ColorSelect Methods

```
/Action        % color --
/SetValue      % color --
```

C Unix Commands

drawps(1)

NAME

drawps

Convert HyperLook drawings to PostScript format.

SYNOPSIS

drawps [-epsf] [-scale f] [-hdr hdr] [file]

DESCRIPTION

drawps is a filter to translate HyperLook drawings produced with the **hyperdraw(1)** graphics editor into PostScript or Encapsulated PostScript format.

Input is taken from the standard input, or from the specified file. The output is produced on the standard output.

OPTIONS

-epsf

Generate Encapsulated PostScript headers. These files can be imported by other programs.

-scale f

Scale the drawing by the indicated factor. The factor must be a floating point number greater than 0.

-hdr hdr

Use this header when generating Encapsulated PostScript files. Some programs insist on a particular header string. The default is:

```
!PS-Adobe-2.0 EPSF-1.2
```

EXAMPLES

To print a drawing at 50% magnification use:

```
drawps -scale 0.5 logo.draw | lpr -Plw
```

ENVIRONMENT

drawps requires that the **$HLHOME** environment variable is set.

FILES

$HLHOME/lib/ps/drawlib.ps
PostScript printer initialization file.

$HLHOME/lib/ps/lib.ps
Initialization file.

SEE ALSO

hyperdraw(1), **hyperlook(1)**.

exithyperlook(1)

NAME

exithyperlook

Exit HyperLook.

SYNOPSIS

```
exithyperlook [ -f ]
```

DESCRIPTION

The **exithyperlook** command exits the HyperLook system. The **ExitHyperLook** stack is shown to let the user save stacks before really exiting.

OPTIONS

-f

Exit without asking. The user gets no chance to save stacks.

EXAMPLES

To restart HyperLook quickly without asking the user:

```
exithyperlook -f ; hyperlook -c
```

SEE ALSO

hyperlook(1).

hlpath(1)

NAME

hlpath

Manage HyperLook resource directories.

SYNOPSIS

hlpath [-a] [-r] [-ra] dir ...

DESCRIPTION

The **hlpath(1)** command adds or removes user or application resource directories to HyperLook. The default is to add user resource directories.

HyperLook resource directories are searched when looking for resources such as stacks. The default user resource directory is **~/stacks**.

OPTIONS

-a

Add the specified application resource directories.

-r

Remove the specified user resource directories.

-ar

Remove the specified application resource directories.

FILES

HyperInit.ps

This PostScript file is executed if it exists in the resource directory when it is added to the HyperLook resource directory list. It allows applications to perform initializations.

./mono

When a resource directory is added when HyperLook is running on a monochrome display, and the resource directory has a sub directory called **mono**, then this directory is added before the resource directory itself. This allows you to design interfaces for both monochrome and color screens.

SEE ALSO

hyperlook(1).

hlpsh(1)

NAME

hlpsh

The HyperLook version of **psh(1)**.

SYNOPSIS

hlpsh [-s] [-i] file ...

DESCRIPTION

hlpsh uses **psh(1)** to interpret PostScript commands from the specified files or from standard input. The PostScript environment is setup so that the HyperLook dictionary is on the top of the dictionary stack.

If HyperLook is not running, it will be started.

OPTIONS

-s

Work silently. If HyperLook is not running, exit immediately.

-i

Run in interactive mode. Report errors to the current terminal instead of the console.

EXAMPLES

This example illustrates the loading and displaying of a stack from PostScript:

```
hlpsh -i
Welcome to HyperLook version 1.5 (TNT 2.0)
/system FindStack dup =
dictionary[40/50000]
ShowStack
^D
```

SEE ALSO

hyperlook(1), **psh(1)**.

hlsend(1)

NAME

hlsend

Send a message to an object in HyperLook.

SYNOPSIS

hlsend object message arg ....

DESCRIPTION

hlsend uses **hlpsh(1)** to send a message to an object in HyperLook. The object is specified either as a stack name, or as **stackname:objectname**. The message is specified as a single word, without the leading slash. The message arguments must be in PostScript format.

EXAMPLE

To move the system stack use:

```
hlsend system Move 100 100
```

To set the value of a field in a stack use:

```
hlsend SystemStatus:status SetValue "(Hello world)"
```

SEE ALSO

hyperlook(1), **hlpsh(1)**.

hyperdraw(1)

NAME

hyperdraw

HyperLook graphics editor.

SYNOPSIS

hyperdraw [filename]

DESCRIPTION

HyperDraw is a PostScript graphics editor. It is part of the HyperLook system.

The **hyperdraw** command starts HyperLook if needed. If a drawing file is specified, it is loaded into the editor. The file name must end in **.draw**.

EXAMPLES

To edit a drawing from the HyperLook home directory use:

```
hyperdraw $HLHOME/draw/garfield.draw
```

SEE ALSO

hyperlook(1).

hyperlook(1)

NAME

hyperlook

install_hyperlook

HyperLook startup script.

SYNOPSIS

hyperlook [-c] [-fix]
install_hyperlook

DESCRIPTION

HyperLook is an object oriented PostScript based user interface development system. It uses the stack and card model for the design of user interfaces by direct manipulation. User interface components can be programmed using PostScript as a scripting language.

The **hyperlook** command starts HyperLook; no action is taken if HyperLook is already running. OpenWindows is started if it is not yet running.

Use **install_hyperlook(1)** to install the HyperLook icons in the **filemgr(1)** data base.

OPTIONS

-c

Startup quickly, not showing the introduction stack.

-fix

Restart and fix things when HyperLook has crashed (not that it has ever happened).

ENVIRONMENT

The **hyperlook(1)** startup script recognizes the following shell variables. Some of these are used when starting OpenWindows.

HLHOME
The HyperLook home directory.

OPENWINHOME
The OpenWindows home directory.

OPENWINOPTIONS
Options passed to OpenWindows at startup. See **openwin(1)**.

DISPLAY
The display on which HyperLook should run. HyperLook will use the color framebuffer if one is available.

FILES

~/stacks
The default directory where user resource files are saved. The most common resource file is a **.stack** file containing a HyperLook stack.

~/stacks/.hyperlook
The file where the HyperLook user preferences are saved.

~/stacks/.hyperlook-init
This file is executed at startup. It should only contain **sh(1)** commands to show stacks and perform initializations.

~/stacks/HyperInit.ps
This PostScript file is executed when the **~/stacks** directory is added by the resource manager. See also **hlpath(1)**.

SEE ALSO

exithyperlook(1), **hidestack(1)**, **hlpath(1)**, **hlsend(1)**, **hlpsh(1)**, **hyperdraw(1)**, **install_hyperlook(1)**, **showstack(1)**.

DIAGNOSTICS

Make sure that you don't have any dangerous files in your home directory when starting HyperLook. Remove **.xinitrc**, **.startup.ps**, **.user.ps**, **.init.ps**, if you have problems starting HyperLook.

Start OpenWindows first and start HyperLook using the **hyperlook(1)** command from the console, if all else fails.

showstack(1)

NAME

showstack

hidestack

Show and hide HyperLook stacks and objects.

SYNOPSIS

```
showstack stack
hidestack stack
```

DESCRIPTION

The **showstack** command displays a HyperLook stack (window) and brings it to the front. If the stack is iconified, the icon is opened. The **hidestack** command hides a stack.

The stack is specified either by its name (without the **.stack** file name extension) or by using its file name. If the stack is specified as a name it must be accessible by the HyperLook resource manager. If the stack is specified as a file name its directory will be added to the user resource directory list.

The stack is loaded from a resource directory if it is not already in memory. See **hlpath(1)** on how to add a resource directory.

SEE ALSO

hlpath(1), **hyperlook(1)**.

DIAGNOSTICS

The **hidestack** command loads the stack before hiding it.

D

Library Functions

hl_alloc(3)

NAME

hl_alloc
hl_destroy
hl_use
hl_free
hl_constant
Manage **hl_any** data structures.

SYNOPSIS

```
#include <hyperlook.h>

void *hl_alloc(bytes)
int bytes;

void hl_destroy(ptr)
void *ptr;

void hl_use(any)
hl_any *any;

void hl_free(any)
hl_any *any;

void hl_constant(any)
hl_any *any;
```

DESCRIPTION

The **hl_alloc()** and **hl_destroy()** routines are used for low level memory management. You can redefine these routines in your program to use your own memory manager.

The **hl_use()** routine increments the reference count of an **hl_any** data structure. **hl_free()** decrements the reference count of an **hl_any** data structure. If it reaches **0** the data structure is actually deleted. **hl_constant()** freezes the reference count of an **hl_any** data structure so that it is never deleted.

SEE ALSO

hyperlook(1), **hl_any(3)**, **hl_send(3)**.

hl_any(3)

NAME

hl_any
hl_new_null
hl_new_boolean
hl_new_number
hl_new_string
hl_new_string_body
hl_new_name
hl_new_array
hl_new_array_body
Create **hl_any** data structures.

SYNOPSIS

```
#include <hyperlook.h>

hl_any *hl_new_null()

hl_any *hl_new_boolean(flag)
int flag;

hl_any *hl_new_number(n)
float n;

hl_any *hl_new_string(str)
char *str;

hl_any *hl_new_string_body(len)
int len;

hl_any *hl_new_name(str)
char *str;

hl_any *hl_new_array(obj0, ..., NULL)
hl_any *obj0, ...;

hl_any *hl_new_array_body(len)
int len;
```

DESCRIPTION

The **hl_new** routines allocate **hl_any** data structures which are used to represent data for communication with HyperLook. The **hl_any** data structure is defined in the file **include/hyperlook.h** in the HyperLook directory.

The **hl_new_null()**, **hl_new_boolean()**, **hl_new_number()**, **hl_new_name()**, **hl_new_string()** routines allocate a data structure containing the appropriate data type.

hl_new_string_body() allocates a data structure containing a string of length **len** containing **'\0'** characters.

hl_new_array() allocates an array of **hl_any** objects. The members of the array are specified in the argument list which must be terminated by a **NULL**. **hl_new_array_body()** allocates an array of length **len** containing **NULL** pointers.

The **hl_any** data structure is defined as:

```
typedef struct hl_any {
   hl_any_type type;
   short len;
   union {
       double number;
       int boolean;
       char *string;
       char *name;
       struct {
           int argc;
           struct hl_any **argv;
       } array;
   } u;
} hl_any;
```

The **len** field is the length of strings and names. The **type** field indicates how the other fields are used. It is of type **hl_any_type**:

```
typedef enum {
    hl_number_type,
    hl_boolean_type,
    hl_string_type,
    hl_name_type,
    hl_array_type,
    hl_null_type,
} hl_any_type;
```

EXAMPLES

Create an array of objects which can be sent to HyperLook:

```
hl_any *a = hl_new_array(
    hl_new_number(30), hl_new_number(40.5),
    hl_new_name("Dug"), hl_new_string("Scoular"),
    hl_new_array(hl_new_boolean(1),NULL),
    NULL);
```

The PostScript version of the array is:

```
[30 40.5 /Dug (Scoular) [true]]
```

Elements of the array can be accessed using the **hl_any** data structure:

```
printf("n = %d\n",a->u.array.argc);
printf("x = %g\n",a->u.array.argv[0]->u.number);
printf("y = %g\n",a->u.array.argv[1]->u.number);
printf("name = %s %s\n",a->u.array.argv[2]->u.name,
    a->u.array.argv[3]->u.string);
```

FILES

include/hyperlook.h
: The include file containing all the HyperLook client interface definitions.

SEE ALSO

hyperlook(1), **hl_print(3)**, **hl_free(3)**, **hl_use(3)**, **hl_send(3)**.

hl_exists(3)

NAME

hl_exists

Check the existence of a HyperLook object.

SYNOPSIS

```
#include <hyperlook.h>

int hl_exists(target)
char *target;
```

DESCRIPTION

hl_exists() checks the existence of a HyperLook object. The object is addressed in the same way as **hl_send(3)**.

EXAMPLES

The following program checks for the existence of an object called "**name**" on a stack called "**entry**".

```
if (hl_exists("entry:name")) {
    /* it's there! */
    ...
}
```

RETURN VALUES

If the object exists, **1** is returned. If not, or if the connection to HyperLook is lost, **0** is returned.

SEE ALSO

hyperlook(1), **hl_send(3)**.

hl_flush(3)

NAME

hl_flush

hl_flush_input

Flush HyperLook input and output.

SYNOPSIS

```
#include <hyperlook.h>

void hl_flush()

void hl_flush_input()
```

DESCRIPTION

hl_flush() flushes all pending output to HyperLook. Output is automatically flushed before reading input (in **hl_listen(3)**).

hl_flush_input() causes all pending input to be discarded.

SEE ALSO

hyperlook(1), **hl_start(3)**, **hl_listen(3)**.

hl_get(3)

NAME

hl_get

hl_put

Access to instance variables.

SYNOPSIS

```
#include <hyperlook.h>

hl_any *hl_get(target,param);
char *target, *param;

int hl_put(target,param,value);
char *target, *param;
hl_any *value;
```

DESCRIPTION

hl_get() provides direct access to instance variables of HyperLook objects. The **target** specifies the object in the same way as the first argument to **hl_send(3)** does.

The **param** argument is the name of the variable which is accessed. **hl_get** returns an **hl_any** data structure containing the variable's value. This data structure must be de-allocated using **hl_free(3)**.

hl_put() assigns a new value to an instance variable of a HyperLook object. The new value is specified as an **hl_any** data structure which must be de-allocated using **hl_free(3)** if it is not used again.

RETURN VALUES

hl_get() returns a **NULL** pointer when an error occurs or if HyperLook is not started. It returns **hl_null** if the parameter does not exist.

hl_put() returns **1** on success, and **0** on failure. Failure only occurs if the connection to HyperLook is lost.

EXAMPLES

Get the "**Editable?**" parameter of a text object called "**name**" on a stack called "**entry**":

```
hl_any *b = hl_get("entry:name","Editable?");
if ((b->type == hl_boolean_type) &&
    (b->u.boolean != 0)) {
   ...
}
hl_free(b);
```

Set the "**Editable?**" flag of the same object:

```
hl_any *b = hl_new_boolean(1);
hl_put("entry:name","Editable?",b);
hl_free(b);
```

SEE ALSO

hyperlook(1), **hl_start(3)**, **hl_send(3)**.

DIAGNOSTICS

hl_get() and **hl_put()** provide unrestricted access to HyperLook objects. This means that they can easily be misused. Use them with care!

hl_listen(3)

NAME

hl_listen
hl_register
hl_register_any
hl_register_quit
hl_register_timeout
hl_register_ioerror
HyperLook message handling.

SYNOPSIS

```
#include <hyperlook.h>

int hl_listen(delay)
int delay;

void hl_register(handler,obj,message)
int (*)() handler;
char *obj;
char *message;

void hl_register_any(handler,match)
int (*)() handler;
hl_any *match;

void hl_register_quit(handler)
int (*)() handler;

void hl_register_timeout(handler)
int (*)() handler;

void hl_register_ioerror(handler)
int (*)() handler;
```

DESCRIPTION

hl_listen() is the main HyperLook event loop. It reads messages from HyperLook and calls the appropriate handlers.

hl_register(), **hl_register_any()**, **hl_register_timeout()**, and **hl_register_ioerror()** install handlers for incoming messages. The handlers are called when a message is received from HyperLook.

The **delay** argument to **hl_listen** is a delay in milliseconds (1000ms is 1 second). If no input is received for **delay** milliseconds, an **hl_timeout** message is generated. You can use **hl_forever**, to make it never time out.

hl_listen doesn't return upon timeout. It calls the timeout handler. Any handler can return **-1** to cause **hl_listen** to return.

The handler argument to the **hl_register()** and **hl_register_any()** is a function returning an integer. Here is an example of a handler function:

```
int button_pressed(message,argc,argv)
hl_any *message;
int argc;
hl_any *argv[];
{
    ....
    return 1;
}
```

When handler returns **0** the message is matched against other handlers. The handler should return **1** if the message is handled and no other handlers should be considered. When the handler returns **-1** the **hl_listen()** event loop returns to its caller.

The **obj** argument of **hl_register()** specifies which object the message responds to. The object can be specified as follows:

```
ANY -- any object
"*" -- any object
"x" -- stack x
"x:*" -- any object on stack "x"
"x:y" -- an object "y" on stack "x"
"*:y" -- any object called "y"
"Button(*)" -- any button
"Button(y)" -- any button called "y"
"x:Button(*)" -- any button on stack "x"
"x:Button(y)" -- a button called "y" on stack "x"
```

In the above examples **Button** is used as an object type. You can use other object types like **Field** and **Slider**, too.

The message argument of **hl_register()** specifies which message is allowed. Any message is matched if you specify the message as **ANY**.

hl_register_any() lets you register for any type of input data. The **match** data structure is matched against the input. Any part of the data structure may contain **ANY** which matches anything.

hl_register_quit() lets you register a handler for **hl_quit** messages. An **hl_quit** message is generated when the connection with HyperLook is lost.

hl_register_ioerror() lets you register a handler for **hl_ioerror** messages. An **hl_ioerror** message is generated when invalid data is read from HyperLook.

RETURN VALUES

hl_listen() returns **1** on normal exit. When an **hl_ioerror** is found it returns **-1**.

EXAMPLES

This is an example of a handler for a Field object. It is registered as:

```
hl_register(search_handler,"*:Field(x)","Action");
```

The **Action** message of a field has one argument (the string). The string is accessed through the **hl_any** data structure.

```
int search_handler(message,argc,argv)
hl_any *message;
int argc;
hl_any *argv[];
{
   char *s = argv[0]->u.string;
   /* start search */
   ....

   return 1;
}
```

SEE ALSO

hyperlook(1), **hl_start(3)**.

DIAGNOSTICS

hl_listen() generates an **hl_quit** message when the connection with HyperLook is lost. An **hl_ioerror** is generated if an I/O error occurs.

hl_path(3)

NAME

hl_path

Register a resource directory.

SYNOPSIS

```
#include <hyperlook.h>

int hl_path(dir)
char *dir;
```

DESCRIPTION

hl_path() registers an application resource directory with the HyperLook resource manager. Once registered, the application can access resources (stacks) in the directory.

The argument **dir** is the full path name of the resource directory. It can be **NULL**, meaning the current directory.

EXAMPLES

The following main program shows a stack which has been saved in the directory from which the program is executed.

```
#include <hyperlook.h>
main()
{
    hl_start("myclient");
    hl_path(NULL);
    hl_show("mystack");
    hl_stop();
}
```

RETURN VALUE

hl_path() returns **1** on success, **0** if the connection to HyperLook is lost.

SEE ALSO

hyperlook(1), **hl_start(3)**, **hl_show(3)**.

hl_print(3)

NAME

hl_print

Print an **hl_any** data structure to a file.

SYNOPSIS

```
#include <hyperlook.h>

void hl_print(fp,any)
FILE *fp;
char *prog;
```

DESCRIPTION

hl_print() prints the **hl_any** data structure specified in **any** to the **fp** file in human readable format. This function should be used for debugging only.

SEE ALSO

hyperlook(1), **hl_verbose(3)**.

hl_ps(3)

NAME

hl_ps

Execute raw PostScript in HyperLook.

SYNOPSIS

```
#include <hyperlook.h>

int hl_ps(code)
char *code;
```

DESCRIPTION

hl_ps() sends a string containing PostScript commands to HyperLook to be executed in the server. You need to call **hl_flush(3)** to flush the output stream after calling **hl_ps()**.

EXAMPLES

The following PostScript example shows the system stack using PostScript only (you can also use **hl_show(3)**).

```
hl_ps("/system FindStack ShowStack");
```

RETURN VALUES

hl_ps() returns 1 on success, 0 if the connection to HyperLook is lost.

SEE ALSO

hyperlook(1), **hl_flush(3)**.

hl_send(3)

NAME

hl_send

hl_send0

Send messages to HyperLook objects.

SYNOPSIS

```
#include <hyperlook.h>

int hl_send(target,message,args)
char *target, *message;
hl_any *args;

int hl_send0(target,message,arg0,....,NULL)
char *target, *message;
hl_any *arg0, ...;
```

DESCRIPTION

The **hl_send()** and **hl_send0()** routines send a message to a HyperLook object. The **message** argument is the name of the message. The **target** argument is the name of the object to which the message should be sent. For example:

```
"x" - a stack called "x"
"x:y" - an object "y" on stack "x"
```

The arguments to **hl_send()** are specified in **args** as an **hl_any** array. The array can be allocated using **hl_new_array(3)**. The argument array must be de-allocated, after sending the message, using **hl_free(3)**.

The arguments to **hl_send0()** are specified in the argument list which must be **NULL** terminated. The arguments are de-allocated by **hl_send0()** using **hl_free()**.

Usually you should use **hl_send0()** to send messages to objects. It uses a more compact notation and it requires no explicit freeing of data structures. If you want to send the same message frequently, or if you want to avoid repeated memory allocations, the you should use **hl_send()**.

RETURN VALUES

Both **hl_send()** and **hl_send0()** return **1** if the message is sent successfully. They return **0** if HyperLook was not started or if the target is invalid.

EXAMPLES

Set the value of a text object called "**name**" on stack "**entry**":

```
hl_any *a = hl_new_array(
   hl_new_string("Some text"),
   NULL);
hl_send("entry:name","SetValue",a);
hl_free(a);
```

The same message can be sent using **hl_send0** as follows (note that the argument string is deallocated automatically):

```
hl_send0("entry:name","SetValue",
   hl_new_string("Some text"),
   NULL);
```

SEE ALSO

hyperlook(1), **hl_any(3)**.

hl_show(3)

NAME

hl_show

hl_hide

hl_connect

Show and hide stacks and objects, and connect to stacks.

SYNOPSIS

```
#include <hyperlook.h>

int hl_show(target);
char *target;

int hl_hide(target);
char *target;

int hl_connect(target);
char *target;
```

DESCRIPTION

The **hl_show()** and **hl_hide()** routines let you show and hide HyperLook objects and stacks. The argument is the name of a stack, like "**mystack**", or an object, like "**mystack:field**". Hiding a stack removes its window from the screen. Hiding an object makes it invisible.

hl_connect() connects the client to a stack directly (without showing it). **hl_show()** implicitly connects to the stack before it is shown. A client must be connected to a stack if it wants to receive messages from the stack. You can't connect to an object.

A client can be connected to any number of stacks. A stack can be connected to only one client.

RETURN VALUES

On success the routines return **1**, on failure **0**. Failure only occurs if the connection to HyperLook is lost.

EXAMPLES

To show a stack called "**entry**" use:

```
hl_show("entry");
```

To connect to the same stack without showing it:

```
hl_connect("entry");
```

To hide an object called "**search**" on stack "**entry**" use:

```
hl_hide("entry:search");
```

SEE ALSO

hyperlook(1), **hl_start(3)**, **hl_send(3)**, **hl_path(3)**.

DIAGNOSTICS

Stacks must be accessible by the resource manager before they can be shown. See **hl_path(3)**.

hl_start(3)

NAME

hl_start

hl_stop

hl_verbose

Connect to HyperLook.

SYNOPSIS

```
#include <hyperlook.h>

int hl_start(prog)
char *prog;

int hl_stop()

int hl_verbose;
```

DESCRIPTION

hl_start() attempts to establish a connection to HyperLook. It should be called before any other HyperLook client interface routine is used. **prog** is the name which is used by HyperLook to identify the client.

hl_stop() terminates the connection with HyperLook.

The **hl_verbose** flag lets you monitor message as they are received. If this flag is non zero message are printed to **stderr**. The default value is **0**.

RETURN VALUES

hl_start() returns **1** when a connection is successfully established, it returns **0** otherwise.

hl_stop() always returns **1**.

EXAMPLES

This is an example of a skeleton main program.

```
#include <stdio.h>
#include <hyperlook.h>

main(argc,argv)
int argc;
char *argv[];
{
   hl_verbose = 1;

   if (!hl_start(argv[0])) {
       fprintf(stderr,"%s: cannot connect to\
           HyperLook\n",
           argv[0]);
       exit(1);
   }

   /* main program */
   ....

   hl_stop();
   exit(0);
}
```

ENVIRONMENT

The HyperLook client interface is build on top of the NeWS wire service. The following environment variables are significant.

HLHOME
The HyperLook home directory.

DISPLAY
The display to which the client connects.

FILES

include/hyperlook.h
The include file containing all the HyperLook client interface definitions.

SEE ALSO

hyperlook(1), **hl_register(3)**, **hl_listen(3)**.

DIAGNOSTICS

`hl_start()` starts HyperLook. It does nothing if HyperLook is running.

libhl.a(3)

NAME

libhl.a

HyperLook client interface library.

DESCRIPTION

The C library **libhl.a** can be linked with any C program to communicate with HyperLook. It contains library routines which are defined in the file **include/hyperlook.h**.

See **hl_start(3)** and **hl_listen(3)** for information on how to use the main functions in this library.

EXAMPLES

To compile a C program called **try.c** you need to specify some include flags and library flags in the makefile:

```
CFLAGS= -I$(HLHOME)/include \
   -I$(OPENWINHOME)/include/wire \
   -I$(OPENWINHOME)/include/NeWS \
   -I$(OPENWINHOME)/include
LDFLAGS= -L$(HLHOME)/lib -lhl

cc -o try try.c $(CFLAGS) $(LDFLAGS)
```

The three OpenWindows include directories are not all necessary. This combination is chosen because it works for all OpenWindows versions.

FILES

include/hyperlook.h
The include file containing all the HyperLook client interface definitions.

lib/libhl.a
The library file containing the HyperLook client interface.

SEE ALSO

hyperlook(1), **hl_start(3)**.

HyperLook

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

S

T

U

V

W

X

Z