# The Kronos IRC Bot Manual And Guide

**An IRC Bot Development Tool, Powered by Perl, POE, and Google's V8 Javascript Engine**

10/04/18

Version 0.07337

https://github.com/danhetrick/kronos

**Usage** 3
**Requirements** 4
**Installation** 5
**Configuration** 6
  **Command Line Options** 6
  **Configuration Files** 6
**Summary** 7
**Kronic Development Environment** 9
  **Events** 9
  **Delay** 10
  **Available Events** 11
    *action* 11
    *begin* 11
    *connect* 12
    *dcc-chat-request* 12
    *dcc-done* 13
    *dcc-error* 13
    *dcc-incoming* 14
    *dcc-send-request* 15
    *dcc-start* 16
    *exit* 16
    *invite* 17
    *join* 17
    *kick* 18
    *mode* 18
    *nick-changed* 19
    *notice* 19
    *part* 19
    *private* 20
    *public* 20
    *raw* 20
    *raw-out* 21
    *topic* 21
  **Variables** 22
  **Functions** 23
    *Miscellaneous Functions* 23
    *IRC Functions* 29
    *Event Functions* 34
    *File and Directory Functions* 36
    *Zip Archive Functions* 42
  **Objects** 45
    *File* 45
    *Zip* 46
  **Examples** 48
    *DCC Partyline* 48
    *Kronic Shell* 52
**Gnu General Public License** 55
**Index** 67

# USAGE

```
perl kronos.pl [OPTIONS] SCRIPT

--(h)elp                         Display usage information
--(s)erver ADDRESS:PORT          IRC server to connect to
--(p)assword PASSWORD            IRC server password
--(n)ick NICK                    IRC nick to use
--(a)lternate-nick NICK          Nick to use if first choice is taken
--(u)sername USERNAME            IRC username
--(i)rcname IRCNAME              IRCname
--(c)hannel CHANNEL[:PASSWORD]   Channel to join
--(C)onfig FILENAME              Load settings from a config file
--(B)lank-config [FILENAME]      Print a blank config file to STDOUT, or
                                 write one to a file
--(v)erbose                      Turns on verbose mode
--(W)arn                         Turns on warnings mode
--(N)ocolor                      Disable console text colors
--(d)cc-ports PORTS              What ports to use with DCC
                                 PORTS can be a comma delimited list or
                                 a range of numbers (example: 10000-10100)
--file-(D)irectory PATH          Directory where uploaded files will be
                                 placed. Also, the first place the bot will
                                 look for files to send. By default, set to
                                 "/where/kronos/is/installed/files"
--(P)roxy SERVER:PORT            Use a proxy server to connect to IRC
--(I)pv6                         Use IPv6 for connections
--(S)sl                          Connect to IRC using SSL

If not entered, the default server port of '6667' is used for regular
IRC connections, and the default port of '6697' is used for SSL connections.

--channel can be used multiple times to join multiple channels
Single character options can be bundled together
```

# REQUIREMENTS

| Requirement | URL | Description |
|---|---|---|
| Perl | https://www.perl.org/ | Perl programming language |
| POE | http://poe.perl.org/ | Perl Object Environment |
| POE::Component::IRC | https://metacpan.org/pod/POE::Component::IRC | POE IRC library |
| Javascript::V8 | https://metacpan.org/pod/JavaScript::V8 | V8 library interface for Perl |
| libv8 | https://developers.google.com/v8/ | Google's V8 library |
| libv8-dev | https://developers.google.com/v8/ | V8 development files |

For **Kronos** to connect to an IRC server via SSL, an additional module is required:

| Requirement | URL | Description |
|---|---|---|
| POE::Component::SSLify | https://metacpan.org/pod/POE::Component::SSLify | POE SSL library |

Several Perl CPAN modules are bundled with **Kronos**, and don't require installation:

- XML::TreePP
- Term::ANSIColor
- Archive::Zip
- Text::Parsewords

# INSTALLATION

A complete **Kronos** installation looks like this:

| | | |
|---|---|---|
| 📁 core | | |
| | functions.js | The files in this directory are objects, functions, and variables necessary for the **Kronic** environment. They are written in Javascript, and loaded automatically into the namespace of all scripts. |
| | objects.js | |
| | variables.js | |
| 📁 docs | | |
| | 📁 examples | Example bot scripts from the documentation. |
| | kronos-manual.pdf | Documentation for **Kronos** and the **Kronic** development environment. |
| 📁 files | | The default directory **Kronos** will place uploaded files. |
| 📁 lib | | Perl modules necessary for functionality. |
| kronos.pl | | The **Kronos** IRC bot. |

# CONFIGURATION

**Kronos** can be configured two ways:  with command line options, or with a configuration file.

## Command Line Options

This one is pretty self explanitory.  **Kronos** can accept a number or options on the command line to apply various settings.  Call **Kronos** with the `--help` option to see a list of available options (or look at the **Usage** section of this document, on page 3).

## Configuration Files

**Kronos** can load setting from a configuration file.  **Kronos** uses XML for its configuration file format; a valid configuration file has two root elements, `kronos` (which contains **Kronos** specific settings) and `irc` (which contains IRC specific settings).  To generate a blank configuration file for you to edit, use the command line option `--blank-config`. You can pass this option a file name to write your blank configuration to a file, or omit it to cause **Kronos** to print your blank configuration file to STDOUT. The blank configuration file is extensively commented, explaining what each setting is and how to set it.

# SUMMARY

## Kronos is an open-source IRC bot development tool

**Kronos** is an Internet Relay Chat[1] bot[2] which, on its own, does nothing; it's a tool that makes writing an IRC bot quick and easy. All the hard work of writing the code to connect to an IRC server, maintain a connection, and support lots of features (like SSL and proxy server support) is already done for you, so you can focus on your bot, and what you want your bot to do. These are some of the features built-in to **Kronos**:

- Support for IPv6 connections
- Support of Secure Sockets Layer (SSL) connections
- Support for proxy server connections
- Optional XML configuration files
- Support for DCC[3] file transfers and chat

**Kronos** is written in Perl, and can run on any platform that can run Perl. In fact, **Kronos** was developed and tested in Windows 10 and Debian Linux, and should be able to run on OSX without alteration. It uses the Perl Object Environment, a "framework for reactive systems, cooperative multitasking, and network operations", and the POE module POE::Component::IRC to handle IRC functionality.

The star of the show, however, is Google's V8 ECMAScript engine, the same Javascript engine built into the Chrome web browser and the Node.js development environment. Bot scripts are written in a customized Javascript environment named **Kronic** (which stands for **Kronos** **I**RC **C**ode), and these scripts are what powers the **Kronos** IRC bot! Here's an example bot, written in **Kronic**, that greets everyone who enters a channel the bot is in:

```
function on_join(joiner){
     say(joiner.Channel,"Hello, "+joiner.Nick+"! Welcome to "+joiner.Channel+"!");
}

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

This is a complete IRC bot written in 5 lines of code! Every time someone enters a channel the bot is in, the bot will greet them with a public message. So, for example, if your **Kronos** bot is in a channel named "#foo", when a user named "bob" joins the channel, **Kronos** would send a public message to #foo that said "Hello, bob! Welcome to #foo!".

- **Kronic uses Google's V8 ECMAScript engine.**

1 https://en.wikipedia.org/wiki/Internet_Relay_Chat
2 https://en.wikipedia.org/wiki/IRC_bot
3 https://en.wikipedia.org/wiki/Direct_Client-to-Client

- **Kronic is a Javascript/ECMAScript development environment.** If you know how to write code in Javascript, you know how to code on **Kronic**.
- **Kronic is event-driven.** Write **Kronic** functions that get triggered whenever something happens in IRC; events can have an unlimited number of functions bound to them.
- **Kronic can read and write files.** Read, write, copy, and edit files.
  - Built-in SHA256 and SHA512 hashing.
  - Built-in Base64 encoding and decoding.
  - View and edit file permissions.
- **Kronic can create, edit, and delete directories.** Create, delete, and list the contents of directories.
- **Kronic can create, edit, and extract zip archives.** Zip archive support is built into the environment, no external libraries needed.
- **Kronic is cross-platform.** Kronos (and Kronic) can run on any platform Perl can run on that has the V8 library installed, and the Kronic functions are written that way. Functions are included for platform-safe file and directory concatenation, temporary directory handling, and file and directory permissions.

**Kronos** is an all-purpose solution for all your IRC bot needs. If you need the functionality, just write a script for it. This document, **The Kronos IRC Bot Manual And Guide** contains everything you need: usage information, function documentation, examples, and more.

# KRONIC

## Kronos IRC Code Environment

The core of the **Kronic Development Environment** (called **Kronic** in the rest of this section) is Google's V8 ECMAScript engine. This is the same engine used by the Chrome web browser and the Node.js development tool. **Kronic** scripts, then, are ECMAScripts, more commonly known as Javascript. If you know how to write Javascript code, you can code with **Kronic**.

**Kronic** is event-driven. You write functions that you can "hook" to events; when the event occurs, the function is executed. You can hook as many functions as you want to an event, and you can hook any (or all!) of the events. This allows **Kronic** scripts to be somewhat modular; since **Kronos** allows you to load multiple scripts at a time, you can write **Kronic** scripts for a specific functionality, and then load the scripts that implement the functionality you want.

## Events

The basic ideas behind **Kronic** scripting are events[4] and hooks[5].

- **Events.** Every time the bot receives some sort of message from the IRC server, **Kronos** executes an event.
- **Hooks.** A hook is a function that is called whenever a specific event is executed.

Hooks are created with the `hook()` function, which takes three arguments: the name of the event to hook, a "tag" for that event hook, and a reference to the function to be called. The called function is passed a single argument: an object containing information about the event. Once a hook is in place, it will get executed every time the hook's event occurs.

Deleting hooks is easy, too. The second argument to `hook()`, the "tag", is for this purpose. To remove all hooks with a given tag, call `unhook()` with desired the tag as the only argument. Since hook tags don't have to be unique, this allows your scripts to remove multiple tags (or "classes" of tags) at once.

4 https://en.wikipedia.org/wiki/Event_(computing)
5 https://en.wikipedia.org/wiki/Hooking

## Delay

**Kronic** scripts can cause functions to be executed after an arbitrary amount of time by using the `delay()` function, which takes two arguments: the number of seconds to delay, and a function reference. Functions called in this manner can also use the `delay()` function, allowing developers to develop their own time-based events for their scripts.

# Available Events

| | | |
|---|---|---|
| `action` | `dcc-start` | `part` |
| `begin` | `exit` | `private` |
| `connect` | `invite` | `public` |
| `dcc-chat-request` | `join` | `raw` |
| `dcc-done` | `kick` | `raw-out` |
| `dcc-error` | `mode` | `topic` |
| `dcc-incoming` | `nick-changed` | |
| `dcc-send-request` | `notice` | |

## action

```
function on_action_event(action){
     print(action.Nick + " " + action.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. |
| `Nick` | The nick of the user sending the action. |
| `Hostmask` | The hostmask of the user sending the action. |

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

## begin

```
function on_begin_event(){
     print("The begin event was triggered!");
}

hook("begin","begin event",on_begin_event);
```

*Argument Object Properties* None

*Description* Triggers when the script begins, right before the bot attempts to connect to the IRC server.

## connect

```
function on_connect_event(){
     print("Connected to IRC server!");
}

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

| | |
|---|---|
| *Argument Object Properties* | None |
| *Description* | Triggers whenever the bot connects to IRC. |

## dcc-chat-request

```
function on_dcc_chat_request_event(request){
    // Only users with a nick of "bob"
    // can connect to DCC
    if(request.Nick == "bob"){
        // Approve the DCC chat request
        return true;
    } else {
        // Reject the DCC chat request
        return false;
    }
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the requesting user. |
| `Hostmask` | The hostmask of the requesting user. |
| `IP` | The IP address of the requesting user. |
| `Port` | The port the requesting user is connected to. |
| `Type` | The type of the request: SEND, GET, or CHAT. |

*Description*

Triggers whenever a user requests a DCC chat connection from the bot. If you want the bot to approve the request and let the user connect, the function should return `true`; to reject the request, return `false`.

## dcc-done

```
function on_done_event(done){
     print(done.Nick + " disconnected!");
}

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

*Argument Object Properties*

| | |
|---|---|
| `IP` | The IP address of the disconnecting user. |
| `Port` | The port of the disconnecting user. |
| `Nick` | The nick of the user disconnecting |
| `Type` | The type of DCC connection (GET, SEND, CHAT, etc). |
| `Cookie` | The cookie of the disconnecting user. |

*Description* Triggers whenever a user disconnects from DCC.

## dcc-error

```
function on_dcc_error_event(err){
     print(err.Nick + " had an error! (" + err.Error + ")");
}

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

*Argument Object Properties*

| | |
|---|---|
| `IP` | The IP address of the user who had an error. |
| `Port` | The port of the user who had an error. |
| `Nick` | The nick of the user who had an error. |
| `Error` | Description of the error. |
| `Cookie` | The cookie of the user who had an error. |

*Description* Triggers whenever a user disconnects due to error.

## dcc-incoming

```
function on_dcc_incoming_event(chat){
     print(chat.Nick + " says " + chat.Message);
}

hook("dcc-incoming","dcc event",on_dcc_incoming_event);
```

| | | |
|---|---|---|
| *Argument Object Properties* | `IP` | The IP address of the user who sent the chat message. |
| | `Port` | The port of the user who sent the chat message. |
| | `Nick` | The nick of the user who sent the chat message. |
| | `Message` | The sent chat message. |
| | `Cookie` | The cookie of the user who sent the chat message. |
| *Description* | Triggers whenever a user sends a DCC chat message. | |

## dcc-send-request

```
function on_dcc_send_request_event(request){

     // Only users with the nick "joe"
     // can send the bot files
     if(request.Nick == "joe"){

          // Approve the request
          return true;
     } else {
          // Everyone else will have their send
          // request rejected

          // Reject the request
          return false;
     }
}

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

*Argument Object Properties*

| | |
|---|---|
| `IP` | The IP address of the requesting user. |
| `Port` | The port of the requesting user. |
| `Nick` | The nick of the requesting user. |
| `Hostmask` | The hostmask of the requesting user. |
| `Type` | The type of DCC connection requested. |
| `Filename` | The name of the file the user is requesting to send. |
| `Size` | The size of the file, in bytes. |

*Description*

Triggers whenever a user requests permission to send a file to the bot. If you want the bot to approve the request and let the user send the file, the function should return `true`; to reject the request, return `false`.

## dcc-start

```
function on_dcc_start_event(start){
     switch(start.Type) {
          case "SEND":
               print(start.Nick + " started sending a file!");
               break;
          case "CHAT":
               print(start.Nick + " started chatting!");
               break;
          case "GET":
               print(start.Nick + " started downloading a file!");
               break;
     }
}

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

*Argument Object Properties*

| | |
|---|---|
| `IP` | The IP address of the user who is starting the session. |
| `Port` | The port of the user who is starting the session. |
| `Nick` | The nick of the user who is starting the session. |
| `Type` | The type of DCC session (SEND, GET, CHAT, etc) |
| `Cookie` | The cookie of the user who is starting the session. |

*Description* Triggers whenever a user starts a DCC session with the bot.

## exit

```
function on_exit_event(ex){
     print("Exited with code " + ex.Code);
}

hook("exit","exit event",on_exit_event);
```

*Argument Object Properties*

| | |
|---|---|
| `Code` | The exit code, either 1 (exit for error) or 0 (no error) |

*Description* Triggers whenever a script uses the `exit()` function; the event is executed immediately before the actual exit.

## invite

```
function on_invite_event(invitation){
     print(invitation.Nick + " invited the bot to " + invitation.Channel);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Channel` | The channel the bot is invited to. |
| `Hostmask` | The hostmask of the user inviting the bot. |
| `Nick` | The nick of the user inviting the bot. |

*Description* Triggers whenever a user sends a channel invite.

## join

```
function on_join_event(joiner){
     print(joiner.Nick + " joined " + joiner.Channel);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Channel` | The channel the user joined. |
| `Hostmask` | The hostmask of the joining user. |
| `Nick` | The nick of the joining user. |

*Description* Triggers whenever a user joins a channel in the bot’s presence

## kick

```
function on_kick_event(kicker){
     if(kicker.Reason == ""){
          print(kicker.Nick + " kicked " + kicker.Target + \
          " from " + kicker.Channel);
     } else {
          print(kicker.Nick + " kicked " + kicker.Target + \
          " from " + kicker.Channel + " (" + kicker.Reason + ")");
     }
}

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

*Argument Object Properties*

| | |
|---|---|
| `Channel` | The channel the the target was kicked from. |
| `Hostmask` | The hostmask of the kicking user. |
| `Nick` | The nick of the kicking user. |
| `Target` | The user being kicked. |
| `Reason` | The reason the user was kicked. |

*Description* Triggers whenever a user is kicked from a channel in the bot’s presence.

## mode

```
function on_mode_event(event){
     print(event.Nick + " set mode " + event.Mode + " " + \
     event.Arguments + " on " + event.Target);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the user setting the mode. |
| `Hostmask` | The hostmask of the user settings the mode. |
| `Mode` | The mode set. |
| `Arguments` | The arguments used for the mode set. |
| `Target` | The user the mode was set on. |

*Description* Triggers whenever a mode is set in the bot’s presence.

## nick-changed

```
function on_newnick_event(newnick){
     print(newnick.Nick + " is now known as " + newnick.New);
}

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

*Argument Object Properties*

| | |
|---|---|
| `New` | The user’s new nick. |
| `Hostmask` | The hostmask of the nick changing user. |
| `Nick` | The old nick of the user. |

*Description* Triggers whenever a user changes their nick in the bot’s presence

## notice

```
function on_notice_event(msg){
     print(msg.Nick + " sent a notice to " + msg.Targets + ": " + msg.Message);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the notice sender. |
| `Hostmask` | The hostmask of the notice sender. |
| `Targets` | The targets of the notice. |
| `Message` | The notice message. |

*Description* Triggers whenever the bot receives a notice.

## part

```
function on_part_event(parter){
     print(parter.Nick + " left " + parter.Channel);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the user parting. |
| `Hostmask` | The hostmask of the user parting. |
| `Channel` | The channel the user left. |

*Description* Triggers whenever a user parts a channel in the bot’s presence.

## private

```
function on_private_event(msg){
     print(msg.Nick + " sent a private message: " + msg.Message);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the user sending the message. |
| `Hostmask` | The hostmask of the user sending the message. |
| `Message` | The message sent. |

*Description* Triggers whenever the bot receives a private message.

## public

```
function on_public_event(msg){
     print(msg.Channel + " " + msg.Nick + ": " + msg.Message);
}

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

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the user sending the message. |
| `Hostmask` | The hostmask of the user sending the message. |
| `Channel` | The channel the message was sent to. |
| `Message` | The message sent. |

*Description* Triggers whenever the bot receives a public message.

## raw

```
function on_raw_event(text){
     print("Message received from server: " + text);
}

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

*Argument* String (the "raw" contents of the server message received)

*Description* Triggers whenever the bot receives a message from the server; this event will receive *all* messages, even ones handled by other events. It is not parsed or changed in any way.

## raw-out

```
function on_raw_out_event(text){
     print("Message sent to server: " + text);
}

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

| | |
|---|---|
| *Argument* | String (the "raw" contents of the message sent to the server) |
| *Description* | Triggers whenever the bot sends a message to the IRC server; this event will receive *all* outbound messages. |

## topic

```
function on_topic_event(event){
     print(event.Nick + " set the topic in " + event.Channel +\
     ": " + event.Topic);
}

hook("topic","topic event",on_topic_event);
```

*Argument Object Properties*

| | |
|---|---|
| `Nick` | The nick of the user setting the topic. |
| `Hostmask` | The hostmask of the user setting the topic |
| `Channel` | The channel where the topic was set. |
| `Topic` | The channel's topic. |

*Description* Triggers whenever a channel's topic is changed in the bot's presence.

# Variables

| Variable Name | Read Only? | Updated During Runtime | Value |
|---|---|---|---|
| `NICKNAME` | *No* | *Yes* | The bot's current nick. |
| `IRCNAME` | *Yes* | *No* | The bot's IRCname. |
| `USERNAME` | *Yes* | *No* | The bot's username. |
| `IRC_SERVER` | *Yes* | *No* | The IRC server connected to. |
| `IRC_PORT` | *Yes* | *No* | The IRC server port connected to. |
| `VERBOSE` | *Yes* | *No* | True if verbosity is turned on, false if not. |
| `WARN` | *Yes* | *No* | True if warning messages are turned on, false if not. |
| `SSL` | *Yes* | *No* | True if the bot is connected to the IRC server via SSL, false if not. |
| `IPV6` | *Yes* | *No* | True is the bot is using IPv6 to connect, false if not. |
| `WHITE` | *Yes* | *No* | "00". For use with the `color()` function. |
| `BLACK` | *Yes* | *No* | "01". For use with the `color()` function. |
| `BLUE` | *Yes* | *No* | "02". For use with the `color()` function. |
| `GREEN` | *Yes* | *No* | "03". For use with the `color()` function. |
| `RED` | *Yes* | *No* | "04". For use with the `color()` function. |
| `BROWN` | *Yes* | *No* | "05". For use with the `color()` function. |
| `PURPLE` | *Yes* | *No* | "06". For use with the `color()` function. |
| `ORANGE` | *Yes* | *No* | "07". For use with the `color()` function. |
| `YELLOW` | *Yes* | *No* | "08". For use with the `color()` function. |
| `LIGHT_GREEN` | *Yes* | *No* | "09". For use with the `color()` function. |
| `TEAL` | *Yes* | *No* | "10". For use with the `color()` function. |
| `CYAN` | *Yes* | *No* | "11". For use with the `color()` function. |
| `LIGHT_BLUE` | *Yes* | *No* | "12". For use with the `color()` function. |
| `PINK` | *Yes* | *No* | "13". For use with the `color()` function. |
| `GREY` | *Yes* | *No* | "14". For use with the `color()` function. |
| `LIGHT_GREY` | *Yes* | *No* | "15". For use with the `color()` function. |

# Functions

## Miscellaneous Functions

| | |
|---|---|
| `base64` | `termcolor` |
| `exec` | `timestamp` |
| `exit` | `tokens` |
| `load` | `trim` |
| `prettyuptime` | `unbase64` |
| `print` | `uptime` |
| `sha256` | `warn` |
| `sha512` | `verbose` |
| `shuffle` | |

### base64

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

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

### exec

```
// Execute a program on the host OS, and returns the result
var message = exec("fortune -s");
```

| | |
|---|---|
| *Arguments* | 1 (command line to execute) |
| *Returns* | Array |
| *Description* | Executes a command on the host operating system, and returns any command line output in an array.  This is a blocking command (your script will not continue until the issued command returns output). |

## exit

```
exit();
```

|  |  |
|---|---|
| *Arguments* | 0 or 1 (exit code, "1" or "0") |
| *Returns* | Nothing |
| *Description* | Causes Kronos to exit, issuing an exit code of the user’s choice (0 or 1), or just exiting without specifying one (which will exit with an exit code of 0). |

## load

```
if(load("myscript.js")){
     print("myscrips.js loaded successfully!");
} else {
     print("Error loading myscript.js!");
}
```

|  |  |
|---|---|
| *Arguments* | 1 (filename) |
| *Returns* | True if the file was loaded successfully, false if not. |
| *Description* | Loads a Javascript file from disk and executes it in the global namespace. |

## prettyuptime

```
var uptime = prettyuptime();
```

|  |  |
|---|---|
| *Arguments* | 0 |
| *Returns* | String |
| *Description* | Returns the bot’s uptime in a human readable format.  If the bot’s uptime, for example, is 120 seconds, **`prettyuptime()`** will return "2 minute". |

## 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. |

## shuffle

```
var scrambled = shuffle(myArray);
```

| | |
|---|---|
| *Arguments* | 1 (array) |
| *Returns* | Array |
| *Description* | Randomly re-orders an array, and returns the shuffled array. |

## tokens

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

| | |
|---|---|
| *Arguments* | 1 (string) |
| *Returns* | Array |
| *Description* | Tokenizes a string; strings are split using whitespace as a delimiter. Whitespace can be contained in quotes; quoted text is treated like a single token. |

## timestamp

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

| | |
|---|---|
| *Arguments* | 0 |
| *Returns* | String |
| *Description* | Returns a time stamp (containing the time and data of the host), just like the timestamp used with verbose and warn modes. |

## termcolor

```
var text = termcolor("bold red","This is red!");
print(text);
text = termcolor("bold black on_white", "Bold, and black on white!");
```

| | |
|---|---|
| *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 |

## trim

```
// Returns "hello"
var text = trim("     hello    ");
```

| | |
|---|---|
| *Arguments* | 1 (string) |
| *Returns* | String |
| *Description* | Trims all leading and trailing whitespace from a string. |

## unbase64

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

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

## uptime

```
var bot_uptime = uptime();
```

| | |
|---|---|
| *Arguments* | 0 |
| *Returns* | Number |
| *Description* | Returns the bot’s uptime, in seconds. |

## verbose

```
// Prints "Hello, world!" if verbose mode is turned on
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 flag is set, prints a string to the console, followed by a newline. |

## warn

```
// Prints "Hello, world!" if warning mode is turned on
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 flag is set, prints a string to the console, followed by a newline. |

### IRC Functions

|  |  |
|---|---|
| `bold` | `oper` |
| `color` | `part` |
| `dcc` | `quit` |
| `invite` | `raw` |
| `italic` | `say` |
| `join` | `send` |
| `mode` | `topic` |
| `nick` | `underline` |
| `notice` | `who` |

## bold

```
// Create a string, and make it bold
var example = bold("Hello, world!");

// Send our string to IRC!
say("#foo",example);
```

|  |  |
|---|---|
| *Arguments* | 1 (text) |
| *Returns* | String |
| *Description* | Formats text with the mIRC control code for "bold" text. |

## color

```
// This will display in red and yellow
var stuff = "This will be printed in " +\
color(RED,YELLOW,"red") + " and " + color(YELLOW,RED,"yellow");
```

|  |  |
|---|---|
| *Arguments* | 3 (foreground color, background color, text) |
| *Returns* | String |
| *Description* | Formats text with mIRC color tags.  A list of available color-related built-in variables is follows. |

#### Text Colors

```
WHITE          YELLOW
BLACK          LIGHT_GREEN
BLUE           TEAL
GREEN          CYAN
RED            LIGHT_BLUE
BROWN          PINK
PURPLE         GREY
ORANGE         LIGHT_GREY
```

## dcc

```
// Send a message to someone connected to the bot’s DCC chat
dcc(cookie,"Hello!");
```

| | |
|---|---|
| *Arguments* | 2 (DCC "cookie", string) |
| *Returns* | Nothing |
| *Description* | Sends a DCC chat message. The "cookie" is a bit of information gained from all DCC events (other than "dcc-request"), and represents an ID for an individual DCC connection. |

## invite

```
// Invite our friend bob to #foo
invite("bob","#foo");
```

| | |
|---|---|
| *Arguments* | 2 (nick, channel) |
| *Returns* | String |
| *Description* | Sends a channel invite to a user. |

## italic

```
// Create a string, and make it italicized
var example = italic("Hello, world!");

// Send our string to IRC!
say("#foo",example);
```

| | |
|---|---|
| *Arguments* | 1 (text) |
| *Returns* | String |
| *Description* | Formats text with the mIRC control code for "italic" text. |

## mode

```
// Op user Bob
mode("+o #foo");
```

| | |
|---|---|
| *Arguments* | 1 (mode to execute) |
| *Returns* | Nothing |
| *Description* | Causes the bot to set a mode on a channel[6] or a user[7]. |

6 https://www.unrealircd.org/docs/Channel_modes
7 https://www.unrealircd.org/docs/User_Modes

## nick

```
// Create a string, and make it bold
nick("joe");
```

| | |
|---|---|
| *Arguments* | 1 (new nick) |
| *Returns* | Nothing |
| *Description* | Changes the bot’s nick. |

## notice

```
notice("#foo","Hello world!");
```

| | |
|---|---|
| *Arguments* | 2 (nick or channel, message) |
| *Returns* | Nothing |
| *Description* | Sends a notice. |

## oper

```
oper("my_username","my_password");
```

| | |
|---|---|
| *Arguments* | 2 (username, password) |
| *Returns* | Nothing |
| *Description* | Logs the bot into an operator account. |

## part

```
part("#foo");

// Leave a channel, displaying the reason why the bot is parting
part("#foo","I need to reboot!");
```

| | |
|---|---|
| *Arguments* | 1 (channel name) or 2 (channel name, reason) |
| *Returns* | Nothing |
| *Description* | Causes the bot to leave a channel. |

## quit

```
// Disconnect from the IRC server
quit();

// Quit, and let the server know why
quit("I’m going to bed");
```

| | |
|---|---|
| *Arguments* | 0 or 1 (text) |
| *Returns* | Nothing |
| *Description* | Disconnects from the IRC server, displaying an optional reason for the quit. Duster will exit after the quit command is issued. |

## raw

```
// Send a notice to everyone in #foo
raw("NOTICE #foo: Hello, everybody!");

// Change the bot’s nick to "MyNewNick"
raw("NICK MyNewNick");
```

| | |
|---|---|
| *Arguments* | 1+ (string) |
| *Returns* | Nothing |
| *Description* | Sends a "raw" command to the IRC server. |

## say

```
// Send a public message
say("#foo","Hello, everyone!");
```

| | |
|---|---|
| *Arguments* | 2 (user nick or channel name,message) |
| *Returns* | Nothing |
| *Description* | Sends a chat message to the IRC server. |

## send

```
// Send a file to a user
send("bob_the_user","/home/user/file.txt");
```

| | |
|---|---|
| *Arguments* | 2 (user nick, filename) |
| *Returns* | Nothing |
| *Description* | Sends a file to a user via DCC. |

## topic

```
topic("#foo","The topic is the Kronos IRC bot");
```

| | |
|---|---|
| *Arguments* | 2 (channel name, new topic) |
| *Returns* | Nothing |
| *Description* | Sets a channel's topic. |

## underline

```
// Create a string, and underline it
var example = underline("Hello, world!");

// Send our string to IRC!
say("#foo",example);
```

| | |
|---|---|
| *Arguments* | 1 (text) |
| *Returns* | String |
| *Description* | Formats text with the mIRC control code for "underlined" text. |

## who

```
var userlist = new Array();

// Get a list of all users the bot can "see"
userlist = who();

// Get a list of all users in a specific channel
userlist = who("#foo");
```

| | |
|---|---|
| *Arguments* | 0 or 1 (channel name) |
| *Returns* | Array |
| *Description* | Gets a list of users in all channels the bot is present in (if no argument is passed) or all users in a specific channel the bot is present in (if passed the channel name as an argument). What's in the array depends on how the command is called:<br><br>• **With no arguments.** Each entry in the array has a channel name, a space, and a user name, for each user in all channels.<br>• **With a channel name as an argument.** Each entry in the array is a user name present in that channel. |

## Event Functions

| | |
|---|---|
| `delay` | `unhook` |
| `hook` | |

### delay

```
// Create a function to run in one minute
function one_minute_later(){
     print("It’s one minute later!");
}

// Tell the bot to run our function in one minute
delay(60,one_minute_later);
```

| | |
|---|---|
| *Arguments* | 2 (seconds to delay, function reference) |
| *Returns* | Nothing |
| *Description* | Executes a function after a set number of seconds. |

### hook

```
// Create a function that will be called whenever
// the bot receives a public message
function my_public_handler(ARGS){
     print(ARGS.Nick + " spoke in " + ARGS.Channel + ": " + ARGS.Message);
}

// Now, hook the public message event to our function
hook("public","my_hook",my_public_handler);
```

*Arguments* 3 (event name, hook ID, function reference)

*Returns* Nothing

*Description* Creates a hook for an IRC event. A hook ID is a string that can be attached to an event hook; this allows for `unhook()` to remove multiple hooks at once. Hook IDs do not have to be unique. There are 22 events that can be hooked:

| Event Name | Description |
|---|---|
| `begin` | Occurs when the script takes full control of the bot. |
| `public` | Occurs whenever the bot receives a public message. |
| `private` | Occurs whenever the bot receives a private message. |
| `part` | Occurs whenever someone leaves a channel the bot is in. |
| `join` | Occurs whenever someone joins a channel the bot is in. |

| Event Name | Description |
|---|---|
| `connect` | Occurs when the bot first connects to IRC. |
| `dcc-chat-request` | Occurs when a DCC chat session is requested. The hook’s function must return true to accept the DCC chat request, or false to reject the chat request. |
| `dcc-send-request` | Occurs when a DCC send session is requested (a user is trying to upload a file). The hook’s function must return true to accept the DCC send request, or false to reject the DCC send request. |
| `dcc-start` | Occurs when a DCC chat session begins. |
| `dcc-incoming` | Occurs when DCC chat is received. |
| `dcc-done` | Occurs when a DCC chat session is closed. |
| `dcc-error` | Occurs when a DCC chat session has an error. |
| `topic` | Occurs when a channel’s topic is set or sent by the server. |
| `action` | Occurs when a CTCP action message is received. |
| `mode` | Occurs when a mode is set in the bot’s presence. |
| `kick` | Occurs when a user is kicked from a channel the bot is present in. |
| `invite` | Occurs when a user invites the bot to a channel. |
| `nick-changed` | Occurs when a user changes their nick in the bot’s presence. |
| `notice` | Occurs when the bot receives a notice. |
| `raw` | Occurs whenever the bot receives *any* message from the IRC server. |
| `raw-out` | Occurs when the bot sends *any* message to the IRC server. |
| `exit` | Occurs when Kronos exits. |

Functions hooked to an IRC event are passed an object when triggered; the contents of the object depends on the event being hooked.  (See *Events*, on page 9).

## unhook

```
// Remove a hook
unhook("my_hook_ID");
```

| | |
|---|---|
| *Arguments* | 1 (hook ID) |
| *Returns* | Nothing |
| *Description* | Removes one or more event hooks. |

## File and Directory Functions

| | |
|---|---|
| `basename` | `fsize` |
| `catdir` | `fwrite` |
| `catfile` | `isdir` |
| `cd` | `isfile` |
| `chmod` | `mkdir` |
| `cwd` | `mkpath` |
| `dirlist` | `rmdir` |
| `flocation` | `rmfile` |
| `fmode` | `rmpath` |
| `fpermissions` | `temp` |
| `fread` | |

### 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+ (strings or arrays) |
| *Returns* | String |
| *Description* | Concatenates directory names (as strings or arrays of strings) 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+ (strings or arrays) |
| *Returns* | String |
| *Description* | Concatenates one or more directory names (as strings or arrays of strings) and a file name into a valid file 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 script's working directory to a new directory. |

## chmod

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

| | |
|---|---|
| *Arguments* | 1 (directory or file name) |
| *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) or 2 (directory name and filter) |
| *Returns* | Array |
| *Description* | Returns a list of files in a directory.  If using a filter, * is a wildcard. |

## flocation

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

| | |
|---|---|
| *Arguments* | 1 (file name) |
| *Returns* | String |
| *Description* | Returns the location (directory) a file is in. |

## fmode

```
var kronos_mode = fmode("kronos.pl");
```

| | |
|---|---|
| *Arguments* | 1 (directory or file name) |
| *Returns* | String |
| *Description* | Returns a file or directory's mode (permissions). |

## fpermissions

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

| | |
|---|---|
| *Arguments* | 1 (directory or file name) |
| *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 (file name) |
| *Returns* | String |
| *Description* | Reads the contents of a file, and returns them. |

## fsize

```
var kronos_size = fsize("kronos.pl");
```

| | |
|---|---|
| *Arguments* | 1 (file name) |
| *Returns* | String |
| *Description* | Returns the file's size in bytes. |

## fwrite

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

| | |
|---|---|
| *Arguments* | 2 (file name, 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 (file or directory) |
| *Returns* | 1 if the passed argument exists and is a directory, 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 (file or directory) |
| *Returns* | 1 if the passed argument exists and is a file, 0 if not. |
| *Description* | Determines if a string is an existing file name. |

## 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 the passed argument exists and is a file, 0 if not. |
| *Description* | Deletes a directory. If the directory is not empty, the operation will fail. |

## rmfile

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

| | |
|---|---|
| *Arguments* | 1 (directory) |
| *Returns* | 1 if the passed argument exists and is a file, 0 if not. |
| *Description* | Deletes a file. |

## rmpath

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

| | |
|---|---|
| *Arguments* | 1 (directory) |
| *Returns* | 1 if the passed argument exists and is a file, 0 if not. |
| *Description* | Deletes a path, which may involve deleting multiple directories. If any of the directories are not empty, the operation 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 **Kronic** functions. The function to create or edit a zip archive, **`zopen()`**, returns a string; this string is called the "zip identification" (or "zip ID"). The zip ID is required for all zip-related functions, with the exception of **`zopen()`**; think of the zip ID as a file descriptor[8].

### 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 a directory to a 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. |

### zextract

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

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

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

## zlist

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

| | |
|---|---|
| *Arguments* | 1 (zip ID) |
| *Returns* | Array |
| *Description* | Lists the contents of a zip archive. |

## zmember

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

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

## zopen

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

| | |
|---|---|
| *Arguments* | 1 (file name) |
| *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* | 2 (zip ID, file name) |
| *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. |

## Objects

### File

**File** is a class you can use to read, write, analyze, create, and delete files.  It has 11 properties, and 3 methods.

## 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 exeutable, 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 content. |

*Methods*

| | | |
|---|---|---|
| **`Delete`** | *Arguments* | None |
| | *Returns* | True if the file was deleted successfully, false if the deletion failed. |
| | *Description* | Deletes the file the object is attached to. This will only delete the file; in-memory content is still retained. |
| **`Save`** | *Arguments* | None |
| | *Returns* | True if the save was successful, false if the save failed. |
| | *Description* | Saves the file the object is attached to 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 executed. `File`'s functionality is written in Javascript and is contained in **`objects.js`** in the "core" directory.

## Zip

```
var archive = new Zip("files.zip");
if(archive.Exists){
     print(archive.Files);
     archive.Extract("/home/user");
} else {
     archive.Add("kronos.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 a directory to a zip archive. |
| **`Close`** | *Arguments* | None |
| | *Returns* | True if successful, 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 the save was successful, false if the save failed. |
| | *Description* | Extracts the contents of a zip to the given directory. |

*Methods*

| | | |
|---|---|---|
| **`Member`** | *Arguments* | 1 (filename) |
| | *Returns* | String |
| | *Description* | Extracts the contents of a file inside a zip archive, and returns its contents. |
| **`Remove`** | *Arguments* | 1 (filename) |
| | *Returns* | True if the save was successful, false if the save failed. |
| | *Description* | Removes a file from a zip archive. |
| **`Write`** | *Arguments* | None |
| | *Returns* | True if the save was successful, false if the save failed. |
| | *Description* | Write 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. `Zip`'s functionality is written in Javascript and is contained in `objects.js` in the "core" directory.

**Kronic** also features a function named `open()`, written to make using the `File` object a bit more intuitive.

## open

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

| | |
|---|---|
| *Arguments* | 1 (filename) |
| *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 exists, and is a zip archive, `open()` will return a `Zip` object for that file. If the file passed is a directory, `open()` will return a `Directory` object. Otherwise, the function returns "undefined". |
| *Description* | Opens a file for editing or analysis. `open()`'s functionality is written in Javascript and is contained in `functions.js` in the "core" directory. |

# EXAMPLES

*Source code for all examples can be found in kronos/docs/examples*

## DCC Partyline

*The source code for this example can be found in docs/examples/partyline.js*

In this example, we're going to implement a basic DCC chat partyline. This will be a simple partyline: no channels, and only one command that returns a list of users on the partyline. A partyline 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"), 5 functions for the hooks, and a few functions that will send chat messages to everyone in the partyline and track users.

Let's start with something simple. First, we need to create a variable to track users; we'll also create an object to represent each user:

```
// This array will contain all connected users
var ChatUsers = new Array();

var User = function(cookie,nick) {
     this.Cookie = cookie;
     this.Nick = nick;
}
```

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()`):

```
function add_chat_user(cookie,nick) {
     var newuser = new User(cookie,nick);
     ChatUsers.push(newuser);
}

function remove_chat_user(cookie) {
     for(var i=0, len=ChatUsers.length; i < len; i++){
          if(ChatUsers[i].Cookie == cookie){
               ChatUsers.splice(i,1);
               break;
          }
     }
}
```

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!

```
function chat(sender,msg){
     for(var i=0, len=ChatUsers.length; i < len; i++){

          // Send chat messages to everyone on the partyline
          // except for the user that sent the chat
          if(ChatUsers[i].Nick != sender){
               dcc(ChatUsers[i].Cookie,msg);
          }

     }
}
```

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:

```
function dcc_chat_request(){
     return true;
}
```

`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:

```
function dcc_start(args){

     // Check the type of DCC session starting, so that we can ignore users
     // uploading or downloading files from or to the bot
     if(args.Type=="GET"){ return; }
     if(args.Type=="SEND"){ return; }

     // Add the user to the user list
     add_chat_user(args.Cookie,args.Nick);

     // Let everyone know who joined!
     chat(args.Nick,args.Nick + " has joined the partyline!");
}
```

Since we've handled connecting to the partyline, let's handle disconnecting from the partyline. When a user disconnects, we need to remove that user from the user list, and let the other users know they disconnected. We'll use two hook events for this: one for when a user disconnects willingly, and one for if they are disconnected due to an error:

```
function dcc_done(args){

     // Ignore DCC events from non-chat users
     if(args.Type=="GET"){ return; }
     if(args.Type=="SEND"){ return; }

     // Remove the user from the user list
     remove_chat_user(args.Cookie);

     // Let the other chatters know
     chat(args.Nick,args.Nick + " has left the partyline!");
}

function dcc_error(args){

     // Ignore DCC events from non-chat users
     if(args.Type=="GET"){ return; }
     if(args.Type=="SEND"){ return; }

     // Remove the user from the user list
     remove_chat_user(args.Cookie);
     // Let the other chatters know
     // We’re setting the chat’s nick to ‘0’ so
     // that this error gets sent to EVERYONE, as
     // no user will have ‘0’ as a nick (and thus,
     // no user will be skipped when we send this chat)
     chat('0',args.Nick + " has left the partyline! (" + args.Error + ")");
}
```

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 partyline. All that’s left to do, after all that, is set up our hooks:

```
function dcc_incoming(args){

     // If the user sends us "!who" as a message...
     if(args.Message=="!who"){

          // Compile a user list
          var ulist = "Users on the partyline: ";
          for(var i=0, len=ChatUsers.length; i < len; i++){
               ulist = ulist + ChatUsers[i].Nick + " ";
          }

          // Send the user list to the requesting user
          dcc(args.Cookie,ulist);

          // There’s nothing more to do (we don’t want to send "!who" as a chat
          // message to the other users), so exit the function
          return;
     }

     // Send chat to the user list
     chat(args.Nick,args.Nick + ": " + args.Message);
}

hook("dcc-chat-request","partyline",dcc_chat_request);
hook("dcc-start","partyline",dcc_start);
hook("dcc-incoming","partyline",dcc_incoming);
hook("dcc-done","partyline",dcc_done);
hook("dcc-error","partyline",dcc_error);
```

Our partyline is complete! Users just need to initiate a DCC chat session with the bot to join the partyline. Save your code to a file named "partyline.js" (or copy the file in the docs/examples/ directory) and load your bot with **Kronos**:

```
user@host:/home/user$ perl kronos.pl -C settings.xml partyline.js
```

# Kronic Shell

*The source code for this example can be found in docs/examples/kshell.js*

In this example, we're going to create a ruidimentary **Kronic** shell for a bot. To use it, connect with the bot via DCC chat, send it the password, and then send **Kronic** commands/functions to the bot, which it will execute. Please note: the security in this example is really, really weak. This is an example to show **Kronic** use only.

First, we need to create two variables. One to store our password, and one to track if someone is logged in or not:

```
// This variable will contain our password.
// You should probably change this :-)
var PASSWORD = "changeme";

// This variable will track if someone is logged in or not.
// When they log in, we set this to the user's cookie.
// When they log out (or disconnect) we reset it to undefined.
var LOGGED_IN = undefined;
```

Now, we're going to make it so that only one user can be logged in at a time. When someone is logged it, the bot will reject all incoming DCC chat requests. We'll handle this with a hook to "dcc-chat-request":

```
// dcc-chat-request event
function dcc_chat_request(){
     if(LOGGED_IN){
          return false;
     } else {
          return true;
     }
}

// Now, hook the event
hook("dcc-chat-request","kshell",dcc_chat_request);
```

We’ll hook "dcc-incoming" to handle logging in and command execution. Since our shell will only reject users once someone is logged in, it’s a realistic possibility that more than one user might try to connect to the bot and unsuccessfully log in before an authorized user does. We’ll mitigate that by allowing multiple users to send chat to the bot, but once someone is logged in, no one else can, even if they have the password.

```
// dcc-incoming event
function dcc_incoming(args){

     // Scan input for the password
     if(args.Message == PASSWORD){

          // Someone might have already connected to the
          // bot, but not logged in. This is to make sure
          // only one user is logged in at a time.
          if(LOGGED_IN){
               dcc(args.Cookie,"Sorry, someone is already logged in");
          } else {

               // Log the user in
               LOGGED_IN = args.Cookie;
               dcc(args.Cookie,"Logged in!");
          }
          return;
     }

     // If you’re not logged in, you can go no further :-)
     if(LOGGED_IN != args.Cookie){
          return;
     }

     // Use the "eval()" function to execute code
     eval(args.Message);
}

// Hook the event
hook("dcc-incoming","kshell",dcc_incoming);
```

Now, we’re going to use a bit of a hack: we’re going to hook the same function to two different events.  We’re going to hook "dcc-done" and "dcc-error" to log out any logged in user:

```
// dcc-done and dcc-error events
function dcc_logout(args){
     if(args.Type=="GET"){ return; }
     if(args.Type=="SEND"){ return; }

     // If the disconnecting user is the one logged in...
     if(args.Cookie == LOGGED_IN){
          // ...log them out.
          LOGGED_IN = undefined;
     }
}

hook("dcc-done","kshell",dcc_logout);
hook("dcc-error","kshell",dcc_logout);
```

Load your script into **Kronos**, and your shell is ready to use!  Just send the bot the same commands you would in your **Kronic** scripts, and watch **Kronos** do your bidding!

# GNU GENERAL PUBLIC LICENSE 3.0

"In real open source, you have the right to control your own destiny." - Linus Torvalds

## Gnu General Public License
Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

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.

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).

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.

# KRONIC INDEX

*Functions and objects and variables, oh my!*

base64........................22
basename......................35
bold..........................28
catdir........................35
catfile.......................35
cd............................36
chmod.........................36
color.........................28
cwd...........................36
dcc...........................29
delay.........................33
dirlist.......................36
exec..........................22
exit..........................23
File object...................44
flocation.....................37
fmode.........................37
fpermissions..................37
fread.........................37
fsize.........................37
fwrite........................38
hook..........................33
invite........................29
isdir.........................38
isfile........................38
italic........................29
load..........................23
mkdir.........................38
mkpath........................39
mode..........................29
nick..........................30
notice........................30
open..........................46
oper..........................30
part..........................30
prettyuptime..................23
print.........................23
quit..........................31
raw...........................31
rmdir.........................39
rmfile........................39
rmpath........................39
say...........................31
send..........................31
sha256........................24
sha512........................24
shuffle.......................24
temp..........................40
termcolor.....................25
timestamp.....................24
tokens........................24
topic.........................32
trim..........................26
unbase64......................26
underline.....................32
unhook........................34
uptime........................26
verbose.......................26
warn..........................27
who...........................32
zadd..........................41
zclose........................41
zextract......................41
Zip object....................45
zlist.........................42
zmember.......................42
zopen.........................42
zremove.......................42
zwrite........................43