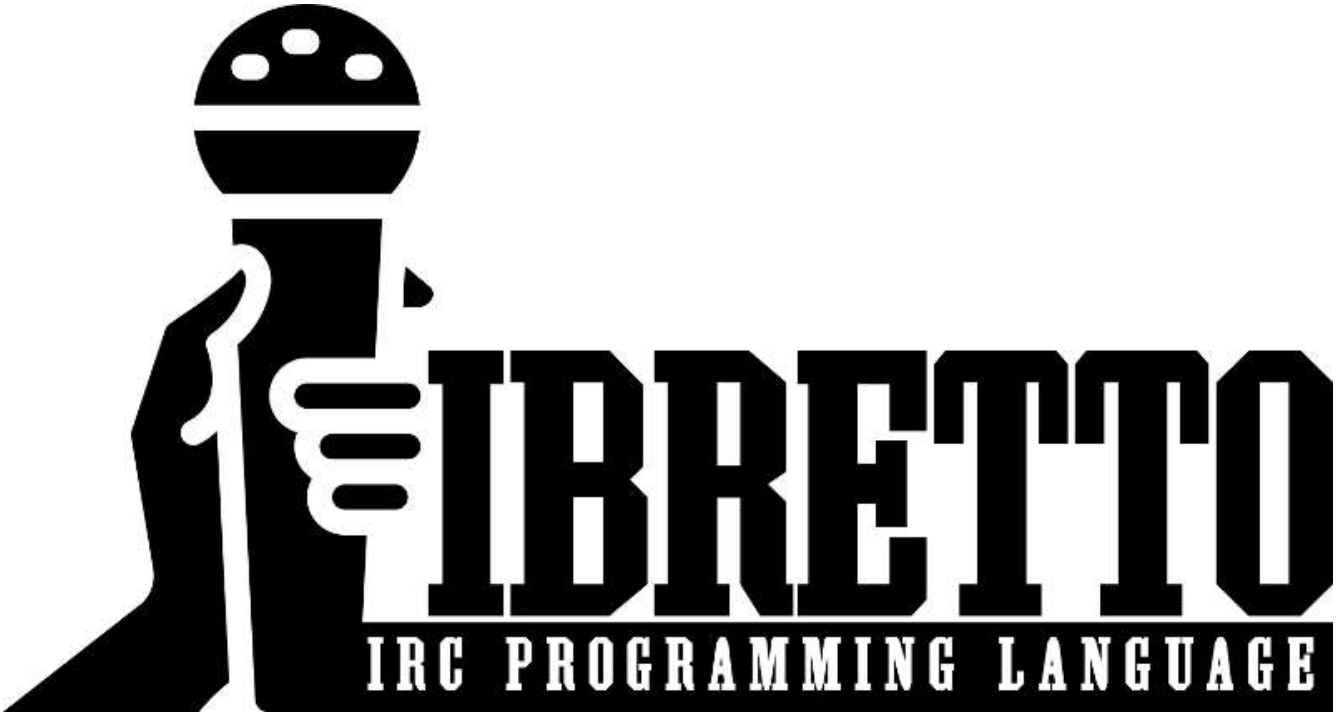

A Programming Language for Internet Relay Chat

Official Manual and Usage Guide

Version 0.08448

https://github.com/danhetrick

Alpha Release - "Aria"

Table of Contents

Usage

```
perl libretto.pl [OPTIONS] SCRIPT [ARGUMENTS]

--(h)elp                                Display usage information
--(v)erbose                             Turn on verbose mode
--(w)arn                                Turn on warnings mode
--(l)icense                             Displays license information
--(c)onfiguration FILE                  Sets the file to load settings from
--(f)iles DIRECTORY                     Sets the directory to place uploaded files
                                        and look for files for users to download
--generate-(C)onfig [FILE]              Generates a generic setting config file

Single letter options can be bundled; for example to turn on both
verbose and warnings mode, you can pass libretto.pl the option "-vw".
```

CONFIGURATION

Libretto uses Extensible Markup Language (XML) for configuration, and gets its settings from a file named **settings.xml**, located in the root directory of your **Libretto** installation. It contains basic IRC settings and options (you can find an example **settings.xml** with all default options on page 80).

Here is an explanation of all of the settings contained in settings.xml; XML elements are in **bold**.

settings		The root element of a **Libretto** configuration file.
	nickname	This is the IRC nickname the bot will use. If this nickname is already in use by another user, **Libretto** will attach a short number to the end of the nickname and attempt to use that instead.
	ircname	The value set in this setting will be used by the IRC server as your "real name". In reality, most users put a description of themselves or something funny.
	username	The username the bot will use. This is the name that appears the bot's hostmask[1].
	flood-protection	Set this to 1 (one) to turn on flood protection; by default, flood protection is turned on. Set this to 0 (zero)to turn flood protection off. Turning off flood protection is not recommended, as it may get your bot banned from an IRC server or network.
	ipv6	Set this to 1 (one) to turn on IPv6 support; set this to 0 (zero) to turn off IPv6 support. By default, IPv6 support is turned off.
	verbose	Set this to 1 (one) to turn on verbose mode; set this to 0 (zero) to turn verbose mode off. By default, verbose mode is turned off. While in verbose mode, **Libretto** will print runtime information to the console about what it's doing.
	warnings	Set this to 1 (one) to turn on warnings mode; set this to 0 (zero) to turn warnings mode off. By default, warnings mode is turned off. While in warnings mode, **Libretto** will print information about non-fatal errors or other possible malfunctions to the console.

1 https://en.wikipedia.org/wiki/Internet_Relay_Chat#Hostmasks

dcc	DCC[2] settings.

enable	Set this to 1 (one) to turn on DCC support; set this to 0 (zero) to turn off DCC support. By default, DCC support is turned on.
ports	This sets a list of port numbers **Libretto** will use for DCC connections. This can be either a comma-delimited list of port numbers (1000,1001,1002,…) or a range of numbers (1000-2000). Both methods can be mixed; for example, you could set ports 1000 to 2000, 30000, and 22000 to 25000 with "1000-2000,30000,22000-25000".
external-ip	Here, you can set **Libretto** to fetch your external IP address if you are behind a firewall or NAT translator from a website. This is not required.

get	Set this to 1 (one) to turn on fetch your bot's external IP address from the Internet; set this to 0 (zero) to disable this functionality. By default, IP retrieval is turned off.
host	Set this to the web host you wish to retrieve you external IP from. By default, **Libretto** uses "http://myexternalip.com/raw" as the host to use.
set	This setting contains the IP address reported to other IRC and DCC users if Internet IP retrieval is not used. By default, this is not set to anything. IRC will report your IP address as the address it can "see" to other users.

colors	Terminal color settings.

enable	Set this to 1 (one) to turn on ANSI color support; set this to 0 (zero) to turn off ANSI color support. By default, this is turned on. A list of colors available for use an be found in the documentation for the `termcolor()` (page 64) function.
verbose	Sets the color to use for verbose messages. Set to `bold bright_green` by default.
warnings	Sets the color to use for warning messages. Set to bold bright_magenta by default.
errors	Sets the color to use for error messages. Set to `bold bright_red` by default.

2 https://en.wikipedia.org/wiki/Direct_Client-to-Client

If this file gets deleted or otherwise lost, a new **settings.xml** can be generated with the command line option --**generate-config** (see *Usage*, page 3).

Frequently Asked Questions

What is Libretto?

Libretto is an event-driven, open source programming environment for developing Internet Relay Chat[3] software. Programs are written in a variant of ECMAScript (also known as JavaScript), and can utilize both client and server functionality. **Libretto** uses Google's V8[4] ECMAScript engine, the open source JavaScript engine used in the Chrome web browser and the Node.js development platform.

In short: **Libretto** is a tool for writing custom IRC bots and IRC servers.

What is Internet Relay Chat?

Internet Relay Chat, or IRC, is a text chat system used by thousands of people on thousands of servers world wide. Created in 1988 by Jarkko Oikarinen, IRC uses minimal bandwidth and has clients for nearly every operating system in existence. Commonly used clients include mIRC[5] and HexChat (for Windows), and ircII[6], XChat[7], and Konversation[8] for Linux. Common uses for IRC include chatting and file sharing.

What is an IRC Bot?

> *An IRC bot is a set of scripts or an independent program that connects to Internet Relay Chat as a client, and so appears to other IRC users as another user. An IRC bot differs from a regular client in that instead of providing interactive access to IRC for a human user, it performs automated functions.*
>
> – Wikipedia[9]

An IRC bot is useful for many purposes. Bots can be used to provide services for other IRC users (like file sharing), or services for you

3 https://en.wikipedia.org/wiki/Internet_Relay_Chat
4 https://v8.dev/
5 http://www.mirc.com/
6 http://www.eterna.com.au/ircii/
7 http://xchat.org/
8 http://konversation.kde.org/
9 https://en.wikipedia.org/wiki/IRC_bot

(like chat logging). Bots can be as simple or complex as you want them to be.

What are some of Libretto's features?

Libretto features over 75 custom functions built into its JavaScript engine, allowing you to write scripts that:

- **Create rich, complex IRC bots.** If an IRC client can do it, and the IRC protocol supports it, your bot can do it. **Libretto** features full DCC support (including chat, file sending, and file receiving), SSL support (for securely connecting to IRC), and even a minimal web server, for when you have information or data you need programs that don't support IRC to access.
- **Create basic IRC servers. Libretto** IRC servers aren't the complex, full-featured IRCds the big networks use, but it'll work for your office, school, or home network. Server features include IRCop support, AUTH classes, and even spoofed clients (which can be used to program custom IRC server-side service[10] bots).
- **A suite of file input and output functions.** Create and edit files and directories. **Libretto** features cross-platform file I/O functions; write your **Libretto** script once, and run it on any operating system that can run **Libretto**.
- **Built-in zip archive support.** Create, edit, and extract zip archives.

Is Libretto hard to learn?

Not at all! If you've ever written a program or script in JavaScript, you can create an IRC program with **Libretto**. And if you've never written a program in JavaScript, it's still pretty easy. There are hundreds (if not thousands) of tutorials and guides to make learning JavaScript easy. These websites are a great place to start:

JavaScript at W3Schools	https://www.w3schools.com/js/
Modern JavaScript	https://javascript.info/
Learn JavaScript	https://www.learn-js.org/

10 https://en.wikipedia.org/wiki/IRC_services

What version of JavaScript does Libretto use?

ECMAScript 6[11], the same version used by Chrome and Node.js.

What are the requirements to run Libretto?

- Perl
- Perl Object Environment (POE)[12]
  - POE::Component::IRC[13]
  - POE::Component::Server::IRC[14]
- libv8
  - JavaScript::V8[15]

Libretto comes with several Perl modules pre-installed:

- Term::ANSIColor[16]
- Text::ParseWords[17]

If you want to use SSL connections in your script, there's a few more requirements.

- POE::Component::SSLify[18]
- POE::Filter:SSL[19]

If you want to use the built-in web server and webhook functionality, the following Perl module is also required:

- HTTP::Response[20]

What programming language is Libretto written in?

The main program (**libretto.pl**) is written in Perl 5[21].

---

11 http://es6-features.org/
12 http://poe.perl.org/
13 https://metacpan.org/pod/POE::Component::IRC
14 https://metacpan.org/pod/POE::Component::Server::IRC
15 https://metacpan.org/pod/JavaScript::V8
16 https://metacpan.org/pod/Term::ANSIColor
17 https://metacpan.org/pod/Text::ParseWords
18 https://metacpan.org/pod/POE::Component::SSLify
19 https://metacpan.org/pod/POE::Filter::SSL
20 https://metacpan.org/pod/HTTP::Response
21 https://www.perl.org/

How easy is it to create a bot?

Very easy! Here's a simple "greeter" bot that says hello to every user that joins a channel the bot is present in (which, in this example, is a channel named "#foo"):

```
1 client({ server:"ircserver.net", port:"6667" });
2 join("ircserver.net:6667","#foo");
3 function greet(event){
4     say(event.ServerID,event.Channel,"Hello"+event.Nickname+"!");
5 }
6 hook("join","greeter",greet);
```

This is a complete IRC bot, written in 6 lines of code! Writing a custom IRC server is even easier:

```
1 server({ name:"My IRC Server", port:6667 });
```

Can I use Node.js[22] or CommonJS[23] modules in my Libretto bot?

Not at this time.

What operating systems does Libretto run on?

Libretto should run on any platform that Perl runs on and that Google's V8 library can be compiled for. **Libretto** is developed on Debian Linux, and tested on Debian, Ubuntu Linux, and Windows 10. Though it hasn't been tested on OSX, it should have no problem running without modification.

22 https://nodejs.org/
23 https://requirejs.org/docs/commonjs.html

Libretto Language

At its core, **Libretto** is ECMAScript, also known as JavaScript. It uses Google's V8 ES6 engine, the same engine used by the Chrome web browser. What makes **Libretto** different is its command set: over 75 new, custom functions to make creating IRC bots and servers easy, with cross-platform support for file input and output and built-in zip archive support.

Four **Libretto** functions are the basis for making all this work:

- `client()` - Creates IRC client connections. This is where IRC bots start.
- `server()` - Creates IRC servers.
- `hook()` - Connects JavaScript functions to IRC client and server events. This is how **Libretto** scripts interact with IRC.
- `unhook()` - Disconnects IRC events from JavaScript functions, allowing **Libretto** scripts to alter how they run while they run.

Libretto is an event-driven[24] system. When a script is connected to an IRC server (as a `client()`) or hosting IRC clients (as a `server()`), whenever something important happens, **Libretto** triggers an event. Scripts can `hook()` functions to these events, and do different things, depending on the information provided by that event. Event information is passed to the function in the form of an object; each event provides different information in that object, as each event is different.

As an example, let's write a simple IRC bot. Our bot will connect to an IRC server as a client, join a channel name "#foo", and greet everyone who connects to that channel with a custom greeting.

First, our bot has to connect to IRC. For that, we'll use the `client()` function:

```
1 const irc_server = { server:"localhost", port:"6667" };
2
3 client(irc_server);
```

24 https://en.wikipedia.org/wiki/Event-driven_programming

On line 1, we define an object that contains the settings we need to pass to `client()` to create a connection to the IRC server. In this example, we're connecting to an IRC server that is running on the same computer as **Libretto**, on port 6667. On line 3, we connect to the IRC server with the `client()` function.

At this point, our script doesn't appear to do anything. That's because our script is just connecting to IRC and sitting there, doing nothing. Let's `hook()` some events and do something!

When our bot first connects to IRC, it should join the channel we're using in this example, "#foo". The event that **Libretto** triggers when it connects to IRC is named, conveniently, "connect" (this event is documented on page 19).

```
1 function connected(event){
2      join(event.ServerID, "#foo");
3 }
4
5 hook("connect","greeter bot",connected);
```

Lines 1-3 are the function we're `hook()`ing to the "connect" event. When **Libretto** connects to IRC and triggers this event, our `connected()` function is executed. Line 2 calls the `join()` function to tell the bot to join "#foo" on the IRC server, using the `server ID` we previously set up.

The first argument to `join()`, `event.ServerID`, is a bit of information provided by the event to tell us which IRC client connection the event occurred on. A `server ID` is a string that nearly all IRC client functions take as a first argument (see the documentation for the `client()` function on page 41 for more information) to determine which IRC client connection to perform the function on. A `server ID` consists of the hostname used to connect to that server, a colon, and the port we're connected to on that server. Since we used the hostname "localhost" to connect to the server, and we connected to port 6667, our `server ID` for this connection is "localhost:6667".

Line 5 uses the `hook()` function to attach our function to the "connect" event. The first argument is the name of the event we want to hook, which is "connect" in this example. The second argument is called the "event tag", which is a string used to describe what our `hook()` does or what we want to call it or whatever. Event tags do *not*

have to be unique; multiple hook()s can have the same event tag. The event tag is used when we remove an event hook() with the unhook() function; the unhook() function takes an event tag as its only argument. This allows scripts to remove groups of hook()s with a single function call. The last argument to hook() is a function reference, the function we want executed when the event occurs: the connected() function.

Now our bot has connected to IRC and joined "#foo". Here, we're going to hook() another event: "join". The "join" event (documented on page 26) is triggered whenever a user joins a channel the bot is in.

```
1 function joined(event){
2      say(event.ServerID,"#foo","Welcome to #foo, "+ event.Nickname + "!");
3 }
4
5 hook("join","greeter bot",joined);
```

The "join" event provides several bits of information about the user that joined the channel, but we're only going to use the server ID and the nickname of the person who joined (event.ServerID and event.Nickname, respectively). We'll use this information, along with the name of the channel we're in, to send a public chat message to the channel with the say() function (documented on page 47): a customized greeting to the user welcoming them to the channel.

Built-In Variables

Variable Name	Type	Read Only?	Description
ARGV	**Array**	**Yes**	Contains any command line arguments passed to the script. So, if the script was executed with `perl libretto.pl script.js localhost port`, then ARGV would be an array with two values, "localhost" and "port", in that order.
DCC	**Boolean**	**Yes**	If DCC is enabled, this variable is set to "true"; if DCC is not enabled, this variable is set to "false".
HOST	**String**	**Yes**	The host operating system.
HTTPD	**Boolean**	**Yes**	If `webhook()`s are available, this variable is set to "true"; if `webhook()`s are not available, this variable is set to "false".
SCRIPTNAME	**String**	**Yes**	The base file name of the script being executed (that is, without the path; so, if the script being ran was "/home/bob/script.js", SCRIPTNAME would be "script.js").
UPTIME	**String**	**No**	How long since **Libretto** was executed, in seconds. Even though this variable is not declared as a constant, trying to store information in it will lead to frustration; its value is set once per second to the current uptime.
VERBOSE	**Boolean**	**Yes**	If verbose mode is turned on, this is set to "true"; if not, this variable is set to "false".
WARNINGS	**Boolean**	**Yes**	If warnings mode is turned on, this is set to "true"; if not, this variable is set to "false".
WHITE BLACK BLUE GREEN RED BROWN PURPLE ORANGE YELLOW LIGHT_GREEN TEAL CYAN LIGHT_BLUE PINK GREY LIGHT_GREY	**String**	**Yes**	Color variables for use with the `color()` function. These represent mIRC color codes, which are supported by most modern IRC clients.

Event Functions

delay

```
Function delayed_function(){
     print("It's ten seconds later!");
}

delay(10,delayed_function);
```

Arguments	2 (time to delay in seconds, function reference)
Returns	Nothing
Description	Sets a function to be executed after a delay.

hook

```
Function on_join(){
     print("Somebody joined the channel!");
}

hook("join","join event",on_join);
```

Arguments	3 (event name, event tag, function reference)
Returns	Nothing
Description	"Hooks" a function to an event, causing the function to be executed every time the event occurs. "Event name" is the name of the event to hook (see *Client Events* on page 18, and *Server Events* on page 33), "event tag" is a non-unique string tag to be attached to the event function (this is for removing hooks with the `unhook()` function), and "function reference" is the name of the function to execute when the event occurs.

unhook

```
unhook("join event");
```

Arguments	1 (event tag)
Returns	Nothing
Description	Removes one or more event hooks. Since the "event tag" placed on hooks with the `hook()` function do not have to be unique, this allows for multiple hooks to be removed with a single function call.

webhook

```
function homepage(client){
     var response = {
          Code: "200", Type: "text/html",
          Content: "<h1>You requested the page for "+client.Request+"</h1>"
     };
     return response;
}

if(webhook("my_web_server",15000,homepage)){
     print("Web server started on port 15000");
}

// Shutdown the web server
shutdown("my_web_server");
```

Arguments	3 (webhook ID, port number, function reference)
Returns	1 if successful, 0 if not.
Description	To use this function, you might have to install a prerequisite (see *Frequently Asked Questions* on page 9 for more information). Without the prerequisite, `webhook()`s will be unavailable. Starts a web server on the desired port, using the response of a function to generate the content served. The first argument, the `webhook ID`, can be a string named anything you wish, as long as it is unique to your script. The second argument is the port number you want the web server to use; unless you are running **Libretto** as root (not recommended), this should use a port greater than 1000 and not already used by another server. The third argument is a reference to a function

that will provide the web server's content.

When a client connects to the web server, this function will be passed an object as its sole argument. The object will always have three properties: "Request", "IP", and "Port". "Request" will contain the web directory, page, or file that the client has requested from the web server; "IP" and "Port" will contain the requesting user's IP address and port. If the request has a query string[25] attached, the object will be populated with values from the query string (assuming a question mark [?] as the separator and ampersands [&] as delimiters).

The function can return content in three different ways:

- **As an object.** The object should have three properties: "Code" (an appropriate HTTP response code[26]; "200" is the code for "request successful"), "Type" (the media type[27] of the response; for example, "text/plain" for plain text or "text/html" for HTML), and "Content" (the served content). If the object returned does not have *all three of these properties*, the objects contents will be rendered as a plain text list of properties and values, and sent to the client as plain text.
- **As a string.** The string will be sent as it to the client, as plain text.
- **As an array.** The array's values will be concatenated with newlines and sent to the client as plain text.

If the function returns any other type, **Libretto** will display an error and exit.

25 https://en.wikipedia.org/wiki/Query_string
26 https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
27 https://en.wikipedia.org/wiki/Media_type

Client Events

action	kick
away	mode
back	mode-channel
connect	mode-user
dcc-chat	nick-changed
dcc-chat-request	nick-taken
dcc-done	notice
dcc-error	part
dcc-get	private
dcc-send	public
dcc-send-request	quit
dcc-start	raw
invite	raw-out
join	topic

action

```
function on_action_event(event){
     print(event.Nickname + " " + event.Action);
}

hook("action","action event",on_action_event);
```

Argument Object Properties

Action	The text of the action message.
Channel	The channel where the action message was sent.
Nickname	The nick of the user sending the action.
Hostmask	The hostmask of the user sending the action.
ServerID	The server ID of the server who sent the message.

Description

Triggers whenever a user in the bot's presence send a CTCP action message.

away

```
function on_away_event(event){
     print(event.Nickname + " went away!");
}

hook("away","away event",on_away_event);
```

Argument Object Properties		
	Nickname	The nick of the user who went away,
	ServerID	The server ID of the server who sent the message.
Description	Triggers whenever a user in the bot's presence sets themselves "away".	

back

```
function on_back_event(event){
     print(event.Nickname + " is back!");
}

hook("back","back event",on_back_event);
```

Argument Object Properties		
	Nickname	The nick of the user who came back.
	ServerID	The server ID of the server who sent the message.
Description	Triggers whenever a user in the bot's presence sets themselves "back".	

connect

```
function on_connect_event(event){
     print("Connected to "+event.Server+"!");
}

hook("connect","connect event",on_connect_event);
```

Argument Object Properties		
	ServerID	The server ID of the server who sent the message.
Description	Triggers whenever a successful IRC server connection is established.	

dcc-chat

```
function on_dcc_chat_event(event){
     print("DCC "+event.Nickname+": "+event.Message);
}

hook("dcc-chat","chat event",on_dcc_chat_event);
```

Argument Object Properties

Nickname	The nick of the user who sent the chat.
IP	The IP address of the user who sent the chat.
Port	The port the chat is using.
Cookie	The DCC chat ID, used to send messages to this user via DCC.
Message	The chat message.

Description

Triggers whenever a user sends a DCC chat message to the bot.

dcc-chat-request

```
function on_dcc_chat_request_event(event){
     print(event.Nickname+" wants to DCC chat!");
     return true;
}

hook("dcc-chat-request","chat request event",on_dcc_chat_request_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user who sent the chat request.
Hostmask	The hostmask of the user who sent the chat request.
IP	The IP address of the user who sent the chat request.
Port	The port the user requesting the chat will use.
Cookie	The DCC request ID, used to send messages to this user via DCC.

Description

Triggers whenever a user requests a DCC chat session from the bot. If you wish to accept the request and begin a DCC chat session, the function hooked to this event should return a true value; if you wish to reject the DCC request, return a false value.

dcc-done

```
function on_dcc_done_event(event){
     print("DCC "+event.Type.+" completed");
}

hook("dcc-done","dcc done event",on_dcc_done_event);
```

Argument Object Properties

Nickname	The nick of the user who's session has completed.
IP	The IP address of the user who's session is complete.
Port	The port the user is using.
Cookie	The DCC user ID, used to send messages to this user via DCC.
Type	The type of DCC session completed.

Description

Triggers whenever a DCC session is complete.

dcc-error

```
function on_dcc_error_event(event){
     print("DCC "+event.Type.+" error! "+event.Error);
}

hook("dcc-error","dcc error event",on_dcc_error_event);
```

Argument Object Properties

Nickname	The nick of the user who's session had an error.
IP	The IP address of the user who's session had an error.
Port	The port the session is using.
Cookie	The DCC user ID, used to send messages to this user via DCC.
Type	The type of DCC session that had an error.
Error	The error that occurred

Description

Triggers whenever a DCC session is disconnected due to error.

dcc-get

```
function on_dcc_get_event(event){
     print("Receiving "+event.File+" "+event.Transferred+" out of "+event.Size);
}

hook("dcc-get","dcc get event",on_dcc_get_event);
```

Argument Object Properties

Nickname	The nick of the user who's sending the file.
IP	The IP address of the user who's sending the file.
Port	The port the session is using.
Cookie	The DCC user ID, used to send messages to this user via DCC.
File	The name of the file being uploaded to the bot.
Size	The size of the file in bytes.
Transferred	The number of bytes transferred to the bot.

Description

Triggers whenever a chunk of a file is sent to the bot via DCC.

dcc-send

```
function on_dcc_send_event(event){
     print("Sending "+event.File+" "+event.Transferred+" out of "+event.Size);
}

hook("dcc-send","dcc send event",on_dcc_send_event);
```

Argument Object Properties

Nickname	The nick of the user who's downloading the file.
IP	The IP address of the user who's downloading the file.
Port	The port the session is using.
Cookie	The DCC user ID, used to send messages to this user via DCC.
File	The name of the file being downloaded from the bot.
Size	The size of the file in bytes.
Transferred	The number of bytes transferred to the user.

Description

Triggers whenever a chunk of a file is sent to a user via DCC.

dcc-send-request

```
function on_dcc_send_request_event(event){
     print(event.Nickname+" wants to send "+event.File+ "("+event.Size+")");
     return true;
}

hook("dcc-send-request","send request event",on_dcc_send_request_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user who sent the send request.
Hostmask	The hostmask of the user who sent the send request.
IP	The IP address of the user who sent the send request.
Port	The port the user requesting to send the file will use.
File	The file name the user wants to send.
Size	The size of the file the user wants to send in bytes.
Cookie	The DCC request ID, used to send messages to this user via DCC.

Description

Triggers whenever a user requests to send a file via DCC to the bot. If you wish to accept the request, the function hooked to this event should return a true value; if you wish to reject the request, return a false value. The file will be placed in the directory set in the settings file.

dcc-start

```
function on_dcc_start_event(event){
     print("DCC "+event.Type+" session started with "+event.Nickname);
}

hook("dcc-start","dcc start event",on_dcc_start_event);
```

Argument Object Properties

Nickname	The nick of the user who's starting the DCC session.
IP	The IP address of the user who's starting the session.
Port	The port the session is using.
Cookie	The DCC user ID, used to send messages to this user via DCC.
Type	The type of DCC session being started ('CHAT', 'GET', or 'SEND').

Description

Triggers whenever a DCC session starts.

invite

```
function on_invite_event(event){
     print(event.Nickname+" invited me to "+event.Channel);
}

hook("invite","invite event",on_invite_event);
```

Argument Object Properties

Nickname	The nick of the user sent the invite.
Hostmask	The hostmask of the user sending the invitation.
Channel	The channel the user is inviting the bot to.
ServerID	The server ID of the server who sent the message.

Description

Triggers whenever the bot receives a channel invitation

join

```
function on_join_event(event){
     print(event.Nickname+" joined "+event.Channel);
}

hook("join","join event",on_join_event);
```

Argument Object Properties

Nickname	The nick of the user who joined.
Hostmask	The hostmask of the user who joined.
Channel	The channel the user joined.
ServerID	The server ID of the server who sent the message.

Description

Triggers whenever the bot receives a channel invitation

kick

```
function on_kick_event(event){
     print(event.Nickname+" kicked "+event.Target+" from "+event.Channel);
}

hook("kick","kick event",on_kick_event);
```

Argument Object Properties

Nickname	The nick of the user who issued the kick command.
Hostmask	The hostmask of the user issued the kick command.
Channel	The channel the target was kicked from.
ServerID	The server ID of the server who sent the message.
Target	The nick of the user that was kicked.
TargetHostmask	The hostmask of the user that was kicked.
Reason	The reason the user was kicked, if any.

Description

Triggers whenever a user is kicked from a channel in the bot's presence.

mode

```
function on_mode_event(event){
     print(event.Nickname+" set mode +event.Mode+" "+event.Arguments);
}

hook("mode"," mode event",on_mode_event);
```

Argument Object Properties

Nickname	The nick of the user who set the mode.
Hostmask	The hostmask of the user set the mode.
ServerID	The server ID of the server who sent the message.
Target	The target of the mode set (a user or a channel).
Mode	The mode set.
Arguments	Any arguments used with the mode set.

Description

Triggers whenever a mode is set in the presence of the bot.

mode-channel

```
function on_mode_channel_event(event){
     print(event.Nickname+" set mode +event.Mode+" "+event.Arguments);
}

hook("mode-channel","mode-channel event",on_mode_channel_event);
```

Argument Object Properties

Nickname	The nick of the user who set the mode.
Hostmask	The hostmask of the user set the mode.
ServerID	The server ID of the server who sent the message.
Channel	The target channel of the mode set.
Mode	The mode set.
Arguments	Any arguments used with the mode set.

Description

Triggers whenever a mode is set on a channel in the presence of the bot.

mode-user

```
function on_mode_user_event(event){
     print(event.Nickname+" set mode +event.Mode+" "+event.Arguments);
}

hook("mode-user","mode-user event",on_mode_user_event);
```

Argument Object Properties

Nickname	The nick of the user who set the mode.
Hostmask	The hostmask of the user set the mode.
ServerID	The server ID of the server who sent the message.
Channel	The target user of the mode set.
Mode	The mode set.
Arguments	Any arguments used with the mode set.

Description

Triggers whenever a mode is set on a user in the presence of the bot.

nick-changed

```
function on_nick_changed_event(event){
     print(event.Nickname+" is now known as "+event.New);
}

hook("nick-changed","nick-changed event",on_nick_changed_event);
```

Argument Object Properties

Nickname	The nick of the user who changed their nickname.
Hostmask	The hostmask of the user who changed their nickname.
ServerID	The server ID of the server who sent the message.
New	The new nickname.

Description

Triggers whenever a user changes their nickname in the bot's presence.

nick-taken

```
function on_nick_taken_event(event){
     print("Our nick is taken! Changing it to 'librettobot'");
     nick(event.Server,"librettobot");
}

hook("nick-taken","nick-taken event",on_nick_taken_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.

Description

Triggers whenever the server notifies the bot that the nick the bot is trying to use is already taken.

notice

```
function on_notice_event(event){
     print("NOTICE: "+event.Nickname+": "+event.Message);
}

hook("notice"," notice event",on_notice_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user that sent the notice.
Hostmask	The hostmask of the user that sent the notice.
Targets	An array of the users and channels the notice was sent to.
Message	The contents of the notice.

Description

Triggers whenever a notice is sent to the bot.

part

```
function on_part_event(event){
     print(event.Nickname+" left "+event.Channel);
}

hook("part"," part event",on_part_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user that left the channel.
Hostmask	The hostmask of the user that left the channel.
Channel	The channel the user left.
Message	The user's parting message, if they sent on.

Description

Triggers whenever a user leaves a channel in the bot's presence.

private

```
function on_private_event(event){
     print("PRIVATE "+event.Nickname+": "+event.Message);
}

hook("private"," private event",on_private_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user that sent the private message.
Hostmask	The hostmask of the user that send the private message.
Message	The private message's contents.

Description

Triggers whenever a user sends a private message to the bot.

public

```
function on_public_event(event){
     print(event.Channel+" "+event.Nickname+": "+event.Message);
}

hook("public"," public event",on_public_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user that sent the message.
Hostmask	The hostmask of the user that send the message.
Channel	The channel the user sent the message to.
Message	The message's contents.

Description

Triggers whenever a user sends a message to a channel the bot is present in.

quit

```
function on_quit_event(event){
     if(event.Message == ""){
          print(event.Nickname+" quit IRC");
     } else {
          print(event.Nickname+" quit IRC ("+event.Message+")");
}

hook("quit"," quit event",on_quit_event);
```

Argument Object Properties

ServerID	The server ID of the server who sent the message.
Nickname	The nick of the user that quit IRC.
Hostmask	The hostmask of the user that quit IRC.
Message	The message sent with the quit command, if any.

Description

Triggers whenever a user quits IRC in the bot's presence.

raw

```
function on_raw_event(event){
     print(event.Line);
}

hook("raw"," raw event",on_raw_event);
```

Argument Object Properties		
	ServerID	The server ID of the server who sent the message.
	Line	The contents of the message the server sent.
Description	Triggers whenever the server sends *any* message to the IRC bot.	

raw-out

```
function on_raw_out_event(event){
     print(event.Line);
}

hook("raw-out"," raw-out event",on_raw_out_event);
```

Argument Object Properties		
	ServerID	The server ID of the server the message was sent to.
	Line	The contents of the message the bot sent.
Description	Triggers whenever the bot sends *any* message to the IRC server.	

client-channel-mode

```
function client_channel_mode(event){
     print(event.Nickname + " set a mode on " + event.Channel + ":" + \
     event.Mode.join(" "));
}

hook("client-channel-mode","ccm event",client_channel_mode);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user setting the channel mode.
Hostmask	The hostmask of the user setting the channel mode.
Channel	The channel the mode was set on.
Mode	An array containing the mode(s) and any arguments.

Description

Triggers whenever a user sets a channel mode on an IRC server.

client-join

```
function client_join(event){
     print(event.Nickname + " joined " + event.Channel);
}

hook("client-join","cj event",client_join);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user joining the channel.
Hostmask	The hostmask of the user joining the channel.
Channel	The channel joined.

Description

Triggers whenever a user sets joins a channel on an IRC server.

client-kick

```
function client_kick(event){
     print(event.Nickname + " kicked " + event.Target + \
     " from " + event.Channel);
}

hook("client-kick","ck event",client_kick);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user issuing the kick command.
Hostmask	The hostmask of the user issuing the kick command.
Channel	The channel the target is kicked from.
Target	The target of the kick command.
Reason	The reason the target was kicked.

Description

Triggers whenever a user is kicked from a channel.

client-mode

```
function client_mode(event){
     print(event.Nickname + " kicked " + event.Target + \
     " from " + event.Channel);
}

hook("client-mode","cm event",client_mode);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user setting the user mode.
Hostmask	The hostmask of the user setting the user mode.
Mode	The mode set and any arguments.

Description

Triggers whenever a user mode is set on an IRC server.

client-part

```
function client_part(event){
     print(event.Nickname + " joined " + event.Channel);
}

hook("client-part","cp event",client_part);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user parting the channel.
Hostmask	The hostmask of the user parting the channel.
Channel	The channel parted.
Message	Optional message.

Description

Triggers whenever a user sets parts a channel on an IRC server.

client-quit

```
function client_quit(event){
     if(event.Message != ""){
          print(event.Nickname + " quit");
     } else {
          print(event.Nickname + " quit:" + event.Message);
     }
}

hook("client-quit","cq event",client_quit);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user quitting.
Hostmask	The hostmask of the user quitting.
Message	Optional message.

Description

Triggers whenever a user quits an IRC server.

client-topic

```
function client_topic(event){
     print(event.Nickname + " set " + event.Channel +"'s topic to " + \
     event.Topic);
}

hook("client-topic","ct event",client_topic);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Nickname	The nickname of the user setting the channel topic.
Hostmask	The hostmask of the user setting the channel topic.
Channel	The channel where the topic was set.
Topic	The new topic.

Description

Triggers whenever a user sets a channel topic on an IRC server.

server-error

```
function server_error(event){
     print(event.Server + " had an error: " + event.Reason);
}

hook("server-error","se event",server_error);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Server	The name of the server that had the error.
Reason	The error that occurred.

Description

Triggers whenever a server on the IRC network has an error.

server-join

```
function server_join(event){
     print(event.Server + "(" + event.Description + ") joined the network");
}

hook("server-join","sj event",server_join);
```

Argument Object Properties

HostID	The host ID of the server triggering the event.
Server	The name of the joining server.
Introducer	The server that "introduced" the new server to the network.
Hops	How many "hops" away the new server is.
Description	A description of the new server.

Description

Triggers whenever a server joins the network of an IRC server.

server-quit

```
function server_quit(event){
     print(event.Nickname + " set " + event.Channel +"'s topic to " + \
     event.Topic);
}

hook("server-quit","sq event",server_quit);
```

Argument Object Properties	**HostID**	The host ID of the server triggering the event.
	Server	The name of the server that quit.
Description	Triggers whenever a server quits the network of an IRC server.	

Client Functions

Most functions that issue commands on an IRC server use a "server ID" to determine which server connection to send the command to. Please see the documentation for the `client()` function for a complete description.

bold

```
// Returns a bold "Hello, world!"
var bold_helloworld = bold("Hello, world!");
```

Arguments	1 (string)
Returns	String
Description	Applies the bold mIRC control code to text, and returns it.

channel

```
// Get a list of users in #foo
var channel_users = channel(server,"#foo","users");

// Get a channel's key (if known or in the channel)
var channel_key = channel(server,"#foo","key");

// Get a channel's topic
var channel_topic = channel(server,"#foo","topic");

// Get a list of channel operators
var channel_ops = channel(server,"#foo","ops");

// Get a list of all the channels the bot is in
var channel_list = channel(server,"list");
```

Arguments 2-3 (see description)

Returns Varies (see description)

Description Gets channel information for a specific connection. What information is retrieved is dependent on what arguments you pass the command:

Arguments	Returns
2 Arguments server ID, "list"	An array containing a list of all the channels the bot is present in.
3 Arguments server ID, channel name, "users"	An array containing a list of all the users in a channel.
3 Arguments server ID, channel name, "topic"	A string containing a channel's topic.
3 Arguments server ID, channel name, "key"	A string containing a channel's key (if known).
3 Arguments server ID, channel name, "ops"	An array containing a list of all the operators in a channel.
3 Arguments server ID, channel name, "admins"	An array containing a list of all the admins in a channel.
3 Arguments server ID, channel name, "halfops"	An array containing all the half-op users in a channel.
3 Arguments server ID, channel name, "voiced"	An array containing all the voiced users in a channel.

If the information requested is not available, the function will return `undefined`.

client

```
// Use an object to connect to a server
client({server:'server.net',port:6667});
```

Arguments 1 (object)

Returns Nothing

Description Connects to an IRC server as a client. Settings for this IRC connection is set via an object passed as an argument (which is explained here). The object should look like this:

```
var server_object = {
      server:"hostname",
      port:6667,
      password:undefined,
     nickname:"libretto_bot",
     username:"librettobot",
     ircname:"Libretto IRC Bot",
      ssl:{
            enable:0,
            certificate:undefined,
            key:undefined
      },
      proxy:{
            enable:0,
            server:undefined,
            port:undefined
      },
      socks:{
            enable:0,
            userid:undefined,
            server:undefined,
            port:undefined
      }
};
```

The "server" and "port" properties are self-explanatory: they set the IP or host name and port of the IRC server to connect to. The "password" property can be set if the IRC server requires a password for connection. If the "port" property is omitted, a default port of "6667" is used. These two properties are used to send commands to a specific IRC server with Libretto functions, called a **server ID**. A server ID consists of the IP or host name use to connect to the server, followed by a colon (:),

followed by the port used to connect to that server. So, if the host name "irc.servercentral.net" and the port "6667" was used to connect, Libretto functions would use the server ID "irc.servercentral.net:6667" to issue commands on that server. The "server" and "port" properties are the only mandatory properties in the connection object; if they are missing, **Libretto** will display an error and exit.

The "nickname" property sets the nickname the client will use. The "username" and "ircname" properties sets the client's username and IRCname (also called the "real name").

The "ssl" property consists of an object that configures a SSL connection to the IRC server. Set "enable" to "1" (one) to enable SSL; if the connection requires a certificate and key files, set "certificate" and "key" to the appropriate file names. If the "port" property is omitted, a default port of "6697" is used for SSL connections. If you want to use SSL connections in your **Libretto** programs, additional requirements must be installed (see *Frequently Asked Questions* on page 9).

The "proxy" property consists of an object that configures a proxy connection to route the IRC connection through. Set "enable" to "1" (one) to enable the proxy connection; set "server" to the proxy's host name or IP address, and "port" to the proxy's port. Omitting either of these will cause an error, and the script will exit.

The "socks" property consists of an object that configures a SOCKS/SOCKS4a connection to route the IRC connection through. Set "enable" to "1" (one) to enable the SOCKS connection; set "server" to the SOCKS server's IP address or host name, and "port" to the SOCKS server's port. If "port" is omitted, a default port of "1080" is used. If the SOCKS server you are connecting to uses a user ID, set "userid" to the user ID to use; this property is optional.

Only necessary properties are required; if the connection does not use SSL, a proxy server, or a SOCKS

server, these properties can be omitted. The only necessary property is the "server" property, which sets the IP address or host name of the IRC server to be connected to; if the "port" property is omitted, a default port of "6667" is used.

A script can connect to a server at any time, and does not have to occur at the beginning of a script. There's no limit to the number of IRC servers a script can connect to, and each connection can be configured independently of the others; a script can connect to one server via SSL, another server via a proxy, and another server via a direct connection.

IMPORTANT NOTE: `client()` *must* be called in the global scope only at this time; this means that it cannot be called from another function. If `client()` is called from a scope other than the global scope, **Libretto** will display an error and exit.

color

```
// Returns a colored "Hello, world!"
var yellow_and_green_helloworld = color(YELLOW,GREEN,"Hello, world!");
```

Arguments	3 (foreground color, background color, string)
Returns	String
Description	Applies the desired mIRC color control code to text, and returns it.

dcc

```
dcc(serverID,cookie,"Hello!");
```

Arguments	3 (server ID, cookie, string)
Returns	Nothing
Description	Sends a DCC chat message. The "cookie" is a short string that identifies who the message is supposed to be sent to; it can be acquired from the miscellaneous DCC client events (pages 20-25).

dcc_close

```
dcc_close(serverID,cookie);
```

Arguments	2 (server ID, cookie)
Returns	Nothing
Description	Terminates a DCC connection. The "cookie" is a short string that identifies which DCC connection to terminate; it can be acquired from the miscellaneous DCC client events (pages 20-25).

disconnect

```
// Disconnects from a server
disconnect(server);

// Disconnects with a message
disconnect(server,"See you later!");
```

Arguments	1+ (server ID, optional message)
Returns	Nothing
Description	Disconnects from an IRC server.

invite

```
// Invite user "bob" to channel "#foo"
invite(server,"bob","#foo");
```

Arguments	3 (server ID, nickname, channel name)
Returns	Nothing
Description	Sends an invitation to another user to join a channel.

italic

```
// Returns an italicized "Hello, world!"
var italic_helloworld = bold("Hello, world!");
```

Arguments	1 (string)
Returns	String
Description	Applies the italics mIRC control code to text, and returns it.

join

```
join(server,"#foo");

// Join a channel that requires a password
join(server,"#foo","changeme");
```

Arguments	3+ (server ID, channel, optional password)
Returns	Nothing
Description	Causes the bot to join a channel.

mode

```
// Give user "bob" operator status in #foo
mode(server,"#foo +o bob");
```

Arguments	2 (server ID, mode to set)
Returns	Nothing
Description	Sets a channel or user mode on a user or channel.

nickname

```
// Change our nick to "libretto"
nickname(server,"libretto");
```

Arguments	2 (server ID, new nickname)
Returns	Nothing
Description	Causes the bot to change nicknames on a server. This new nickname will *not* be changed on any other server the bot is connected to automatically.

notice

```
// Send a notice to user "bob"
notice(server,"bob","I'm sending you this notice!");
```

Arguments	3 (server ID, nickname, message)
Returns	Nothing
Description	Sends a notice to a user or channel.

part

```
part(server,"#foo");

// Leave a channel with a parting message
part(server,"#foo","See you later!");
```

Arguments	3+ (server ID, channel, optional message)
Returns	Nothing
Description	Causes the bot to leave a channel.

raw

```
// Change the bot's nick the "hard" way
raw("NICK newname");
```

Arguments	2 (server ID, IRC command)
Returns	Nothing
Description	Sends a raw command to the IRC server. This allows for bots to send commands supported by the IRC server but not necessarily by Libretto.

reconnect

```
// Reconnect to a server after disconnecting
reconnect(server);
```

Arguments	1 (server ID)
Returns	Nothing
Description	Reconnects to an IRC server after disconnection.

say

```
// Says hello to the people in a channel
say(server,"#foo","Hi, everybody!");

// Sends a private message
say(server,"alice","What's up, alice?");
```

Arguments	4 (server ID, channel or user, message)
Returns	Nothing
Description	Sends a message to the IRC server.

send

```
// Send a document to user "bob"
send(server,"bob","poetry.docx");
```

Arguments	3 (server ID, nickname, file name)
Returns	Nothing
Description	Sends a file to a user via DCC.

topic

```
// Sets a new channel topic
topic(server,"#foo","Here's my new topic!");
```

Arguments	3 (server ID, channel, string)
Returns	Nothing
Description	Set's an IRC channel's topic. This will only be set if the user has the correct channel permissions.

underline

```
// Returns an underlined "Hello, world!"
var underlined_helloworld = bold("Hello, world!");
```

Arguments	1 (string)
Returns	String
Description	Applies the underline mIRC control code to text, and returns it.

whois

```
var userinfo = whois(server,"bob");
if(userinfo){
     print("User "+userinfo.Nick+" is "+userinfo.Hops+" hops away");
}
```

Arguments 2 (server ID, nickname)

Returns Object or `undefined`

Description Looks for information on a user, and if found, returns it; if the information is not found, the function returns `undefined`. The bot must be in the same channel as the user to see this information. The information is returned in object format:

Property	Description
Nick	The user's nickname.
User	The user's username.
Userhost	The user's hostmask.
Host	The host the user is connected to.
Server	The name of the server the user is connected to.
Real	The user's real name, if set.
Away	"1" (`true`) if the user is set away, "0" (`false`) if not.
Hops	How many server hops the user is away from (how many servers in the IRC network the user's messages travel through)
IRCop	Set to "1" if the user is an IRCop (this property will only be set if the user is an IRCop)

Server Functions

addauth

```
// Adds a new auth class
addauth(hostID,mask,password,spoof);
```

Arguments

- 2 (host ID, mask)
- 3 (host ID, mask, password)
- 4 (host ID, mask, password, spoof host)

Returns

Nothing

Description

Adds a new class of authorized users (hosts allowed to connect to an IRC server. The "mask" is an IP mask; this is a string that is compared to a connecting user. "*@*" allows anyone to connect. "*" (without the quotes) can be used to represent a single or multi-character wildcard; "?" can be used to represent a single character wildcard. The password, if set, will require matching connecting users to supply the given password to connect. The "spoof host", if set, will spoof the matching users' hostname; so, if the spoof host is set to "google.com", users' hostmasks will appear as if they are connecting from Google's network.

If no auth classes have been created, a default auth class with a mask of "*@*" is set, allowing anyone to connect.

addop

```
// Adds a new IRCop account
addop(hostID,"username","password","*@google.com");
```

Arguments	• 3 (host ID, username, password) • 4 (host ID, username, password, IP mask)
Returns	Nothing
Description	Adds an IRCop account to an IRC server. If the IRC server has very recently been created (say, in the previous function call), this command may fail; use `delay()` to call this function in a second or two.

allchannels

```
// Gets a list of all the channels on an IRC server
var channels = allchannels(hostID);
```

Arguments	1 (host ID)
Returns	Array
Description	Returns a list of all channels on an IRC server.

allnicks

```
// Gets a list of all the users on an IRC server
var users = allnicks(hostID);
```

Arguments	1 (host ID)
Returns	Array
Description	Returns a list of all users on an IRC server.

deleteauth

```
// Removes an auth class
deleteauth(hostID,mask);
```

Arguments	2 (host ID, mask)
Returns	Nothing
Description	Removes an auth class; the mask provided as an argument should be the mask the auth class was created with.

deleteop

```
// Removes an IRCop account
deleteop(hostID,"username");
```

Arguments	2 (host ID, username)
Returns	Nothing
Description	Removes an IRCop account from an IRC server.

forcekick

```
// Forces a channel kick
forcekick(hostID,"#foo","badguy","nobody likes you");
```

Arguments	• 3 (host ID, channel, nickname) • 4 (host ID, channel, nickname, message)
Returns	Nothing
Description	Forces a user to be kicked from a channel.

forcekill

```
// Forces a user kill
forcekill(hostID,"alice","stop harassing people");
```

Arguments	• 2 (host ID, nickname) • 3 (host ID, nickname, message)
Returns	Nothing
Description	Forces a user kill (client disconnection) on an IRC server.

forcemode

```
// Forces a channel mode
forcemode(hostID,"#foo","+o my_friend");
```

Arguments	3 (host ID, channel, mode with arguments)
Returns	Nothing
Description	Forces a channel mode to be set on an IRC server.

server

```
// Creates an IRC server
server(CONFIG);
```

Arguments 1 (object)

Returns Nothing.

Description Creates an IRC server. Configuration is set via an object passed as the only argument to the server() function. The object should look like this:

```
var irc_server = {
      name:"IRC Server",
      nicklength:20,
      network:"Bot.js.net",
      maxtargets:20,
      maxchannels:20,
      description:"Libretto Bot.js Server",
      port:7000
};
```

The "name" and "description" properties are the server's name and description reported to IRC clients. "nicklength" sets the maximum allowed length of user nicknames, "maxtargets" sets the maximum number of users or channels a message or notice can be sent to, and "maxchannels" sets the maximum number of channels a user can join. "port" sets what port the IRC server will listen for connections on.

Other server functions usually call for a **host ID** as a first argument; the host ID for a given server is the port number the server listens on. So, for example, if you create a server with the server() function that listens on port 7000, that server's host ID would be "7000".

File and Directory Functions

basename

```
// Returns "myfile.txt"
var f = basename("/home/user/myfile.txt");
```

Arguments	1 (filename)
Returns	String
Description	Extracts and returns the filename from a full file path.

catdir

```
// Returns "/home/user/dir" on *NIX
var d = catdir("home","user","dir");
```

Arguments	1+ (directory names as strings or arrays of strings)
Returns	String
Description	Concatenates directory names into a valid path for the platform it's called on.

catfile

```
// Returns "/home/user/dir/program.exe" on *NIX
var f = catfile("home","user","dir","program.exe");
```

Arguments	1+ (filenames or directories as strings or arrays of strings)
Returns	String
Description	Concatenates directories and filenames into a valid path for the platform it's called on.

cd

```
cd("/var/www");
```

Arguments	1 (directory name)
Returns	1 if successful, 0 if not
Description	Moves the current working directory to a new directory.

chmod

```
chmod("777");
```

Arguments	1 (file/directory permissions)
Returns	1 if successful, 0 if not
Description	Changes a file or directory's permissions.

cwd

```
var mydirectory = cwd();
```

Arguments	0
Returns	String
Description	Returns the script's current working directory.

dirlist

```
var files = dirlist("/home/user");

// List only Javascript files
var files = dirlist("/home/user","*.js");
```

Arguments	• 1 (directory name) • 2 (directory name, filter)
Returns	Array
Description	Returns a list of files in a directory. If using a filter, * is a wildcard.

flocation

```
var where = flocation("libretto.pl");
```

Arguments	1 (filename)
Returns	String
Description	Returns the directory a file is in.

fmode

```
var libretto_mode = flocation("libretto.pl");
```

Arguments	1 (directory or filename)
Returns	String
Description	Returns a file's or directory's mode (permissions).

fpermissions

```
var libretto_permissions = fpermissions("libretto .pl");
if(libretto_permissions.indexOf("r") !== -1){ print("File isn't readable!"); }
```

Arguments	1 (directory or filename)
Returns	String
Description	Returns a short string describing a file or directory's permissions. If a file is readable, the string will contain "r"; if the file is writable, the string will contain "w"; if the file is executable, the string will contain "x".

fread

```
var contents = fread("file.txt");
```

Arguments	1 (filename)
Returns	String
Description	Reads the contents of a file and returns them.

fsize

```
var libretto_size = fsize("libretto.pl");
```

Arguments	1 (filename)
Returns	String
Description	Returns a file's size in bytes.

fwrite

```
fwrite("file.txt","This is the contents of my file!");
```

Arguments	2 (filename, contents)
Returns	1 if successful, 0 if not
Description	Writes to a file.

isdir

```
if(isdir("/home/user")){
     print("/home/user is a directory!");
}
```

Arguments	1 (string)
Returns	1 if the passed string is a valid directory name, 0 if not
Description	Determines if a string is an existing directory name.

isfile

```
if(isfile("/home/user/file.txt")){
     print("/home/user/file.txt is a file!");
}
```

Arguments	1 (string)
Returns	1 if the passed string is a valid filename, 0 if not
Description	Determines if a string is an existing filename.

mkdir

```
mkdir("mydir");
```

Arguments	1 (directory name)
Returns	1 if successful, 0 if not
Description	Creates a directory.

mkpath

```
mkpath("/home/user/x/y/mydir");
```

Arguments	1 (directory path)
Returns	1 if successful, 0 if not
Description	Creates a directory path; this may involve the creation of multiple directories. For example, if `mkpath("/home/user/x/y/mydir")` is issued, this will create three directories: "/home/user/x", "/home/user/x/y", and "/home/user/x/y/mydir".

rmdir

```
// Deletes a directory named "unwanted" in the current directory
rmdir("unwanted");
```

Arguments	1 (directory)
Returns	1 if successful, 0 if not
Description	Deletes a directory. If the directory has any files in it, the deletion will fail.

rmfile

```
rmfile("/home/user/myfile.txt");
```

Arguments	1 (filename)
Returns	1 if successful, 0 if not
Description	Deletes a file.

rmpath

```
rmpath("/home/user/x/y/mydir");
```

Arguments	1 (directory path)
Returns	1 if successful, 0 if not
Description	Deletes a path, which may involve deleting multiple directories. If any of the directories are not empty, the deletion will fail.

temp

```
var tempdir = temp();
```

Arguments	0
Returns	String or `undefined`
Description	Returns the first writable temporary directory, depending on the platform, or the current working directory. If the current working directory is not readable and writable, the function will return `undefined`.

Zip Archive Functions

zadd	zmember
zclose	zopen
zextract	zremove
zlist	zwrite

The functions for creating and manipulating zip files work a little differently than the rest of the **Libretto** functions. The function to create or edit a zip archive, `zopen()`, returns a string; this string is called the **zip ID**. The zip ID is required for all zip-related functions, with the exception of `zopen()`; think of the zip ID as something like a file descriptor[28].

zadd

```
// Adds a file to a zip
zadd(zid,"file.txt");

// Adds a directory to a zip
zadd(zid,"/home/user");
```

Arguments	2 (zip ID, file or directory name)
Returns	1 if successful, 0 if not
Description	Adds a file or directory to an open zip archive. If a directory is added, the basename of the directory is retained and used in the zip file.

zclose

```
zclose(zid);
```

Arguments	1 (zip ID)
Returns	1 if successful, 0 if not
Description	Closes a zip archive. Note that you still have to save the archive with `zwrite()` if you want any changes to the zip written to disk.

28 https://en.wikipedia.org/wiki/File_descriptor

zextract

```
zextract(zipid,"/home/user");
```

Arguments	2 (zip ID, directory)
Returns	1 if successful, 0 if not
Description	Extracts a zip archive into a directory.

zlist

```
var ziplist = new Array();
ziplist = zlist(zipid);
```

Arguments	1 (zip ID)
Returns	Array
Description	Returns a list of the contents of a zip archive.

zmember

```
var contents = zmember(zid,"file.txt");
```

Arguments	1 (zip ID, filename)
Returns	String
Description	Extracts the contents of a file inside a zip archive, and returns the contents as a string.

zopen

```
var zipid = zopen("myfile.zip")
```

Arguments	1 (filename)
Returns	String (zip ID)
Description	Opens a zip file for creation or editing. This function returns a string, the "zip ID"; this string should be saved, as it is needed to manipulate the zip archive opened here. The zip ID is randomly generated, and is unique for each zip file opened. If the file exists, and is a zip archive, it will be opened for editing or extraction. If the file does not exist, an in-memory zip archive is created; it will only be written to disk if the `zwrite()` function is called.

zremove

```
zremove(zid,"file.txt");
```

Arguments	1 (zip ID, filename)
Returns	1 if successful, 0 if not
Description	Removes a file from a zip archive.

zwrite

```
zwrite(zid);
```

Arguments	1 (zip ID)
Returns	1 if successful, 0 if not
Description	Writes an open zip archive to disk.

Miscellaneous Functions

base64	termcolor
exec	timestamp
exit	tokens
print	unbase64
sha256	use
sha512	verbose
shutdown	warn

base64

```
var encoded = base64(data);
```

Arguments	1 (data)
Returns	String
Description	Encodes data with Base64, and returns the encoded data.

exec

```
var my_fortune = exec("fortune");
```

Arguments	1 (command to execute)
Returns	String
Description	Executes a command on the host operating system, and returns the result. This is a blocking command; the script will not continue executing until the executed command finishes and returns the result.

exit

```
exit(0);
```

Arguments	0 or 1 (exit code, "1" or "0")
Returns	Nothing
Description	Causes **Libretto** to exit, issuing an exit code of the user's choice (0 or 1), or just exiting without specifying one (which will display a warning and exit with an exit code of 0).

print

```
// Prints "Hello, world!"
print("Hello, world!");

// Each string passed to verbose
// is printed with a newline
print("Multiple","Strings");
```

Arguments	1+ (string)
Returns	Nothing
Description	Prints a string to the console, followed by a newline.

sha256

```
var hash = sha256(data);
```

Arguments	1 (data)
Returns	String
Description	Calculates a SHA256 hash from data, and returns the hash.

sha512

```
var hash = sha512(data);
```

Arguments	1 (data)
Returns	String
Description	Calculates a SHA512 hash from data, and returns the hash.

shutdown

```
// Shutdown an IRC connection
shutdown(server);

// Shutdown a webhook
shutdown("my_webhook");
```

Arguments	1 (server ID or webhook ID)
Returns	Nothing
Description	Sends a shutdown to an IRC server connection (causing the script to disconnect from the IRC server) or a webhook (causing the associated HTTPD to shutdown).

termcolor

```
var text = termcolor("bold red","This is red!");
print(text);
```

Arguments	2 (color and formatting, text)
Returns	String
Description	Uses ANSI color codes to color text send to the console. Any combination of attributes, text colors, or background colors can be passed as the first argument. If the color or formatting passed is not valid, a warning is printed and the original, non-colored text is returned.

Attributes	clear	italic
	bold	underline
	dark	underscore
	faint	reverse
Text Color	black	bright_black
	red	bright_red
	green	bright_green
	yellow	bright_yellow
	blue	bright_blue
	magenta	bright_magenta
	cyan	bright_cyan
	white	bright_white

Background Color	on_black	on_bright_black
	on_red	on_bright_red
	on_green	on_bright_green
	on_yellow	on_bright_yellow
	on_blue	on_bright_blue
	on_magenta	on_bright_magenta
	on_cyan	on_bright_cyan
	on_white	on_bright_white

timestamp

```
var ts = timestamp();
```

Arguments	0
Returns	String
Description	Returns a time stamp (containing the time and date), just like the timestamp printed with messages in verbose and warn modes.

tokens

```
var t = tokens(data);
```

Arguments	1 (string)
Returns	Array
Description	Tokenizes a string; strings are split using white space as a delimiter. White space can be contained in quotes, and quoted text is treated like a single token.

unbase64

```
var decoded = unbase64(data);
```

Arguments	1 (data)
Returns	String
Description	Decodes Base64 encoded data, and returns the decoded data.

use

```
// Load and execute the contents of "bot_functions.js"
use("bot_functions.js");

// Load and execute multiple files
use("bot_functions.js","other_file.js");
```

Arguments	1+ (file names)
Returns	Nothing
Description	Loads the contents of a JavaScript file from disk and executes it. If the file is not found, **Libretto** will look for the file in the standard library location (which is located in "/lib/Libretto" in the **Libretto** installation directory).

verbose

```
verbose("Hello, world!");

// Each string passed to verbose
// is printed with a newline
verbose("Multiple","Strings");
```

Arguments	1+ (string)
Returns	Nothing
Description	If the verbose mode is turned on, prints a string to the console, followed by a newline.

warn

```
warn("Hello, world!");

// Each string passed to warn
// is printed with a newline
warn("Multiple","Strings");
```

Arguments	1+ (string)
Returns	Nothing
Description	If the warnings mode is turned on, prints a string to the console, followed by a newline.

Standard Library

Libretto comes with a standard library: a number of functions and objects written in JavaScript for use in scripts. The standard library can be found in "/lib/Libretto" in the **Libretto** installation directory.

io.js

`io.js` contains functions and objects for creating, editing, and manipulating files and directories.

- **Functions:** `open()`
- **Objects:** Directory, File, Zip

```
use("io.js");

// Create zip archives
var archive = open("my_archive.zip");
archive.Add("libretto.pl");
archive.Add("manual.pdf");
archive.Write();
archive.Close();

// Open and edit text files
var txt = open("my_file.txt");
if(txt){
     print(txt.Contents);
}

// Open directories
var home = open("/home/user");
if(home){
     var contents = home.Files;
     for(var i=0, len=contents.length; i < len; i++){
          print(contents[i]);
     }
}
```

Functions

open

```
var settings_file = open("settings.xml");
if(settings_file) {
     print("File opened successfully!");
     print(settings_file.Content);
} else {
     print("settings.xml doesn't exist!");
}
```

Arguments	1 (file or directory name)
Returns	If the file exists, and is not a zip archive, `open()` will return a `File` object loaded with the file's contents. If the file passed is a directory, `open()` will return a `Directory object`. Otherwise, the function returns `undefined`. If the argument passed ends with ".zip", `open()` will return a `Zip` object for that file if the file exists, or a new, empty `Zip` object if it does not.
Description	Opens a file or directory for editing and analysis; opens zip archives for editing, extraction, or creation.

Objects

File

```
var myfile = new File("file.txt");
if(myfile.Exists){
     print(myfile.Contents);
     print(myfile.Location);
     if(myfile.Read){
          print("file.txt is readable!");
     } else {
          print("file.txt is not readable!");
     }
} else {
     myfile.Contents ="Hello, world!";
     myfile.Save();
}
```

Arguments	1 (filename)
Properties	**Contents** — Contains the file's contents. This property can be edited to alter the file's contents.

Exists	True if the file exists, false if not. Read only.
Mode	The file's permissions. To change the file's permissions, set `Mode` to the new permissions.
Read	True if the file is readable, false if not. Read only.
Write	True if the file is writable, false if not. Read only.
Execute	True if the file is executable, false if not. Read only.
Size	The file's size, in bytes. Read only.
SHA256	The SHA256 hash of the file's contents. Read only.
SHA512	The SHA512 hash of the file's contents. Read only.
Base64	The file's contents, Base64 encoded. Read only.
Basename	The file's basename (that is, the name of the file without the directory it is located in). Read only.
Location	The directory the file is located in. Read only.

Methods

Append	*Arguments*	1 (data)
	Returns	Nothing
	Description	Appends data to the file's contents
Delete	*Arguments*	None
	Returns	`true` if the file was deleted successfully, `false` if not.
	Description	Deletes the file the object represents. This will delete the file on disk; the file's in-memory contents are undisturbed.
Save	*Arguments*	None
	Returns	`true` if the file file save was successful, `false` if not.
	Description	Saves the file the object represents to disk.

Description

The `File` object can be used to create and edit existing files. Any changes to the file object's properties will *not* be saved to disk until the `Save()` method is called.

Zip

```
var archive = new Zip("files.zip");
if(archive.Exists){
     print(archive.Files);
     archive.Extract("/home/user");
} else {
     archive.Add("libretto.pl");
     archive.Write();
}
archive.Close();
```

Arguments 1 (filename)

Properties

Files	An array containing a list of files in the zip archive.
Exists	`true` if the zip archive exists, `false` if not. Read only.

Methods

Add	*Arguments*	1 (file or directory name)
	Returns	`true` if successful, `false` if not.
	Description	Adds a file or directory to a zip archive.
Close	*Arguments*	None
	Returns	`true` if the file was deleted successfully, `false` if not.
	Description	Closes a zip archive; the zip ID will be discarded so no further action on the zip archive is possible.
Extract	*Arguments*	1 (directory)
	Returns	`true` if successful, `false` if not.
	Description	Extracts the contents of a zip archive to the give directory.
Member	*Arguments*	1 (filename)
	Returns	String
	Description	Extracts the contents of a file inside a zip archive into memory, and returns its contents.
Remove	*Arguments*	1 (filename)
	Returns	`true` if successful, `false` if not.
	Description	Removes a file from a zip archive.

	Write	*Arguments*	None
		Returns	`true` if successful, `false` if not.
		Description	Writes the zip archive to disk.
Description	The `Zip` object can be used to create and edit zip archives. Any changes to the zip object's properties will *not* be saved to disk until the `Write()` method is executed.		

Reserved Words

This list includes both **Libretto**-specific reserved words and JavaScript/ES6 reserved words.

arguments	exit	LIGHT_GREY	throw
ARGV	export	mkdir	timestamp
await	extends	mkpath	tokens
base64	false	new	topic
basename	finally	nickname	true
BLACK	flocation	notice	try
BLUE	fmode	null	typeof
bold	for	ORANGE	unbase64
break	forcekick	package	underline
BROWN	forcekill	part	unhook
case	forcemode	PINK	UPTIME
catch	fpermissions	print	use
catdir	fread	private	var
catfile	fsize	protected	VERBOSE
cd	function	public	verbose
chmod	fwrite	PURPLE	void
class	GREEN	raw	warn
client	GREY	RED	WARNINGS
color	hook	return	webhook
const	HOST	rmdir	while
continue	HTTPD	rmfile	WHITE
cwd	if	rmpath	whois
CYAN	implements	say	with
DCC	import	SCRIPTNAME	YELLOW
dcc	in	server	yield
dcc_close	instanceof	sha256	zadd
debugger	interface	sha512	zclose
default	invite	shutdown	zextract
delete	isdir	static	zlist
dirlist	isfile	super	zmember
disconnect	italic	switch	zopen
do	join	TEAL	zremove
else	let	temp	zwrite
enum	LIGHT_BLUE	termcolor	
eval	LIGHT_GREEN	this	

EXAMPLES

Using Command-Line Arguments

Libretto scripts can accept arguments from the command-line. Any arguments passed to **libretto.pl** after the name of the script to run will be passed to your script in an array named ARGV. In this example, we're going to create a script that connects to an IRC server as a client. Our script will take two arguments: the hostname or IP of the IRC server to connect to, and the port we want to use.

First, let's make sure that the person running our script has passed the correct number of arguments to **libretto.pl**:

```
1 if(ARGV.length != 2){
2      print("Error: expecting 2 arguments.");
3      print("Usage: perl libretto.pl " + SCRIPTNAME + " SERVER PORT");
4      exit(1);
5 }
```

With that out of the way, let's pull our hostname and port out of ARGV and use them to create a client connection to the IRC server:

```
1 var server_argument = ARGV[0];
2 var port_argument = ARGV[1];
3
4 print("Connecting to " + server_argument + ":" + server_port + " ... ");
5 client({ server:server_argument, port:port_argument });
```

Now, let's save our script to a file named "arguments.js" and test it!

```
$> perl libretto.pl arguments.js
Error: expecting 2 arguments.
Usage: perl libretto arguments.js SERVER PORT
$> perl libretto.pl arguments.js irc.servercentral.net 6667
Connecting to irc.servercentral.net:6667 ...
```

Using Event Tags

Whenever you hook() an event, you have to label the hook with an event tag, a non-unique string. This allows for scripts to remove multiple hook()s at once. Here's an example of how that's useful.

First, let's create a simple greeter bot. We'll create a server and start it running on port 6667, and then create a bot and connect our bot to the server. On connection, it'll join channel "#foo", and greet anyone who joins it; if spoken to, the bot will respond.

```
 1 // Create the IRC server
 2 server({name:"TagExample",port:6667});
 3
 4 // Create a bot and connect to the server
 5 client({server:"localhost",port:6667,nickname:"greetbot"});
 6
 7 // Hook the "connect" event and join #foo
 8 function on_connect(event){
 9      join(event.ServerID,"#foo");
10 }
11 hook("connect","connect",on_connect);
12
13 // Hook "join" and send our greeting message
14 function on_join(event){
15      say(event.ServerID,event.Channel,"Welcome to #foo, "+event.Nickname+"!");
16 }
17 hook("join","greeter",on_join);
18
19 // Hook public messages and respond when users use our name
20 function on_public(event){
21      if(event.Message.search("greetbot")){
22          say(event.ServerID,event.Channel,"Hi, "+event.Nickname+"!");
23      }
24 }
25 hook("public","greeter",on_public);
```

Our base functionality is complete: we're hook()ing three events, with the greeting function hook()s sharing the tag "greeter". Let's add another hook(): if someone sends "off" as a private message to the bot, it will stop greeting people and responding to its nickname. If someone sends "on", it will resume greeting and responding. Since this is an example, we're not going to worry about security. Anybody can send a private message to the bot and turn on and off the greeting functions. To do this, we're going to hook() the "private" event:

```
function on_private(event){
     if(event.Message == "off"){ unhook("greeter"); }
     if(event.Message == "on"){
          hook("join","greeter",on_join);
          hook("public","greeter",on_public);
     }
}
hook("private","private",on_private);
```

The `unhook()` function takes only a event tag as an argument; since both our "public" and "join" event `hook()`s share the "greeter" tag, `unhook("greeter")` removes both hooks at the same time. There's no function with the equivalent functionality to restore `hook()`s, so we have to restore both `hook()`s manually when we want to "reinstall" them.

DCC Party-Line

In this example, we're going to implement a basic DCC chat party-line. This will be a simple party-line: no channels, and only one command that returns a list of users on the party-line. A party-line is a sort of chatroom inside the bot; all of the users connect directly to the bot, bypassing the IRC server. The bot hosts the chat for all the other users. We're going to use 5 hooks ("dcc-chat-request", "dcc-start", "dcc-incoming", "dcc-done", and "dcc-error"), 4 functions for the hooks, and a few functions that will send chat messages to everyone in the party-line and track users.

First, we need to create a variable to track users; we'll also create an object to represent each user:

```
1 // This array will contain all connected users
2 var ChatUsers = new Array();
3
4 var User = function(cookie,nick) {
5      this.Cookie = cookie;
6      this.Nick = nick;
7 }
```

A "cookie" is an identifier sent to the user when they connect; it's how the bot remembers who is who. Without a user's "cookie", the bot has no way to interact with that user. These "cookies" will be stored in our array, `ChatUsers`. There's no need to generate this value; it'll be automatically generated by the bot on connection.

Now, let's create a function to add users to the user list (`add_chat_user()`) and another function to remove users from the user list (`remove_chat_user()`):

```
 1 function add_chat_user(cookie,nick) {
 2      var newuser = new User(cookie,nick);
 3      ChatUsers.push(newuser);
 4 }
 5
 6 function remove_chat_user(cookie) {
 7      for(var i=0, len=ChatUsers.length; i < len; i++){
 8           if(ChatUsers[i].Cookie == cookie){
 9                ChatUsers.splice(i,1);
10                break;
11           }
12      }
13 }
```

Our user management now works like this: when a user first connects to the partyline, we add the user to our user list by calling `add_chat_user()`. This saves each user's cookie and nick. When a user disconnects from chat, we call `remove_chat_user()` to remove them from the user list. With that out of the way, let's write a function to broadcast chat to everyone on the partyline!

```
1 function chat(serverID,sender,msg){
2      for(var i=0, len=ChatUsers.length; i < len; i++){
3           // Send chat messages to everyone on the partyline
4           // except for the user that sent the chat
5           if(ChatUsers[i].Nick != sender){
6                dcc(serverID,ChatUsers[i].Cookie,msg);
7           }
8      }
9 }
```

This function steps through the user list and sends a chat message to every person on the list *except* the user that sent the chat message. With our support functions and variables all set up, it's time to start writing our hook functions. The first one we're going to write is the easiest:

```
1 function dcc_chat_request(){
2      return true;
3 }
```

`dcc_chat_request()` is the hook function for the "dcc-chat-request" event. It's called every time a user tries to initiate DCC chat with the bot; if this function returns `true`, the bot will accept the chat request, and if the function returns `false` it will reject the request. If we wanted to get fancy, we could implement some kind of user management functionality, like only allowing users with certain nicks to join, but we're not worried about that for this example. We'll just return `true` by default and accept chat requests from anyone who asks.

When a user first enters the chat, we should announce that to the other users. We also need to add the new user to our user list:

```
1 function dcc_start(event){
2
3      // Check the type of DCC session starting, so that we can ignore users
4      // uploading or downloading files from or to the bot
5      if(event.Type=="GET"){ return; }
6      if(event.Type=="SEND"){ return; }
```

```
 7
 8      // Add the user to the user list
 9      add_chat_user(event.Cookie,event.Nick);
10
11      // Let everyone know who joined!
12      chat(event.ServerID,event.Nickname,event.Nickname + \
13           " has joined the partyline!");
14 }
```

Since we've handled connecting to the party-line, let's handle disconnecting from the party-line. When a user disconnects, we need to remove that user from the user list, and let the other users know they disconnected:

```
 1 function dcc_disconnect(event){
 2
 3      // Ignore DCC events from non-chat users
 4      if(event.Type=="GET"){ return; }
 5      if(event.Type=="SEND"){ return; }
 6
 7      // Remove the user from the user list
 8      remove_chat_user(event.Cookie);
 9
10      // Let the other chatters know
11      // We're setting the chat's nick to '0' so
12      // that this error gets sent to EVERYONE, as
13      // no user will have '0' as a nick (and thus,
14      // no user will be skipped when we send this chat)
15      chat(event.ServerID,'0',event.Nickname + " has left the partyline!");
16 }
```

The last step is to handle user chat! We're going to hook "dcc-incoming", so that any chat sent to the bot gets sent to all the users. We're also going to look for any user sending a command; more specifically, we're going to check to see if any user has sent the bot "!who" as a command, and if they have, we're going to send that user a list of users on the party-line. All that's left to do, after all that, is set up our hooks:

```
 1 function dcc_incoming(event){
 2
 3      // If the user sends us "!who" as a message...
 4      if(event.Message=="!who"){
 5
 6           // Compile a user list
 7           var ulist = "Users on the partyline: ";
 8           for(var i=0, len=ChatUsers.length; i < len; i++){
 9                ulist = ulist + ChatUsers[i].Nick + " ";
10           }
```

```
11
12          // Send the user list to the requesting user
13          dcc(event.ServerID,event.Cookie,ulist);
14
15          // There's nothing more to do (we don't want to send "!who" as a chat
16          // message to the other users), so exit the function
17          return;
18      }
19
20      // Send chat to the user list
21      chat(event.ServerID,event.Nickname,event.Nickname + ": " + event.Message);
22  }
23
24  hook("dcc-chat-request","partyline",dcc_chat_request);
25  hook("dcc-start","partyline",dcc_start);
26  hook("dcc-incoming","partyline",dcc_incoming);
27  hook("dcc-done","partyline",dcc_disconnect);
28  hook("dcc-error","partyline",dcc_disconnect);
```

Our party-line is complete! Users just need to initiate a DCC chat session with the bot to join the party-line. Save your code to a file named "party-line.js" and load your bot with **Libretto**:

```
user@host:/home/user$ perl libretto.pl partyline.js
```

Default Libretto Configuration

Example XML file with all defaults set

```
<?xml version="1.0"?>

<settings>

      <nickname>bot</nickname>
      <ircname>libretto irc bot</ircname>
      <username>bot</username>

      <flood-protection>1</flood-protection>
      <ipv6>0</ipv6>
      <verbose>0</verbose>
      <warnings>0</warnings>

      <dcc>
            <enable>1</enable>
            <ports>10000-11000</ports>
            <external-ip>
                  <get>0</get>
                  <host>http://myexternalip.com/raw</host>
                  <set></set>
            </external-ip>
      </dcc>

      <color>
            <enable>1</enable>
            <verbose>bold bright_green</verbose>
            <warnings>bold bright_magenta</warnings>
            <errors>bold bright_red</errors>
      </color>

</settings>
```

Creating a new "settings.xml" if the original has been lost

Open a terminal, navigate to where ever **Libretto** is installed, and issue this command:

```
$> perl libretto.pl --generate-config settings.xml
```

A **Libretto** configuration file will be written to "settings.xml" with all default settings.

LICENSE

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Preamble The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS 0. Definitions. “This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the

making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code. The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system

(if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions. All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law. No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies. You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions. You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date. b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”. c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply,

along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so. A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms. You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need

not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d. A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied

by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms. “Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or b) Requiring

preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or d) Limiting the use for publicity purposes of names of licensors or authors of the material; or e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors. All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination. You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies. You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients. Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate

litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents. A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more

identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom. If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License. Notwithstanding any other provision of this License, you have permission to link or

combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License. The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty. THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16. If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

INDEX