu and content modules are organised into a hierarchy (tree) with menus as nodes and content (including the index) as leaves. For example:



The GARDEN compiler determines these relationships automatically by examining an application's resource tree.

Sequence Relation

All modules have a unique "Module ID" that enables them to be sorted into an ordered list. For example:

1 < 1.1 < 1.1.1 < 1.1.2 < 1.2 < 1.3 < 2 < 2.1 < 3

The GARDEN compiler determines these relationships automatically by examining an application's resource tree.

Prerequisite Relation

Constraints may be defined so that a module can only be displayed if some other module has already be viewed in full. For example:

3 requires 2.1

The GARDEN compiler is informed about these relationships by interaction control files called "require".

Glossary Relation

Links from content modules to the index can be established using "active text" (displayed as blue italics). For example:

Content Index

<Acid> ----------> Index entry for "Acid"

The GARDEN compiler is informed about these relationships by content markup.

Reference Relation

Links from the index to content modules can be established using "reference marks". For example:

Index Content

Under index entry for "Acid"

Acid Rain ----------> #begin|Acid|Acid Rain#
...
#end|Acid#

Chemical Properties ----------> #begin|Acid|Chemical Properties#
...
#end|Acid#

The GARDEN compiler is informed about these relationships by content markup.

2.2. THE GARDEN FILE SYSTEM

2.2.1. THE INSTALLATION DIRECTORY

The UNIX environment variable $GARDENPATH (set by the administrator) specifies the file system location for an installation directory containing everything related to GARDEN. Note only the administrator should have write permission on this directory.

The installation directory has the following sub-directories:

bin	For GARDEN programs (called bin for historical reasons).
doc	For GARDEN documentation.
template	A template for new applications.

The $GARDENPATH directory also contains a sub-directory (or symbolic link) for each GARDEN application.

2.2.2. APPLICATION DIRECTORIES

A GARDEN application directory has the following contents:

map	A file created by the 'compile' program.
resources	A directory for the application's resource tree.
users	A directory for the application's user logs.

Note only developers and the administrator should have write permission on an application's directory and the files reachable from it.

2.2.3. RESOURCE TREES

An application's "resources" directory contains a hierarchy of sub-directories representing the structure of the application. Each directory in this "resource tree" represents a module in the application. The tree's nodes (directories containing sub-directories) represent menus, while it's leaves (directories not containing sub-directories) represent content.

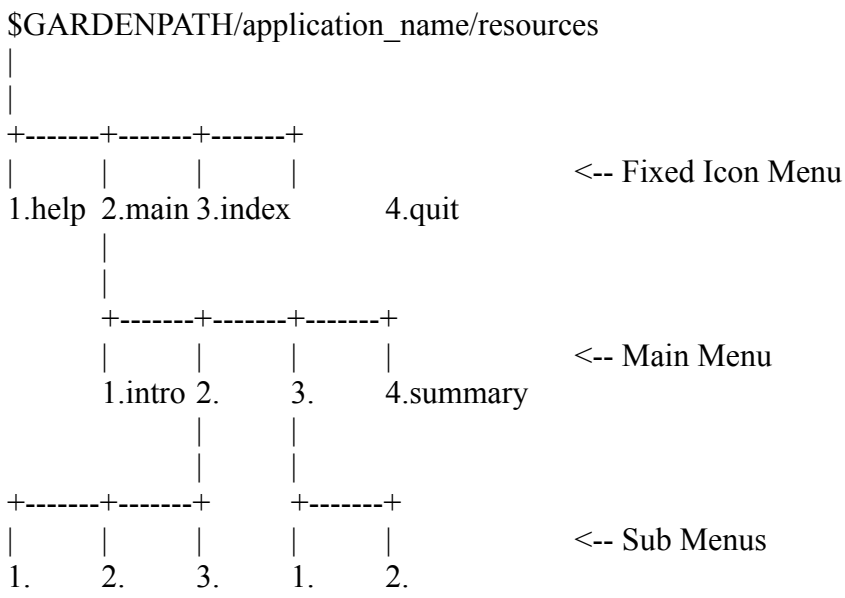
The directories for both menu and content modules contain various "control files" that affect their presentation and interaction. The content module directories also contain "content" files. These are ASCII files containing the module's text, plus optional markup symbols controlling the insertion of images, font usage and so on.

The root directory of an application's resource tree represents the fixed icon menu, this is the menu of icons that is always present while running a GARDEN application.

Certain modules may be assigned special roles by including one of the following interaction control

files: "help", "main.menu", "section.holder", "index", "quit" (see Appendix).

Example Directory Structure



In the simple example above the top "resources" directory represents the fixed icon menu with icons for help, main menu, index and quit.

The following directories represent other menu modules:

Directory	Module ID	Number of Icons
resources/2.main	2	4
resources/2.main/2.	2.2	3
resources/2.main/3.	2.3	2

The following directories represent content modules:

Directory	Module ID
resources/1.help	1
resources/2.main/1.intro	2.1
resources/2.main/2./1.	2.2.1
resources/2.main/2./2.	2.2.2
resources/2.main/2./3.	2.2.3
resources/2.main/3./1.	2.3.1
resources/2.main/3./2.	2.3.2
resources/2.main/4.summary	2.4
resources/3.index	3
resources/4.quit	4

Locations of interaction control files:

File	Location
help	resources/1.help
index	resources/3.index
quit	resources/4.quit

2.2.4. MODULE DIRECTORY NAMES

Module directory names must be valid UNIX filenames and begin with a number followed by a dot, optionally followed by a label to jog the developer's memory, eg: "1.help".

GARDEN sorts modules by their directory name. Because "10." is ordered before "1." when sorted, GARDEN requires that groups of module directory names use the same number of digits. Note that "01.", "02." ... "10.", "11." sorts as expected, while "1.", "2." ... "10.", "11." does not.

Note: UNIX filenames should be kept to a reasonable length (ideally 14 characters maximum, although longer names are allowed under most versions of UNIX, including Sun's) and constructed from 'a'..'z', 'A'..'Z', '0'..'9', '_', '-', '+', and '.'. DO NOT use spaces, tabs or any other weird characters.

2.2.5. MODULE IDS

A module within the resource tree of an application is unambiguously identified by it's "Module ID". A Module ID is simply a short form of the module's directory pathname, relative to the application's "resources" directory. Only the numerical prefix from each directory is used. eg

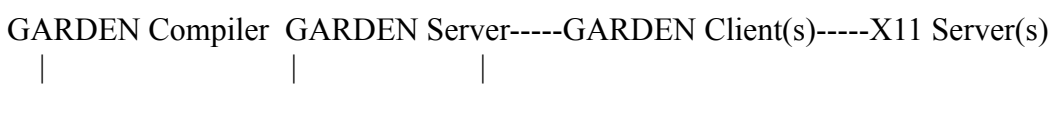
1.help	--has the Module ID-->	1
2.main/1.intro	--has the Module ID-->	2.1
2.main/3./2.	--has the Module ID-->	2.3.2

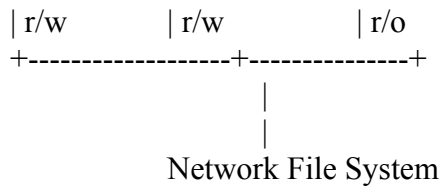
Module IDs are used for logging user interactions and for expressing prerequisite relationships between modules.

2.3. ARCHITECTURE

2.3.1. GARDEN PROCESSES

The GARDEN system consists of multiple processes communicating via local or network links (Internet Domain Sockets) and a distributed file system:





Note: The GARDEN Compiler and Server have Read/Write access to the File System, while GARDEN Clients have Read/Only access. Write operations for Clients are delegated to the Server for execution.

2.3.2. GARDEN COMPILER (program name: 'compile')

A developer process that compiles an application's resource tree into a "map" file. Map files contain information that optimises GARDEN Client's runtime access to resource trees. The compiler also detects and reports errors.

The GARDEN compiler runs under the developer's account and therefore has permission to write to all files reachable from $GARDENPATH/application_name.

Command Line Syntax:

'compile' application_name

Environment Variables:

$GARDENPATH Pathname of GARDEN installation directory.

Files:

$GARDENPATH/bin/compile
The compiler program.
$GARDENPATH/application_name/resources
Application resource tree.
$GARDENPATH/application_name/map
Map file output by compiler.

2.3.3. GARDEN SERVER (program name: 'garden')

A server process providing logging and security services to GARDEN clients. A single GARDEN server process normally serves an entire site - all users and all applications. The GARDEN server is keyed to a particular host machine by an encrypted code and enforces licensing terms based on a maximum number of simultaneous clients.

The GARDEN server runs under the administrator's account and therefore has permission to write

to all files reachable from $GARDENPATH.

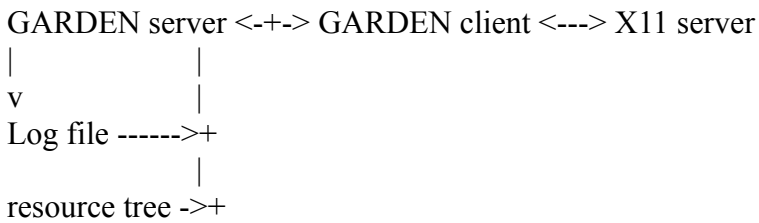
Command Line Syntax:
'garden'

Environment Variables:
$GARDENKEY Encrypted security key.
$GARDENPATH Pathname of GARDEN installation directory.

Files:
$GARDENPATH/bin/garden
The GARDEN Server program.
$GARDENPATH/server.log
System wide log.
$GARDENPATH/application_name/users/user_name
Application/user log.

2.3.4. GARDEN CLIENT (program name: 'run')

A client process controlling an individual user's access to a GARDEN application. A client process is required for each simultaneous user/application pair and connects to the GARDEN server, a user's X11 server and the file system.



GARDEN clients run under user accounts and therefore usually do not have permission to write to any of the files reachable from $GARDENPATH. Clients can however request write operations to be performed by the GARDEN server on their behalf (note the Server can veto such requests).

Command Line Syntax:
'run' application_name [width height]

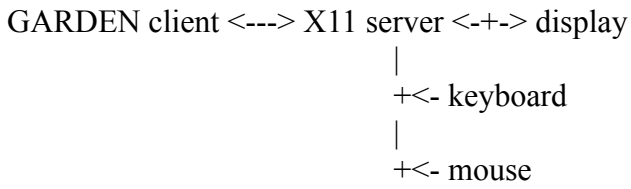
Environment Variables:
$GARDENPATH Pathname of GARDEN installation directory.
$GARDENFLAGS Optional configuration flags (see Appendix).
$GARDENHOST Name of GARDEN Server host (default is local host).
$DISPLAY Name of X11 Server display (default :0.0).

Files:
$GARDENPATH/bin/run
The GARDEN Client program.
$GARDENPATH/application_name/users/user_name
Application/user log.
$GARDENPATH/application_name/resources
Application resource tree.
$GARDENPATH/application_name/map

Map file output by compiler.

2.3.5. X11 SERVER

A server process providing network transparent, machine and operating system independent Graphical User Interface services for one or more GARDEN clients. Each delivery platform has it's own X11 Server.



2.3.6. NETWORK FILE SYSTEM

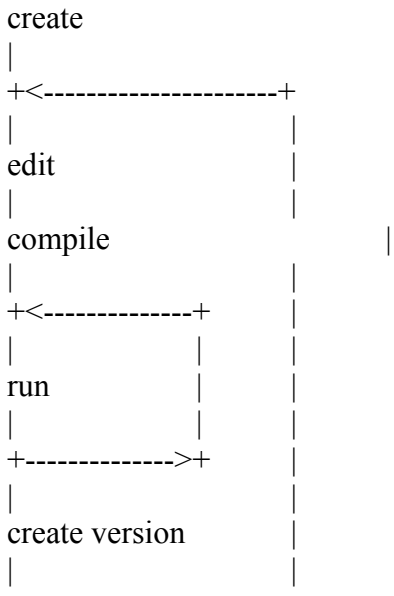
GARDEN is designed to exploit the machine independent support for remote mounting and symbolic linking provided by the distributed file system 'NFS'. This enables the storage space used by GARDEN applications to be easily partitioned or duplicated and distributed across a number of file servers. Partitioning maximises storage space, duplication optimises performance.

See SYSTEM ADMINISTRATION for further details.

3. APPLICATION DEVELOPMENT

3.1. AN OVERVIEW OF THE DEVELOPMENT PROCESS

The diagram below shows the typical life-cycle of a GARDEN application:



```
+---------------------->+
|
destroy
```

Creating a New Application

New applications are most easily created by making a copy of an existing one. A 'skeletal' application is provided in $GARDENPATH/template/ for this purpose.
The following UNIX command creates a new application called "application_name" by copying the template:

```
cp -r $GARDENPATH/template $GARDENPATH/application_name
```

Note: Only the administrator may create new applications.

Editing an Application

Application editing is performed by modifying the application's resource tree. Typical operations are create a new menu module, create a new content module, markup content, add a new index entry.

Resource trees contain only three types of element:

UNIX Directories
ASCII Text Files
GARDEN Format Image Files

UNIX directories and ASCII text files can be created, renamed and deleted using a variety of tools, eg, file managers, text editors and UNIX shells.

GARDEN format image files are translated from standard image format files by one of the GARDEN image translation tools:

pcx2g	PCX to GARDEN
sr2g	Sun Raster to GARDEN [NOT AVAILABLE IN V1R3]

Compiling an Application

After editing an application it must be compiled. The GARDEN compiler ('compile') inspects an application's resource tree and builds a map file read by the GARDEN Client program. The following UNIX command compiles an application called "application_name":

```
compile application_name
```

Running an Application

The following UNIX command runs an application called "application_name" on the local host:

run application_name

To run an application on a remote host enter the following UNIX commands:

```
rlogin remote_host
setenv DISPLAY x11_host:0.0
setenv LD_LIBRARY_PATH /usr/openwin/lib
run application_name
```

Replacing 'remote_host', 'x11_host' and 'application_name' with the name of the remote host, the name of the X11 Server host and the name of the application, respectively.

Creating a Version of an Application

The following UNIX command creates a new application called "new_application_name" by making a copy of an existing application called "old_application_name":

```
cp -r $GARDENPATH/old_application_name $GARDENPATH/new_application_name
```

Note: Only the administrator may create a version of an application.

Destroying an Application

The following UNIX command destroys an application called "application_name":

```
rm -r $GARDENPATH/application_name
```

Note: Only the administrator may destroy applications.

3.2. BASIC MODULE DEVELOPMENT

3.2.1. DIRECTORY MANIPULATIONS

A GARDEN module is represented by a UNIX directory and it's contents.

Directories can be manipulated using a file manager or the following UNIX commands:

Change current directory:

```
cd directory_pathname
```

Make a directory:

```
mkdir directory_pathname
```

List contents of a directory:

ls directory_pathname

Move/rename a directory:

mv old_directory_pathname new_directory_pathname

Copy a directory (and it's contents, recursively):

cp -r old_directory_pathname new_directory_pathname

Remove a directory (and it's contents, recursively):

rm -r directory_pathname

Further information about these commands can be obtained from the on-line manual by entering the following UNIX commands:

man cd
man mkdir
man ls
man mv
man cp
man rm
man man

Alternatively, try using the xman interactive manual browser, just enter:

xman

3.2.2. MODULE TITLES

GARDEN refers to a module by it's module ID or pathname, however, humans normally prefer meaningful titles. If a module's directory contains an ASCII text file called "title" then the file's contents will be read and used as the module's title.

A module's title is used for the title on content pages and also to label icons (if the file "icon.label" exists). Content page titles are always displayed as a single line of text, however, icon labels may be displayed as several (centre justified) lines. Vertical bar characters '|' may be used to mark points where the title should be split when used as an icon label, eg:

"This is a|rather long|title" --> This is a
rather long
title

3.2.3. BACKGROUND IMAGES

Every module has a "background image" that is scaled and displayed when the module is displayed.

Module background images are stored in presentation control files called "back.i" using the GARDEN image file format.

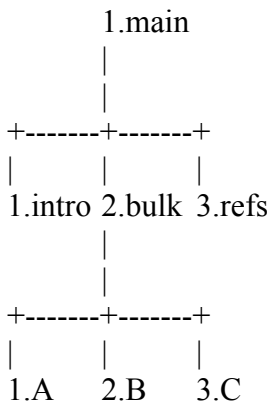
If a module's directory contains a "back.i" file then the file's contents will be used as the module's background image. If no "back.i" file is found then GARDEN searches up the resource tree until a "back.i" file is found - this is known as "inheriting a resource".

Resource Inheritance is used for the following presentation control files:

"back.i"	Background image.
"color.code"	Color used for drawing lines on pages.
"paper.i"	Paper image tile for pages.

Example of Resource Inheritance

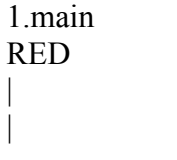
Consider the following resource tree structure:

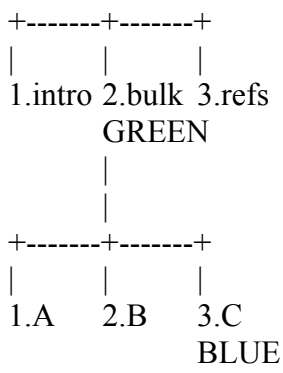


This structure would be represented in the GARDEN file system as the following directories (pathnames relative to $GARDENPATH/application_name/resources):

Directory	Module ID
1.main	1
1.main/1.intro	1.1
1.main/2.bulk	1.2
1.main/2.bulk/1.A	1.2.1
1.main/2.bulk/2.B	1.2.2
1.main/2.bulk/3.C	1.2.3
1.main/3.refs	1.3

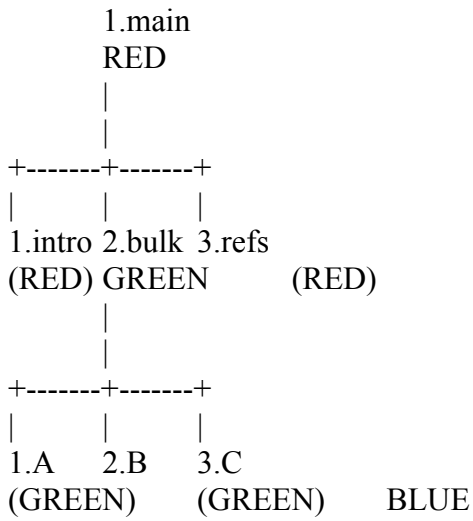
Imagine we have three different "back.i" files - a RED one in "1.main", a GREEN one in "1.main/2.bulk" and a BLUE one in "1.main/2.bulk/3.C":





The four directories that do not contain a "back.i" file inherit the first "back.i" file found as GARDEN searches up through the directories "above" it in the resource tree.

The diagram below shows which "back.i" file would be used by each module, given the example above:



3.2.4. PREREQUISITES

An ASCII text file called "require" may be used to specify the prerequisites of a module, ie which module(s) must be completed by the current user before they are allowed to access the module containing this file. The file should contain either the word "previous" or one or more valid Module IDs.

The "previous" option covers the common case where a module requires the previous module in a series to be completed, eg if a module with an ID of 2.4 has a "require" file containing the word "previous", then the module may only be accessed after the user has completed a module with an ID of 2.3).

The module ID option enables arbitary prerequisite relationships to be established. To prevent "deadlock" situations (eg, where module A requires module B and module B requires module A) the module ID in a "require" file must be less than the ID of the module containing the file, eg 2.4 can

require 2.3 but 2.3 can not require 2.4.

3.3. ICONS

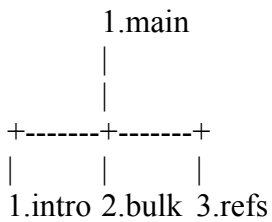
3.3.1. ICON FILES

All modules, except for the fixed icon menu, have a menu module above them in the resource tree (known as a "parent menu"). A module is represented in it's parent menu by an icon. Information about this icon is stored in the following files, located in the module's own directory (not the parent menu's one):

icon.i	Contains the icon's graphics.
iconv.i	Contains a "visited" variant of the "icon.i" graphics.
icon.label	Controls a textual label for the icon.

The "icon.i" file is always required, the other files are optional.

Example:



The module 1.main is a menu with 3 icons in it.
The directory 1.main/1.intro contains the first icon's files,
1.main/2.bulk contains the second icon's files and finally
1.main/3.refs contains the third icon's files.

Note the icon files in 1.main define the icon for this menu in the menu above it in the resoucrce tree (in the fixed icon menu).

3.3.2. ICON FORMAT

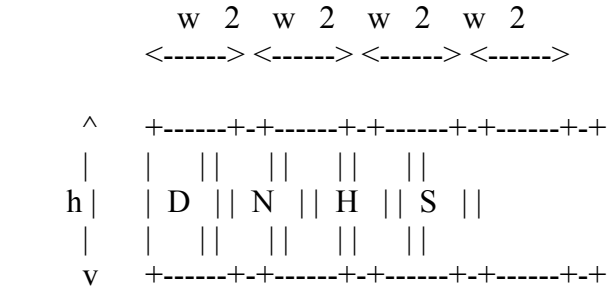
GARDEN icons have four basic states:

State	Interactive?	Under Pointer?	Button Pressed?
Disabled	No	Don't care	Don't care
Normal	Yes	No	Don't care

Highlighted	Yes	Yes	No
Selected	Yes	Yes	Yes

Each of these states is represented by a different graphic, providing the user with visual feedback about the icons interactivity.

Graphics for these four states are stored in a GARDEN image file called "icon.i" and are arranged according to a standard convention - in a horizontal line along with four 2 pixel wide spacers:

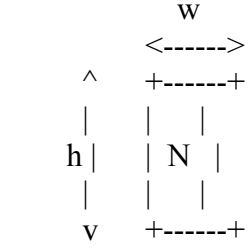


w = Icon Width [Note: Total image width = (w + 2) * 4]
h = Icon Height
D = Disabled
N = Normal
H = Highlighted
S = Selected

The GARDEN image translation tools enable a developer to designate one of the image colors as "transparent". This simple mechanism enables arbitary shaped icons to be defined.

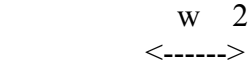
The following procedure can be used to produce an icon.i file using any suitably sized image as a starting point.

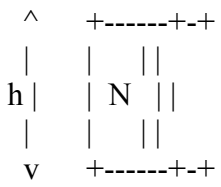
1. Get the initial image into a decent graphics editor, this image will the form the basis of the icon's Normal graphic.



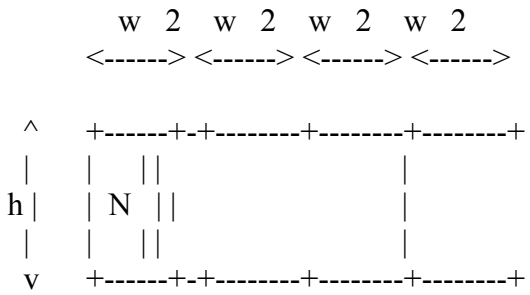
2. Select pure bright blue (Red=0%, Green=0%, Blue=100%) and paint all pixels that are to be transparent using this color. (This particular blue is color index 21 in the GARDEN palette, the number 21 is therefore give to the pcx2g program later on in this procedure.)

3. Resize the canvas so that you have a 2 pixel wide border on the right-hand side.

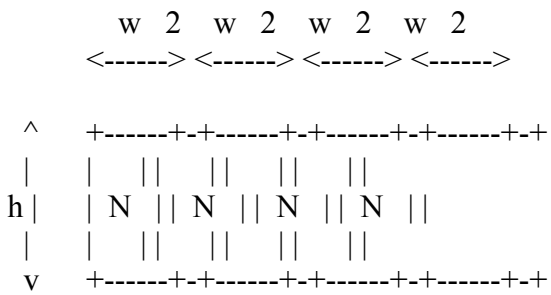




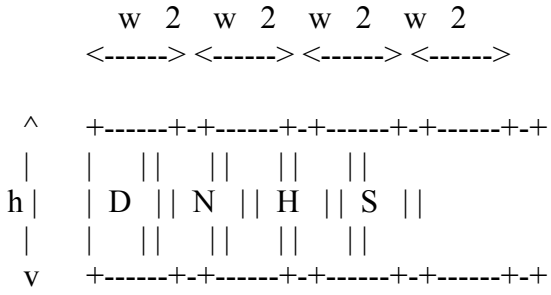
4. Increase the width of the canvas by a factor of 4.



5. Copy the graphic including the 2 pixel spacer 3 times into the newly created space.



6. Modify the colors of the graphics so that the four versions look different. For example make the first graphic gray for the disabled state, leave the second as is for the normal state, make the third brighter for the highlighted state and finally change the hue of the last graphic for the selected state.



7. Save the file in 8 bit indexed color PCX format as "icon.pcx" and exit the editor.

8. Enter

pcx2g

Do not add the image filename to this command line, just press return then answer the prompts as

follows:

Enter PCX filename:
    icon.pcx

Enter output filename:
    icon.i

Enter required width (or 0 for width of input):
    0

Enter required height (or 0 for height of input):
    0

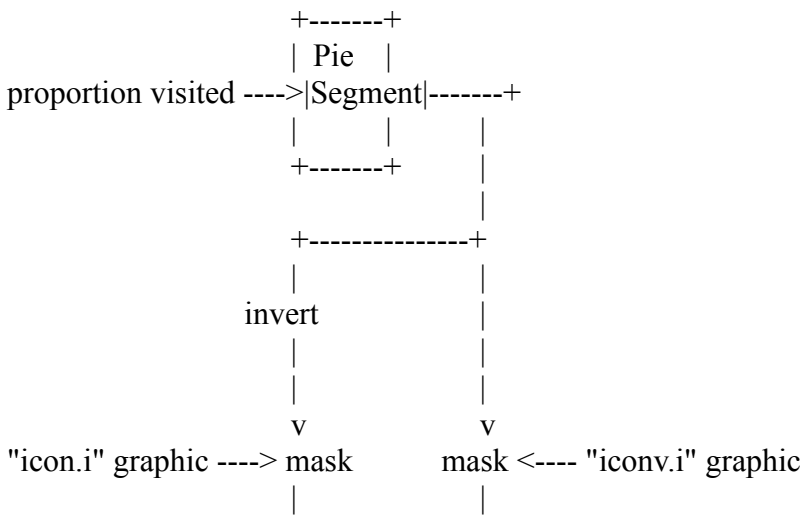
Enter transparent color (or 0 for none):
    21

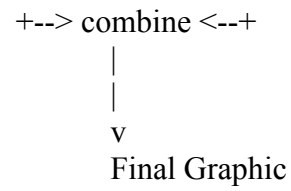
9. Copy icon.i to the required place in the applications resource tree.

3.3.3. PROPORTION VISITED FEEDBACK

In addition to the feedback provided by the basic four-state icon described above, GARDEN also supports a novel mechanism that provides the user with an indication about the proportion of the application (available via an icon) that has already been visited. This is achieved by the inclusion of an "iconv.i" file in a module's directory.

The "iconv.i" file should be a contrasting variant of "icon.i", eg slightly brighter or a different hue. The graphics supplied in "icon.i" and "iconv.i" are combined by replacing a "pie chart" shaped segment from "icon.i" with the graphics from "iconv.i". The size of the segment being proportional to the amount of material (available via the icon) that the current user has already visited (in the current AND previous sessions).





3.3.4. ICON RECOMMENDATION

If an "iconv.i" file exits for an icon then it may become a "recommended icon".
This is when GARDEN recommends that a user selects a particular icon by displaying a little flashing moon orbiting the icon.

Together, icon states, proportion visited feedback, prerequisites and icon recommendation provide users with sophisticated navigation support.

3.3.5. ICON LABELS

The existance of an "icon.label" file in the same directory as an "icon.i" file indicates that the icon should have a textual label attached. The text used for the icon is taken from the "title" file in the same directory. The position and background color for the label are controlled by ASCII text inside the "icon.label" file. This file contains one of the words above, right, left or below then a space then the color number to use for the labels background (See GARDEN Palette appendix). Note a color number of 0 makes the text float over whatever background is behind it.

3.3.6. ICON LAYOUT

The layout of icons within a menu is controlled by the inclusion in the menu module's directory of an icon layout file called "vertical", "horizontal", "circular", "radial" or "arbitary". These ASCII text files contain numeric parameters that control the details of the layout.

Icon Positions

Icons are positioned using a relative coordinate system that scales layouts according to the display area available. X coordinates represent horizontal positions relative to the width of the display area - from 0.0 on the left to 1.0 on the right. Y coordinates represent vertical positions relative to the height of the display area - from 0.0 at the top to 1.0 at the bottom. An icon with coordinates of 0.5, 0.25 would therefore be horizontally centred and one quarter of the way down the display area.

Control Files

Filename	Parameter 1	Parameter 2	Parameter 3	Parameter 4

vertical	x	min_y	max_y	ideal_gap
horizontal	min_x	max_x	y	ideal_gap
circular	centre_x	centre_y	radius	
radial	centre_x	centre_y	[angle	radius]*
arbitary	[x	y]*		

Notes:

x parameters in range 0.0..1.0 mapping to area width
y parameters in range 0.0..1.0 mapping to area height
radius parameters in range 0.0..1.0 mapping to area height
angle parameters are degrees clockwise from Noon
parameters enclosed in []* are repeated for each icon.

Examples:

Filename	File content	Layout
vertical	0.2 0.1 0.8 0.2	Icons are arranged vertically. Icons have x = 20% of area width. Icons have y >= 10% of area height and y <= 80% of area height. When possible, icons are vertically separated by 20% of the area height.
horizontal	0.25 0.75 0.5 0.15	Icons are arranged horizontally. Icons have x >= 25% of area width and x <= 75% of area width. Icons have y = 50% of area height. When possible, icons are horizontally separated by 15% of the area width.
circular	0.5 0.4 0.33	Icons are arranged in a circle. The centre of the circle is located at x = 50% of the area width and y = 40% of the area height. The radius of the circle is 33% of the area height.
radial	0.5 0.4 0 0 0 0.1 90 0.2 180 0.3	Four icons arranged in a spiral. The centre of the spiral is located at x = 50% of the area width and y = 40% of the area height. The first icon has angle = 0, radius = 0 and is located at the centre of the spiral. The second icon has angle = 0, radius = 0.1 and is located 10% of the area height above the first icon. The third icon has angle = 90, radius = 0.2 and is located 20% of

the area height to the right of the
first icon.
The fourth icon has angle = 180,
radius = 0.3 and is located 30% of
the area height below the first icon.

arbitary 0.4 0.25 0.6 0.25 0.4 0.45 0.6 0.45

Four icons arranged in a square.
The first icon has x = 40% of area
width and y = 25% of area height.
The second icon has x = 60% of area
width and y = 25% of area height.
The third icon has x = 40% of area
width and y = 45% of area height.
The fourth icon has x = 60% of area
width and y = 45% of area height.

If no icon layout file found then the default is "vertical" with parameters 0.5, 0.1, 0.9, 0.2.

3.4. CONTENT MARKUP

Directories for content modules contain a file called "content" and do not contain any sub-modules. A content file is simply an ASCII text file containing the text for the module plus optional markup notation using the GARDEN Content Markup Language (GCML).

GARDEN performs automatic text formatting and page layout. Text should therefore only include return characters at the end of paragraphs. Most word processors can export plain ASCII text in this format.

Font usage, picture insertion and indexing is achieved by inserting "markup" symbols (also known as tags) into the text. Note that these markup symbols are only interpreted by GARDEN, they are NOT displayed.

Example Content Markup

```
<<Water>>
Water <molecules> have the chemical equation: {-serif}H{sub}2{-sub}O{serif}.
#begin|water|Picture of water molecule#
#water.i#
#end|water#
```

The above content markup is interpreted as follows:

<<Water>> indicates that "Water" is a subtitle, subtitles are displayed in bold and start on a new

column or page if only a small amount of space is available in the current column.

<molecules> indicates that a click on "molecules" should cause the corresponding Index entry to be displayed. Text marked this way is displayed in blue italics and changes colour when beneath the pointer. A beep and an error message will be emitted when this text is displayed if there is no entry for "molecules" in the index.

{-serif}H{sub}2{-sub}O{serif} indicates that 'H2O' should use a sans-serif font and that the '2' should be subscripted. Let's look at this in more detail...

{-serif} indicates that a font without serifs (eg Helvetica) should be used from this point (until {serif} is found). The default is a serif font (eg New Century Schoolbook).

'H' is displayed in a sans-serif font.

{sub} indicates that text should be subscripted from this point (until {-sub} is found). The default is non-subscripted.

'2' is displayed subscripted in a sans-serif font. Note that the sans-serif and subscripting attributes are independent of each another.

{-sub} indicates that subscripting should be turned off at this point.

'O' is displayed in a sans-serif font.

{serif} indicates that a font with serifs should be used form this point until {-serif} is found.

In general {X} means switch on attribute 'X', while {-X} means switch off attribute 'X'; where X may be any of 'bold', 'italic', 'underline', 'super' (superscript), 'sub' (subscript) or 'serif'.

#begin|water|Picture of water molecule# indicates the beginning of a reference relation. Any content between this and the next #end|water# to be found is available by selecting the reference "Picture of water molecule" on the index card for water.

#water.i# indicates that an image called "water.i" (located in the module's directory) should be inserted at this point.

#end|water# indicates the end of the reference described above.

See GCML SYNTAX for a further information.

Displaying Markup Characters

To display (rather than interpret) a character used for markup, place a backslash '\' character immediately before it, eg:

```
{       ->      \{
}       ->      \}
<       ->      \<
>       ->      \>
<<      ->      \<\<
```

>> -> \>\>

Unfortunately, it is not currently possible to display the # character.

3.5. THE INDEX MODULE

The index module contains a special 'content' file. This ASCII text file contains a list of index extries in a format known as the GARDEN Index Markup Language (GIML), this differs slightly from the markup language used for regular content files (GCML). See GIML SYNTAX for further information.

Example Index Entry

```
##Acid|Acids|Acidic##
A substance which, on being dissolved in water produces hydrogen <ions> at concentrations which exceed those found in pure water. Acids are strongly corrosive and will dissolve metals from many natural materials, eg. rock.
```

The above content markup is interpreted as follows:

##Acid|Acids|Acidic## describes the keys associated with this index entry, ie "Acid", "Acids" and "Acidic". The remainder of the text is the definition.

<ions> indicates that a click on the word "ions" should cause the corresponding index entry to be displayed.

4. SYSTEM ADMINISTRATION

4.1. INFORMATION REQUIRED BY THE GARDEN ADMINISTRATOR

A) The hostname and hostid of the machine that will host the GARDEN Server.
B) The login name and password for the account to be used for administration. This does not have to be root, but should use 'csh' not 'sh'.
C) The pathname of the directory in which to install GARDEN.
D) The name of the tape device to use for loading GARDEN.

4.2. INSTALLATION

1) Login to the host (A) using the login (B), eg.

```
rlogin -l gardener sun01
```

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2) Create the installation directory (C), eg.

```
mkdir /apps/GARDEN
```

Replacing '/apps/GARDEN' with the installation directory pathname (C).

3) Obtain the hostid for the server, eg.

```
hostid
```

4) Ring Colin (on 0532 745536) and ask for a security key for the hostid.

5) Add the following lines to your '.cshrc' file:

```
setenv GARDENPATH /apps/GARDEN
setenv GARDENKEY 73b10a46
setenv GARDENTAPE /dev/rst0
set path = ($GARDENPATH/bin $path)
```

Replacing '/apps/GARDEN' with the installation directory pathname (C)
and '73b10a46' with the security key obtained in step (4)
and '/dev/rst0' with the name of the tape device (D).

6) Re-execute the '.cshrc' file, eg.

```
source .cshrc
```

7) Read the contents of the distribution tape, eg.

```
cd $GARDENPATH
tar xvf $GARDENTAPE
```

4.3. SERVER MANAGEMENT

4.3.1. LAUNCHING THE GARDEN SERVER

1) Login to the host (A) using the login (B), eg.

```
rlogin -l gardener sun01
```

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2a) Execute the 'garden' Server in '$GARDENPATH/bin' as a foreground process, eg

```
garden
```

OR

2b) Execute the 'garden' Server in '$GARDENPATH/bin' as a background process, eg

```
garden>/dev/null&
```

4.3.2. SHUTTING DOWN THE GARDEN SERVER

If the 'garden' Server was launched as a forground process (2a above), then simply use control-C to terminate the Server.

If the 'garden' Server was launched as a background process (2b above), then use 'kill -9 %garden'. Alternatively, find the 'pid' using 'ps' and terminate the Server using 'kill -9 pid'.

4.4. APPLICATION MANAGEMENT

4.4.1. RUNNING AN APPLICATION ON A LOCAL HOST

1) Login to the local host.

2) Run the GARDEN client, eg.

```
run application
```

Replacing 'application' with the name of the application.

4.4.2. RUNNING AN APPLICATION ON A REMOTE HOST

1) Login to the remote host, eg.

```
rlogin sun02
```

Replacing 'sun02' with the hostname of the remote host.

2) Set the environment variable 'DISPLAY' to the local X11 display name, eg.

setenv DISPLAY sun03:0.0

3) Set the environment variable 'LD_LIBRARY_PATH', eg

setenv LD_LIBRARY_PATH /usr/openwin/lib

4) Run the GARDEN client, eg.

run application

Replacing 'application' with the name of the application.

4.4.3. IMPORTING AN APPLICATION

1) Login to the host (A) using the login (B), eg.

rlogin -l gardener sun01

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2) Read the contents of the application tape, eg.

```
cd $GARDENPATH
tar xvf $GARDENTAPE
```

4.4.4. EXPORTING AN APPLICATION

1) Login to the host (A) using the login (B), eg.

rlogin -l gardener sun01

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2) Write the application to the tape, eg.

```
cd $GARDENPATH/application
tar cvf $GARDENTAPE
```

Replacing 'application' with the name of the application to export.

4.4.5. UPDATING AN APPLICATION

1) Login to the host (A) using the login (B), eg.

rlogin -l gardener sun01

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2) Save the existing user logs, eg.

cp -r $GARDENPATH/application/users $GARDENPATH/tmp

Replacing 'application' with the name of the application.

3) Read the contents of the application tape, eg.

cd $GARDENPATH
tar xvf $GARDENTAPE

4) Restore the old user logs, eg.

mv $GARDENPATH/tmp $GARDENPATH/application/users

Replacing 'application' with the name of the application.

4.5. USER MANAGEMENT

4.5.1. ADDING USERS

1) Login to the host (A) using the login (B), eg.

rlogin -l gardener sun01

Replacing 'gardener' with the login name (B)
and 'sun01' with the hostname (A).

2) Create an empty log file for the user, eg.

touch $GARDENPATH/application/login_name

Replacing 'application' with the name of the application
and 'login_name' with the login name of the user.

3) Add the following lines to the user's '.cshrc' file:

setenv GARDENHOST sun01
setenv GARDENPATH /apps/GARDEN
set path = ($GARDENPATH/bin $path)

Replacing 'sun01' with the hostname (A)

and '/apps/GARDEN' with the installation directory pathname (C).

If full logging is required or the X11 servers are not Suns you may also wish to set the GARDENFLAGS environment variable by adding one of the following lines to the user's '.cshrc' file:

```
setenv GARDENFLAGS L+
        -- switches on full logging

setenv GARDENFLAGS S-
        -- switches off Sun specific behaviour

setenv GARDENFLAGS L+S-
        -- switches on full logging and
        -- switches off Sun specific behaviour
```

See the GARDEN ENVIRONMENT VARIABLES appendix for more details.

4.5.2. REMOVING USERS

1) Login to the host (A) using the user account (B), eg.

```
rlogin -l gardener sun01
```

2) Delete the log file for the user, eg.

```
rm  $GARDENPATH/application/login_name
```

Replacing 'application' with the name of the application
and 'login_name' with the login name of the user.

3) Remove the lines added to the user's '.cshrc' file in (3 above).

4.5.3. RESETTING USER LOGS

1) Login to the host (A) using the user account (B), eg.

```
rlogin -l gardener sun01
```

2) Delete the log file for the user, eg.

```
rm  $GARDENPATH/application/login_name
```

Replacing 'application' with the name of the application
and 'login_name' with the login name of the user.

3) Create a new empty log file for the user, eg.

touch $GARDENPATH/application/login_name

Replacing 'application' with the name of the application
and 'login_name' with the login name of the user.

Refer to the LOG FILE FORMATS appendix for more information on logging.

4.6. DISTRIBUTION

4.6.1. PERFORMANCE ISSUES

GARDEN's flexible distribution model allows heavy-duty applications to spread storage and computational load across large networks of hetrogeneous machines.

Typical amounts of network traffic between processes:

Connection	Load
'garden' - 'run'	LIGHT
'garden' - 'NFS'	LIGHT
'run' - 'X11'	HEAVY
'run' - 'NFS'	MEDIUM

For optimal performance a 'run' process should execute on the same host as the X11 server and the application files that it reads should be mounted as locally as possible.

A GARDEN Client (run program) examines the $GARDENHOST and $DISPLAY environment variables to determine which machine hosts the GARDEN server (garden program) and which machine (and screen) is acting as a delivery platform (X11 server).

4.6.2. TYPICAL CONFIGURATIONS

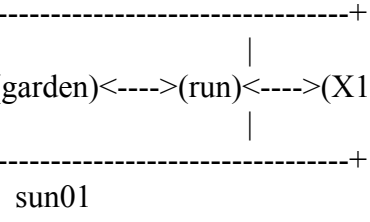
Note: NFS services are omitted for clarity.

Key:

```
+----+
|    |      = host
+----+

(name)  = process

<---->   = local connection

<--/~/--> = network connection
```

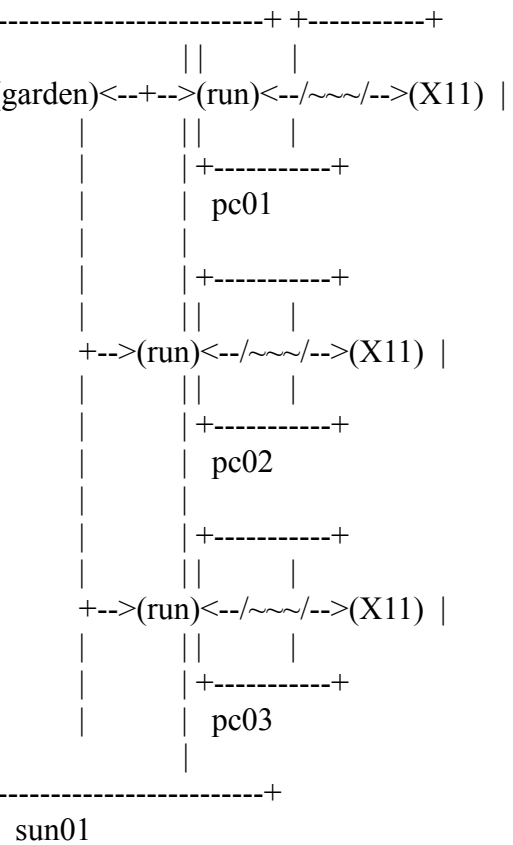
Stand-Alone Single-User Configuration:

(garden) host ($GARDENHOST) = sun01
(run) host = sun01
(X11) display ($DISPLAY) = :0.0



Small-Scale Multi-User Configuration:

(garden) host ($GARDENHOST) = sun01
(run) host = sun01
(X11) display ($DISPLAY) = pc01:0.0 / pc02:0.0 / pc03:0.0

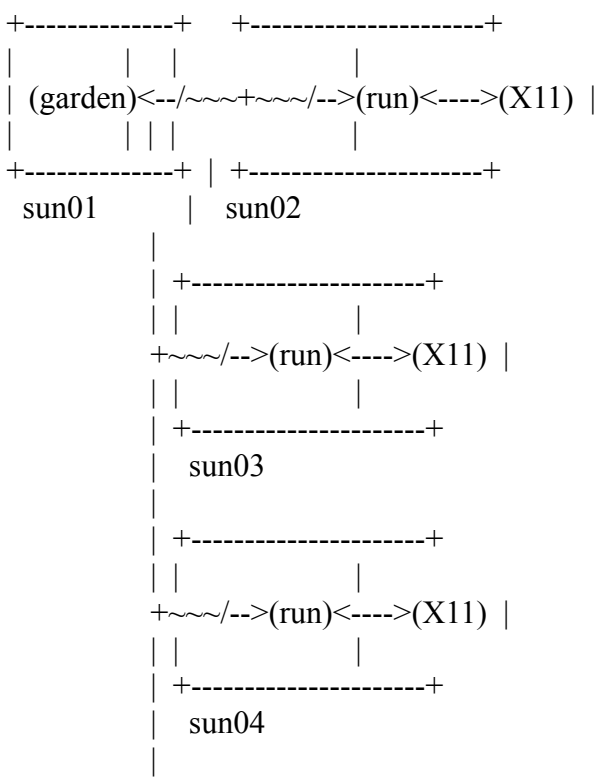


This configuration allows a single UNIX host to support multiple non-UNIX delivery platforms (eg PCs running X11 server software).
Note: Virtual memory management techniques are employed to enable multiple 'run' processes to

efficiently share memory when running on a single host machine.

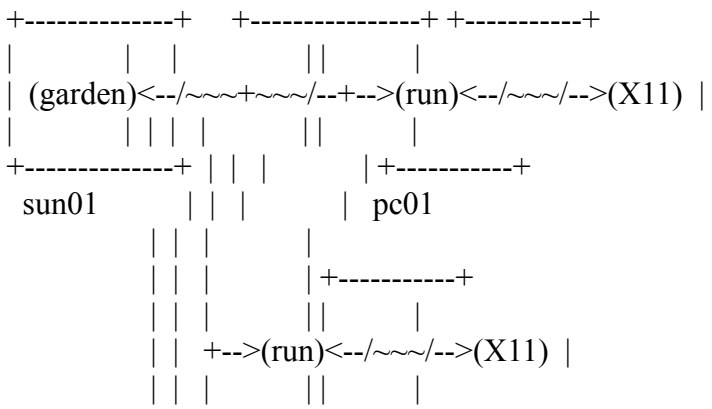
Medium-Scale Multi-User Configuration:

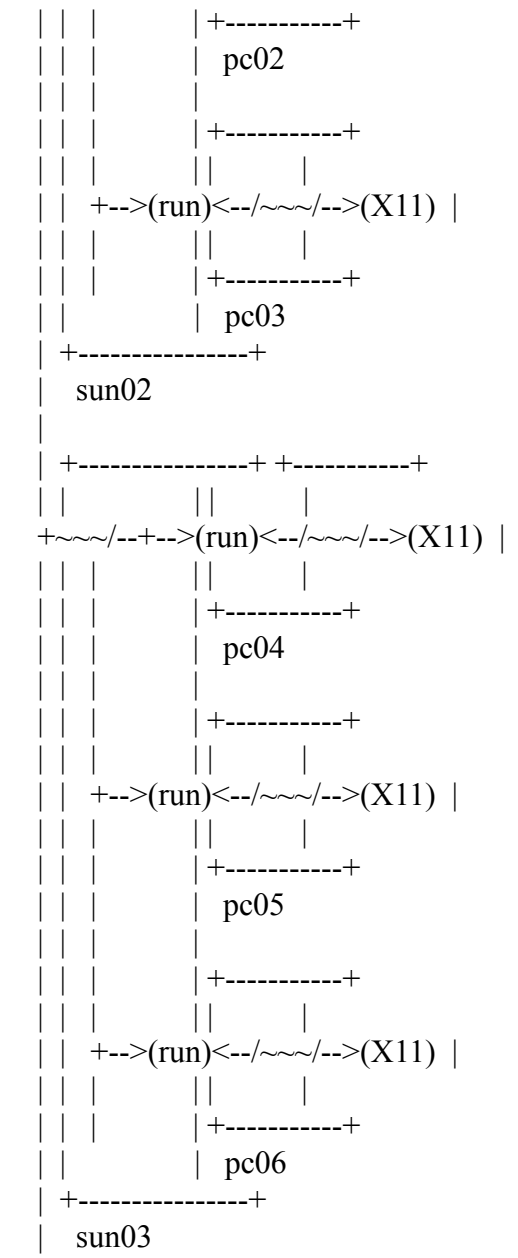
(garden) host ($GARDENHOST) = sun01
(run) host = sun02 / sun03 / sun04
(X11) display ($DISPLAY) = :0.0



This configuration delivers better performance than the previous one, but requires that the delivery platforms are UNIX machines.
Note: Using a local connection to carry the heavy traffic between a 'run' process and it's 'X11 Server' improves network performance.

Large-Scale Multi-User Configuration:





This configuration illustrates how GARDEN applications may be a distributed (via mid-range departmental UNIX servers) to large numbers of non-UNIX delivery platforms.

Note: The above configurations are just examples, practical configurations may mix UNIX and non-UNIX delivery platforms as required. Also, note that NFS remote file system mounting, symbolic links and multiple file server support provides another layer of configurations possibilities on top of those shown here.

APPENDIX A. BNF NOTATION

BNF notation uses a small number of symbols to represent the syntax of a formal language.

Optional items are enclosed in brackets, eg:

[X]

An item that may have zero, one or multiple occurances is indicated by brackets and a star, eg:

[X]*

A choice between a number of items is indicated by a bar '|' separated list enclosed in braces, eg:

{ A | B | C }

Terminals (literal occurances) are enclosed in single quotes, eg:

'0'
'bold'

A range of terminals is indicated using '..', eg:

'0'..'9'.
'A'..'Z'

Non-terminals are indicated by identifiers, eg:

digit
letter

Non-terminals are defined using the ::= symbol, eg:

number ::= digit [digit]*

APPENDIX B. GCML SYNTAX

BNF syntax for the GARDEN Content Markup Language:-

content::=

[{ text | font_control | active_text | image | line_break
begin_reference | end_reference | sub_title }]*

font_control::=

'{'['-']font_attribute'}'

font_attribute::=

{ 'bold' | 'italic' | 'underline' | 'super' | 'sub' | 'serif' }

active_text::=

'<'key'>'

image::=

'#'filename'#'

line::=

'##'

begin_reference::=

'#begin|'key'|'comment'#'

end_reference::=

'#end|'key'#'

sub_title::=

'<<'text'>>'

==========

APPENDIX C. GIML SYNTAX

==========

BNF syntax for the GARDEN Index Markup Language:-

index_content::=

[index_entry]*

index_entry::=

'##' key['|'key]* '##' definition

definition::=

[{ text | font_control | active_text }]*

font_control::=

'{'['-']font_attribute'}'

font_attribute::=

{ 'bold' | 'italic' | 'underline' | 'super' | 'sub' | 'serif' }

active_text::=

'<'key'>'

==
APPENDIX D. IMAGE FILE FORMAT
==

All GARDEN images files have the following format:

name	byte offset	number of bytes
magic number	0	2
transparent color	2	2
width	4	4
height	8	4
image data	12	'width' * 'height'

The image data is processed to use the standard GARDEN palette.

TRANSLATING IMAGES FORMATS

The 'pcx2g' program processes image colours so that all images share the same 8 bit colour palette (essential if more than one image is to be displayed at a time).

The program 'pcx2g' converts 256 color PCX format image files to the standard GARDEN image format. eg:

pcx2g fred.pcx

reads the PCX file fred.pcx and produces a new file called fred.i.

If pcx2g is run without a filename argument then it will prompt for the information it requires. This way of using pcx2g also enables the user to specify scaling and transparency parameters.

In order to support a wide range of display sizes, GARDEN can perform runtime image scaling, but the processing involved introduces slight delays. In situations where the majority of delivery platforms have the same size display these delays can be eliminated by pre-scaling images to the exact size. (This is, of course, not necessary when the original images already have the correct dimensions - in practice, this is suprisingly rare.)

Note: The development of additional file translation programs would enable image file formats other than PCX to be used, eg TIFF, GIF, PICT, etc.

==

========
APPENDIX E. STANDARD PALETTE
==========
========

Group	Range	Number	Decscription
A	0..11	12	Copied from default (Window Manager) colormap.
B	12..221	216	HSI color model.
C	222..234	13	Miscellaneous colors not in HSI model.
D	235..253	19	Grey-scale (black..17 greys..white).
E	254..255	2	Copied from default (Window Manager) colormap.

Hue-Saturation-Intensity (HSI) color model:

7 Hues

0	blue	(r=0%, g=0%, b=100%)	cold
1	cyan	(r=0%, g=100%, b=100%)	
2	green	(r=0%, g=100%, b=0%)	
3	yellow	(r=100%, g=100%, b=0%)	
4	orange	(r=100%, g=50%, b=0%)	
5	red	(r=100%, g=0%, b=0%)	
6	magenta	(r=100%, g=0%, b=100%)	warm

3 Saturations

0	100%	(pure color)	pure
1	40%	(pure color + 60% pastel)	
2	15%	(pure color + 85% pastel)	pale

10 Intensities

0	10%	dark
1	20%	
2	30%	
3	40%	
4	50%	medium
5	60%	
6	70%	
7	80%	
8	90%	
9	100%	bright

Color index = 10 * H + 30 * S + I + 12

Example color index calculations:

Q. What is the color index of pure bright blue?

A. First work out the HSI numbers:

H = 0 (blue)
S = 0 (pure)

I = 9 (bright)

Then do the sum:

10 * 0 + 30 * 0 + 9 + 12

The color index is therefore:

21

Q. What is the color index of pale medium red?

A. First work out the HSI numbers:

H = 5 (red)
S = 2 (pale)
I = 4 (medium)

Then do the sum:

10 * 5 + 30 * 2 + 4 + 12

The color index is therefore:

126

==========

APPENDIX F. GARDEN ENVIRONMENT VARIABLES

==========

$DISPLAY Name of X11 server display (default :0.0).

$GARDENFLAGS Default if undefined is "D+V+F-L-S+"
Syntax: [<char>{+|-}]*
D = double buffering (default on)
V = venetian blind effect (default on)
F = filler at end of topic (default off)
L = full logging (default off)
S = Sun bug fixes (default on)
+ = on, - = off
eg If X11 server is not a Sun:
setenv GARDENFLAGS S-
will fix the misaligned font problem.
eg To turn off the venetian blind effect and turn on
the stick tree effect used as a filler at the end of
topics:
setenv GARDENFLAGS V-F+

$GARDENHOST Name of GARDEN server host (default is local host).

$GARDENKEY Encrypted security key for $GARDENHOST.

$GARDENPATH Directory containing all GARDEN files.

$GARDENTAPE Name of transfer device (eg /dev/rst0).

APPENDIX G. GARDEN PROGRAMS

GARDEN consists of the following programs (in $GARDENPATH/bin):

garden
: Launches the GARDEN server.

pcx2g files
: Translates PCX format image files into GARDEN format.

compile application_name
: Required after editing an application.

run application_name [width height]
: Executes an application.

APPENDIX H. PRESENTATION CONTROL FILES

Presentation Control Files control the visual appearance of a GARDEN application. Below is a full alphabetically sorted list of the files:

arbitary

Applies only to "menu" modules. An ASCII text file containing numeric parameters for arranging icons. See ICON LAYOUT for more information.

back.i

An image file containing a module's background image. If omitted then GARDEN searches up the resource tree until a "back.i" file is found (this is known as inheriting a resource). If the module is at the root of the application's "resources" tree then it is used as the background for the fixed icon

menu, otherwise it is used as the background for the work area.

blue.i

Applies only to the "index" module. An image file containing the blue paper texture for the index cards. The image used must tile seamlessly.

box.i

Applies only to the "index" module. An image file containing the index box icon. See ICON FORMAT for more information.

circular

Applies only to "menu" modules. An ASCII text file containing numeric parameters for arranging icons. See ICON LAYOUT for more information.

color.code

An ASCII text file containing the color number to use for the lines drawn on content pages. If omitted then GARDEN searches up the resource tree until a "color.code" file is found. If no file is found then the default of 235 (black) is used.

column.width

An optional ASCII text file containing the minimum column width in pixels. If this file is omitted then 280 is used as the default.

content

Applies only to "content" (topic) modules, ie modules not containing sub-modules. An ASCII text file containing the module's textual content with optional "mark-up" notation. See CONTENT FORMAT for more information.

footnote

Applies only to "content" modules. An ASCII text file containing a short line of text to use as a footnote. If omitted then GARDEN uses the application's name and the module's ID as a default footnote.

green.i

Applies only to the "index" module. An image file containing the green paper texture for the index cards. The image used must tile seamlessly.

horizontal

Applies only to "menu" modules. An ASCII text file containing numeric parameters for arranging icons. See ICON LAYOUT for more information.

icon.i

Each module (except the one at the root of the resources tree) is represented as a menu icon in it's parent module. An icon.i file is an image file containing the module's icon. Every module must contain an "icon.i" file. See ICON FORMAT for more information.

icon.label

An optional ASCII text file containing parameters for the placement and background color of an icon text label. The first parameter is one of the following words "above", "right", "below", "left". The second parameter is a color number to use for the background of the text label (zero indicates no background). egs:

right 235
-- indicates that the label should be on right of the icon
-- and have a black background color

below 0
-- indicates that the label should be below the icon
-- and have no background

If this file is omitted the icon does not have a text label added.

iconv.i

An optional image file containing a variant of the module's "icon.i" file. The presence of an "iconv.i" file indicates that the module's icon should provide feedback about the proportion of the module and it's sub-modules that have been visited by the current user. This feedback is in the form of a pie segment copied from "iconv.i" instead of "icon.i". Note "iconv.i" is usually the same graphic as "icon.i" but with much brighter colors. See ICON FORMAT for more information.

layout

Applies only to the root module of the application's "resources" tree, ie the "resources" directory itself. An ASCII text file containing parameters for the placement and size of fixed icon menu. The first parameter is one of the following words "top", "right", "bottom", "left". The second parameter is floating point number representing the proportion of the display area to use for the fixed icon menu. A value of 0.05 means 5%, 0.1 means that 10%, 0.2 means 20% and so on. eg:

left 0.1
-- indicates that the fixed icon menu should be on the left
-- and occupy 10% of the display's width

bottom 0.15
-- indicates that the fixed icon menu should be at the bottom
-- and occupy 15% of the display's height

If this file is omitted then "left 0.1" is used as a default.

logo.i

Applies only to the root module of the application's "resources" tree, ie the "resources" directory itself. An image file containing the FIRST full display background image used when a GARDEN

Client is executed. (See "title.i")

next.i

Applies only to the root module of the application's "resources" tree. An image file containing the "next page" icon. See ICON FORMAT for more information.

paper.i

An image file containing a paper texture for a module's "content" pages. If omitted then GARDEN searches up the resource tree until a "paper.i" file is found.

previous.i

Applies only to the root module of the application's "resources" tree. An image file containing the "previous page" icon. See ICON FORMAT for more information.

radial

Applies only to "menu" modules. An ASCII text file containing numeric parameters for arranging icons. See ICON LAYOUT for more information.

red.i

Applies only to the "index" module. An image file containing the red paper texture for the index cards. The image used must tile seamlessly.

return.i

Applies only to the "index" module. An image file containing the "return" icon. See ICON FORMAT for more information.

title

An ASCII text file containing the title of a module. This text is used for the title on content pages and also to label icons (if the file "icon.label" exists). Content page titles are always displayed on a single line, however, icon labels may be displayed as several (centre justified) lines. Vertical bar characters '|' may be used to mark points where the title should be split when used as an icon label, eg:

"This is a|rather long|title" --> This is a
rather long
title

title.i

Applies only to the root module of the application's "resources" tree, ie the "resources" directory itself. An image file containing the SECOND full display background image used when a GARDEN Client is executed. (See "logo.i")

vertical

Applies only to "menu" modules. An ASCII text file containing numeric parameters for arranging icons. See ICON LAYOUT for more information.

APPENDIX I. INTERACTION CONTROL FILES

Interaction Control Files control the behaviour of a GARDEN application. Below is a full alphabetically sorted list of the files:

help

An empty file indicating that the module is the "help" module. Only one module per application should contain this file, and it must be a direct sub-module of the root of the resources tree. This module will always be presented first to new users.

index

An empty file indicating that the module is the "index" module. Only one module per application should contain this file, and it must be a direct sub-module of the root of the resources tree.

main.menu

An empty file indicating that the module is the "main menu" module. Only one module per application should contain this file, and it must be a direct sub-module of the root of the resources tree.

parent.holder

An empty file indicating that the module is the "parent holder" module. Only one module per application should contain this file, and it must be a direct sub-module of the root of the resources tree. The icon for the module containing this file causes the parent of the current module to be displayed when selected.

quit

An empty file indicating that the module is the "quit" module. Only one module per application should contain this file. The icon for the module containing this file causes the GARDEN client to exit when selected.

require

An optional ASCII text file specifying the prerequisites of a module, ie which module(s) must be completed by the current user before they are allowed to access the module containing this file. The file should contain either the word "previous" or one or more valid Module IDs.

The "previous" option covers the common case where a module requires the previous module in a series to be completed, eg if a module with an ID of 2.4 has a "require" file containing the word

"previous", then the module may only be accessed after the user has completed a module with an ID of 2.3).

The module ID option enables arbitary prerequisite relationships to be established. To prevent "deadlock" situations (where module A requires module B and module B requires module A) the module ID in a "require" file must be less than the ID of the module containing the file, eg 2.4 can require 2.3 but 2.3 can not require 2.4.

section.holder

An empty file indicating that the module is the "section holder" module. Only one module per application should contain this file, and it must be a direct sub-module of the root of the resources tree. The icon for the module containing this file causes the most recently visited "section menu" to be displayed when selected. A section menu is any menu that is a direct sub-module of the "main menu" module.

Note that "parent.holder" should be used instead of "section.holder" in large applications.
lff
==========
APPENDIX J. LOG FILE FORMATS
==========

User Logs

Each user-application pairing has a unique user log file:

$GARDENPATH/application_name/login_name

User log files are simply ASCII text files, and may be printed out or read into a database package for full analysis.

Entries in the log are of a fixed format and occupy one line each. A typical log extract looks like:

```
LOGIN from bentleysun1:0.0 @ Tue Jun  7 16:48:57 1994
*1_1            @ 16:49:06
*1_2            @ 16:49:06
*2              @ 16:49:10
*2.1            @ 16:49:16
*2.1.1          @ 16:49:21
*2.1.1_1        @ 16:49:22
*2.1.1_2        @ 16:49:22
*2.1.2          @ 16:49:27
*2.1.2_1        @ 16:49:27
*2.2            @ 16:49:32
*2.2.1          @ 16:49:34
*2.2.1_1        @ 16:49:34
*2.2.1_2        @ 16:49:34
*2.2.1_3        @ 16:49:38
*2.2.1_4        @ 16:49:38
*2.2.1_5        @ 16:49:39
```

*2.2.1_6 @ 16:49:39
*2.2.1_7 @ 16:49:39
*5 @ 16:49:50
LOGOUT @ Tue Jun 7 16:49:53 1994 duration 56 sec

Whenever a user runs an application a LOGIN entry is recorded:

LOGIN from display_name @ time_date_stamp

Whenever a user quits an application a LOGOUT entry is recorded:

LOGOUT @ time_date_stamp duration period

Whenever a user views a new piece of information an entry is recorded

*id @ time_stamp

In addition, if full logging is enabled (see GARDENFLAG environment variable) then whenever a user views a piece of information subsequently then an entry is recorded:

id @ time_stamp

Note that only those entries beginning with a star (*) are needed in order to determine how much of an application a user has navigated. For this reason, and to economise on disk space, GARDEN only performs partial logging by default.

The id used in these entries relates directly to the module id of the content viewed. So the entry

*2.2.1_3 @ 16:49:38

can be interpreted as the 3rd item in module 2.2.1.

The GARDEN compiler produces a mapping between module id and titles. To place this information in an ASCII text file called table enter the following:

compile application_name > table

Server Log

The GARDEN server keeps it's own log file:

$GARDENPATH/server.log

The server log file is simply an ASCII text file, and may be printed out or read into a database package for full analysis.

Entries in the log are of a fixed format and occupy one line each. A typical log extract looks like:

START-UP
Host = bentleysun1

```
        $GARDENPATH = /external/colin/GARDEN.1.4
        User = colin
        Time = Tue Jun  7 16:48:51 1994
LOGIN
        Student = colin
        Course = manual
        Display = bentleysun1:0.0
        Time = Tue Jun  7 16:48:57 1994
LOGOUT
        Student = colin
        Course = manual
        Display = bentleysun1:0.0
        Time = Tue Jun  7 16:49:53 1994
        Duration = 56 sec
```

Whenever the GARDEN server is launched or a user starts or quits an application an entry is made in the server log.

An entry of the form "SECURITY ALERT: $GARDENKEY incorrect!" may occur if the GARDEN server is launched on an unauthorised host.

The server log may be safely deleted if disk space becomes a problem.

END OF FILE
