# HexChat Documentation

***Release 2.12.0***

**TingPing**

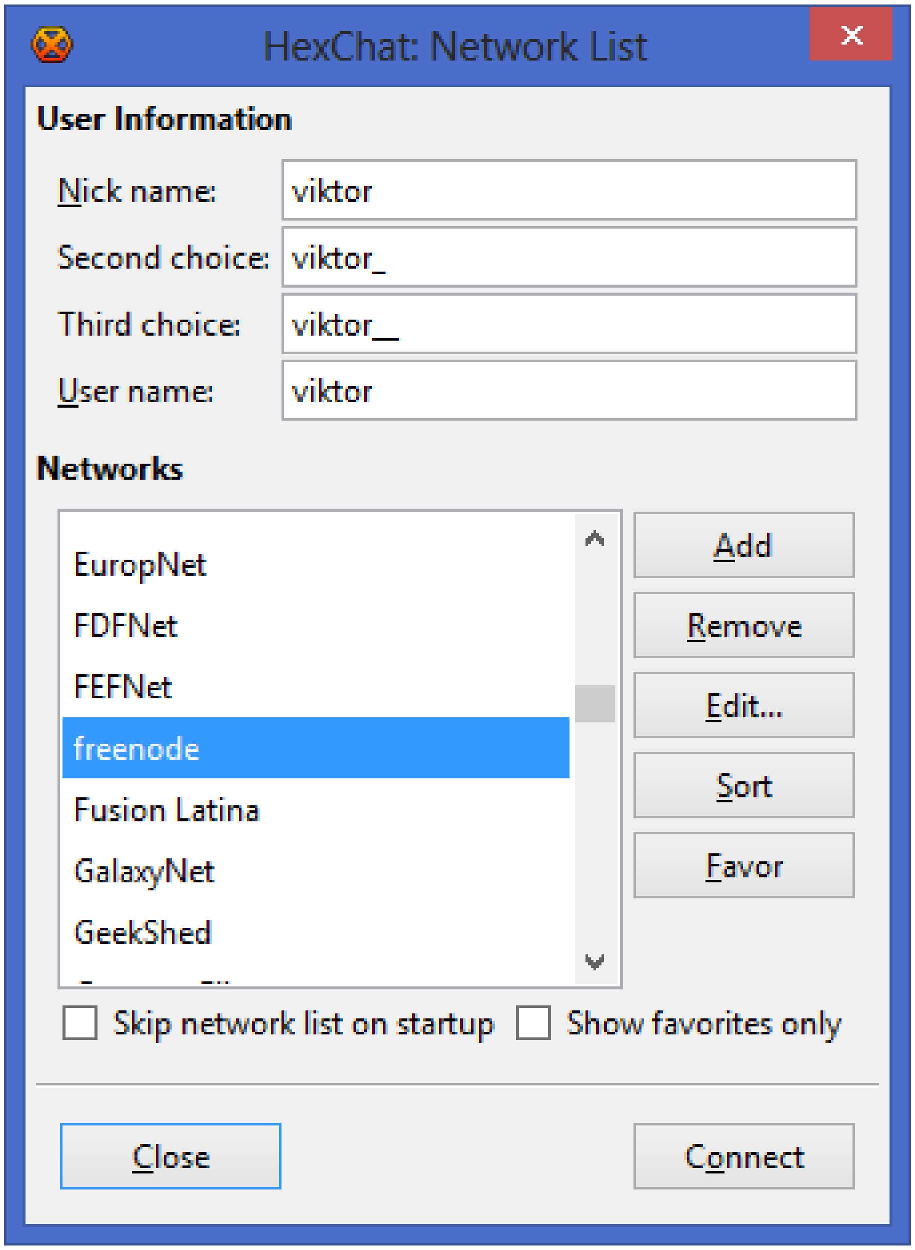
January 18, 2017

CHAPTER 1

# Getting Started

## 1.1 Quick Start

The first time you start HexChat you'll see the *Network List* as seen below:


HexChat: Network List

**User Information**

Nick name: viktor

Second choice: viktor_

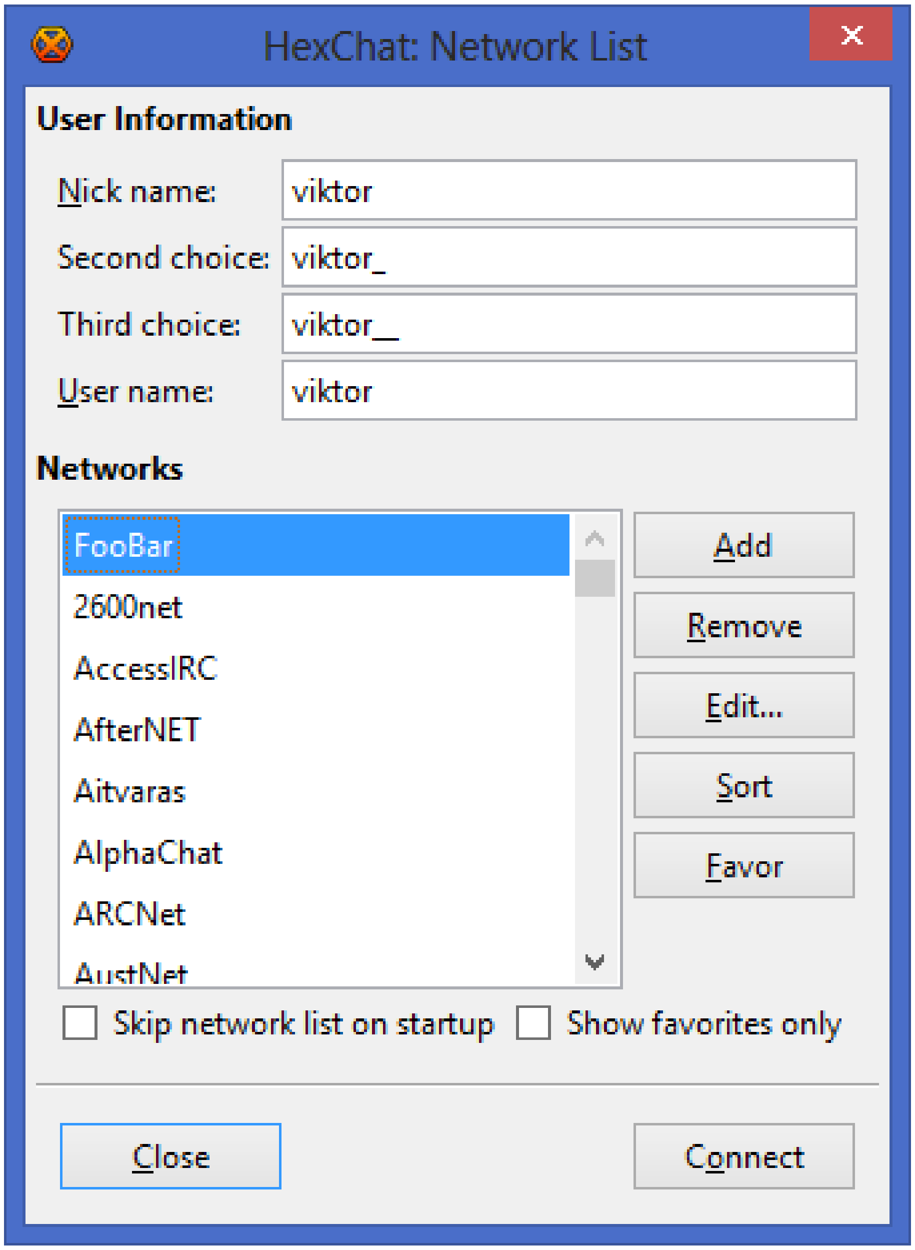
Third choice: viktor__

User name: viktor

**Networks**

EuropNet
FDFNet
FEFNet
freenode
Fusion Latina
GalaxyNet
GeekShed

Here you can specify your global logon details. The *Nick name* will be your name visible in IRC channels (or second/third choice if it's already taken), and the *User name* is how you identify yourself to the server. You can pick a server from the default list, or if it's not there, you can add your own by clicking the *Add* button:



Nick name: viktor

Second choice: viktor_

Third choice: viktor__

User name: viktor

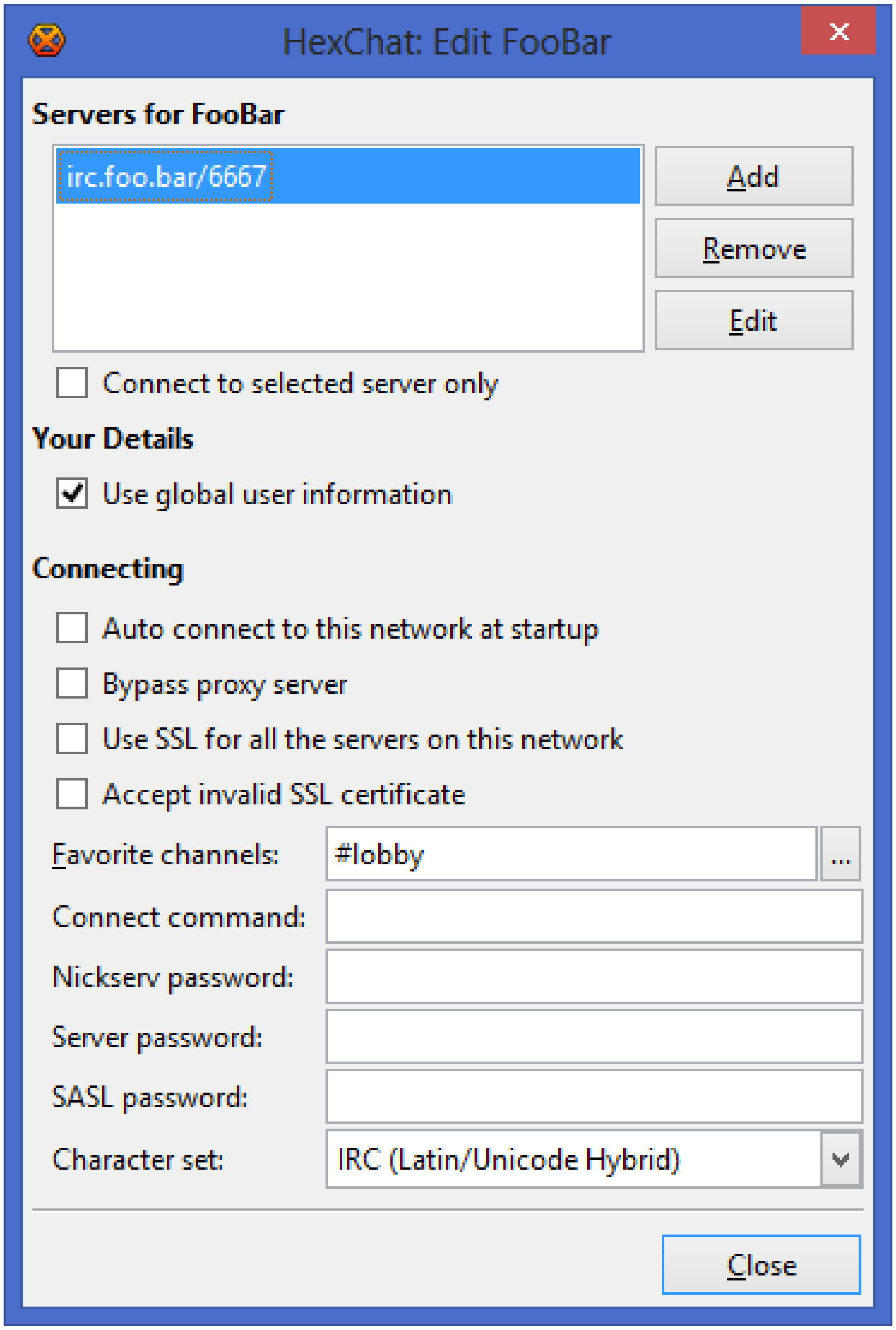
FooBar
2600net
AccessIRC
AfterNET
Aitvaras
AlphaChat
ARCNet
AustNet

☐ Skip network list on startup ☐ Show favorites only

After you named it, click *Edit...* and specify the details of your connection. You need at least the following information to join to a certain group of people:

- server address
- server port
- channel name

You can see an example below:



Servers for FooBar

irc.foo.bar/6667

- [ ] Connect to selected server only

Your Details

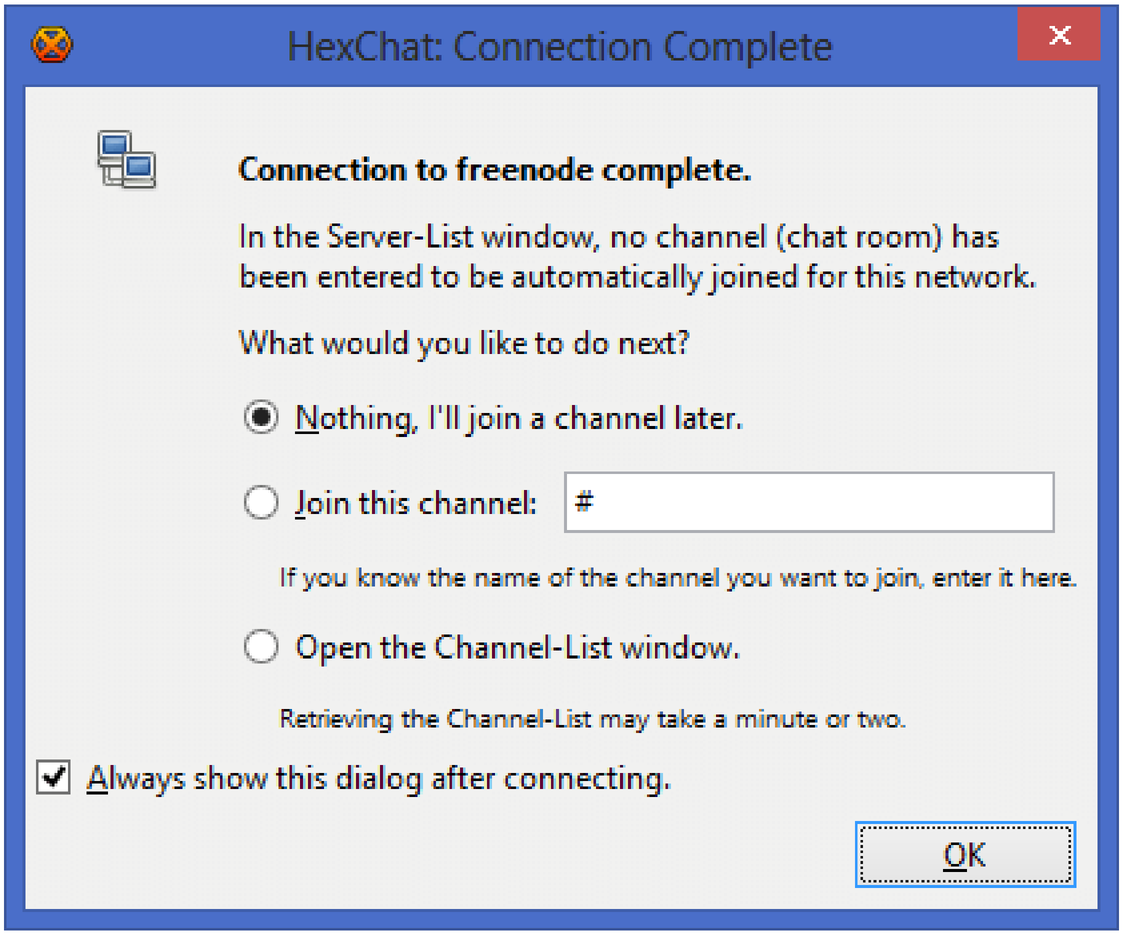
- [✓] Use global user information

Connecting

- [ ] Auto connect to this network at startup
- [ ] Bypass proxy server
- [ ] Use SSL for all the servers on this network
- [ ] Accept invalid SSL certificate

| | |
|---|---|
| Favorite channels: | #lobby |
| Connect command: | |
| Nickserv password: | |
| Server password: | |
| SASL password: | |
| Character set: | IRC (Latin/Unicode Hybrid) |

In this example, the server address is *irc.foo.bar* and the port is *6667*. The channel of choice is *#lobby*. Favorite channels are joined to automatically upon connecting to the network. After you finished editing, click *Close* to return to the *Network List*. Now select the network you want to connect to and click *Connect*. After a successful connection you'll see the following window:


HexChat: Connection Complete

**Connection to freenode complete.**

In the Server-List window, no channel (chat room) has been entered to be automatically joined for this network.

What would you like to do next?

[✓ Nothing, I'll join a channel later.]

Join this channel: #

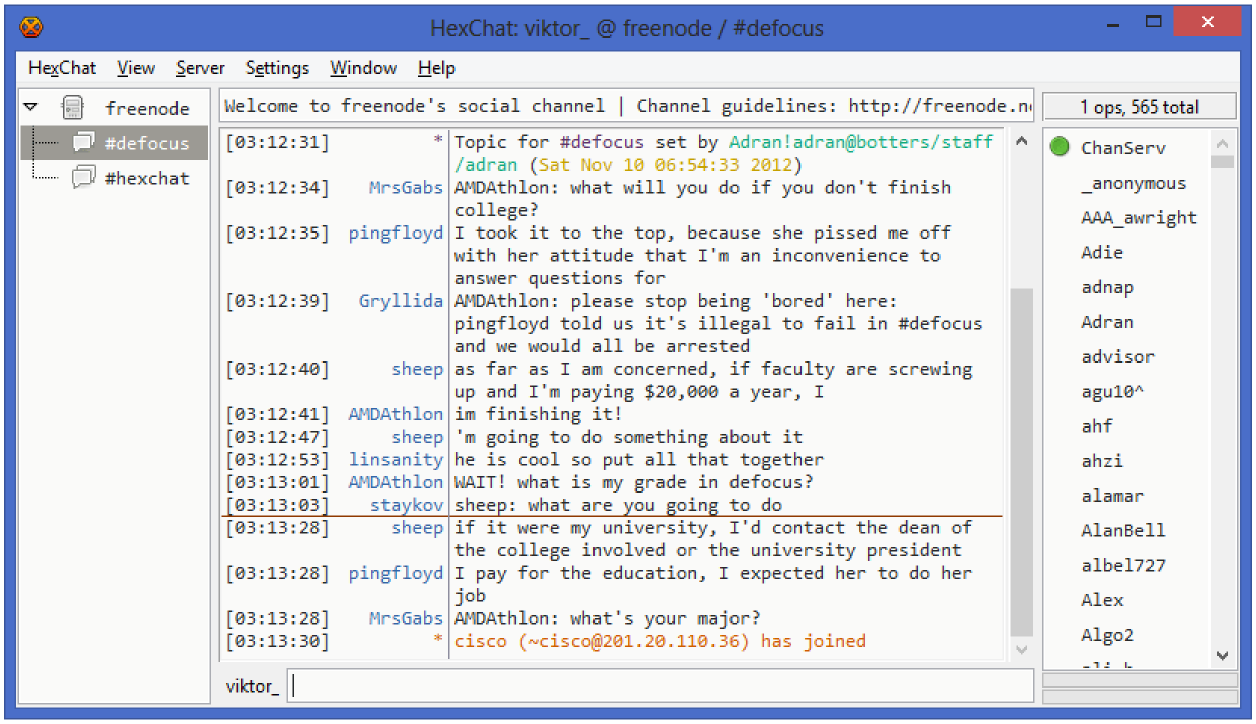
If you know the name of the channel you want to join, enter it here.

Open the Channel-List window.

Retrieving the Channel-List may take a minute or two.

[✓ Always show this dialog after connecting.]

If you don't want to join a channel just yet, leave it as it is. If you know the channel name already, specify it with the second option. In case you want to browse through the channel list, select the third option. Then press *OK* to close this window. After you successfully joined a channel, you should see something like this:


HexChat: viktor_ @ freenode / #defocus
HexChat View Server Settings Window Help
freenode
#defocus
#hexchat
1 ops, 565 total
viktor_

```
Welcome to freenode's social channel | Channel guidelines: http://freenode.n
[03:12:31]          * Topic for #defocus set by Adran!adran@botters/staff
                      /adran (Sat Nov 10 06:54:33 2012)
[03:12:34]   MrsGabs  AMDAthlon: what will you do if you don't finish
                      college?
[03:12:35] pingfloyd  I took it to the top, because she pissed me off
                      with her attitude that I'm an inconvenience to
                      answer questions for
[03:12:39]  Gryllida  AMDAthlon: please stop being 'bored' here:
                      pingfloyd told us it's illegal to fail in #defocus
                      and we would all be arrested
[03:12:40]     sheep  as far as I am concerned, if faculty are screwing
                      up and I'm paying $20,000 a year, I
[03:12:41] AMDAthlon  im finishing it!
[03:12:47]     sheep  'm going to do something about it
[03:12:53] linsanity  he is cool so put all that together
[03:13:01] AMDAthlon  WAIT! what is my grade in defocus?
[03:13:03]   staykov  sheep: what are you going to do
[03:13:28]     sheep  if it were my university, I'd contact the dean of
                      the college involved or the university president
[03:13:28] pingfloyd  I pay for the education, I expected her to do her
                      job
[03:13:28]   MrsGabs  AMDAthlon: what's your major?
[03:13:30]          * cisco (~cisco@201.20.110.36) has joined
```

```
ChanServ
_anonymous
AAA_awright
Adie
adnap
Adran
advisor
agu10^
ahf
ahzi
alamar
AlanBell
albel727
Alex
Algo2
```

That's it, you're online. Now you can learn more about HexChat and customize it for your needs. This website is a good starting point, but you can find a lot more on the net. Have fun!

## 1.2 Frequently Asked Questions

### 1.2.1 Frequently Asked Questions

#### How do I migrate my settings from XChat?

**Caution:** This is generally not recommended. Some issues that may arise are mentioned here: https://github.com/hexchat/hexchat/pull/1794#issuecomment-240102695

- Unix
  1. Copy `~/.xchat2` to `~/.config/hexchat`
  2. Rename `~/.config/hexchat/xchat.conf` to `~/.config/hexchat/hexchat.conf`
  3. Rename `~/.config/hexchat/servlist_.conf` to `~/.config/hexchat/servlist.conf`
  4. Rename `~/.config/hexchat/xchatlogs` to `~/.config/hexchat/logs`
  5. Move all your 3rd party addons (plugins/scripts) to `~/.config/hexchat/addons`
  6. Move all your client certs to `~/.config/hexchat/certs`
- Windows
  1. Copy `%APPDATA%\X-Chat 2` to `%APPDATA%\HexChat`
  2. Rename `%APPDATA%\HexChat\xchat.conf` to `%APPDATA%\HexChat\hexchat.conf`
  3. Rename `%APPDATA%\HexChat\servlist_.conf` to `%APPDATA%\HexChat\servlist.conf`

4. Rename `%APPDATA%\HexChat\xchatlogs` to `%APPDATA%\HexChat\logs`
5. Move all your 3rd party addons (plugins/scripts) to `%APPDATA%\HexChat\addons`
6. Move all your client certs to `%APPDATA%\HexChat\certs`

The server list format also changed, instead of a giant autojoin list formatted `J=chan1,chan2 key1,key2` it is now formatted on seperate lines `J=chan1,key1\nJ=chan2,key2`

### How do I autoconnect and join a channel on start?

In the Network List, select the network to which you would like to automatically connect to at startup, click the "Edit..." button, and then tick the checkbox for "Connect to this network automatically".

To automatically join channels for this network upon connecting, select the "Autojoin channels" tab in this same window and add your desired channels here. You can also add a currently joined channel by right-clicking its name in the channel switcher and by toggling "Autojoin" on.

### How do I auto-reconnect after my computer wakes up from being in sleep/hibernate mode?

Try the following command from a chat window (time is in seconds):

> /set net_ping_timeout 60

### Why are channels joined before identifying?

There are 3 ways to authenticate before joining a channel, all are network dependant but nickserv is common and SASL is the best:

1. Use a Nickserv password and increase the delay before joining in *Settings* → *Preferences* → *Advanced*
2. Use SASL which can be enabled in *HexChat* → *Network list* → *Edit* (2.9.4+). Your username must match your nickserv account, you can set it by unchecking *Use global user information*.
3. Use a client cert which requires the most setup.

### How do I change what browser is opened?

- Windows:

  *Control Panel* → *Default Programs*
- Unix:
  - Gnome 3: *System Settings* → *Details* → *Default Applications*
  - Other DE's have their own settings that may or may not work.

  If these do not work or you do not use a DE use the command **`gvfs-mime`** which may need to be installed:

```
gvfs-mime --set x-scheme-handler/http firefox.desktop
```

  Don't forget to do the same for *https* and run this as your user.

  Now upon launching it will use the *Exec* line in their desktop file replacing *%u* with the url. If you get a blank window this is where the problem is.

Alternatively you can add a Url Handler

### How do I connect through a proxy?

Go to the menus, *Settings* → *Preferences* → *Network Setup* and fill in the requested information there. Authentication (using a username and password) is only supported for HTTP and Socks5.

For information on Tor see our tips page

### How do I show @ and + in front of nicknames that are Op and Voice when they talk?

To display @ and + characters next to nicknames as they talk, do the following:

In the menus, open up *Settings* → *Text Events*. Find the *Channel Message* event in the list. The $3 code can be inserted to print the user's mode-character (e.g. @ or +). For example, you might want to change the default:

> %C18%H<%H$4$1%H>%H%O$t$2

to

> %C18%H<%H$4$3$1%H>%H%O$t$2

Don't forget to **press Enter**, so the changes take effect in the list at the top of the window.

### How do I set different ban types?

1. Right click the nickname in the userlist, and choose a ban type from the "Kick/Ban" submenu.
2. You can also do it manually: > /ban nick bantype where the bantype is a number from 0 to 3.
3. Or set the default with:

   /set irc_ban_type bantype sets the default ban type to use for all bans. The different types are:

   - 0 = *!@.host
   - 1 = *!*@domain
   - 2 = *!*user*@.host*
   - 3 = *!*user@domain

### Why does the timestamp overlap some nicknames?

Some networks allow very long nicknames (up to 32 letters). It can be annoying to have the separator bar move too far to the right, just for one long nick. Therefore, it has a set limit for the distance it will move to the right. If you use a large font, you may need to adjust this distance. It is set in pixels, for example:

> /set text_max_indent 320

Once you adjust this setting high enough, overlapping timestamps and nicknames should not occur. The adjustment will not take effect immediately, a restart may be needed.

### How do I hide join and part messages?

To disable joins and parts from being displayed in all channels check 'Hide join and part messages' under *Settings -> Preferences -> Chatting -> General* (Advanced pre-2.9.6)'.

Then all channels you join **after** setting this will start with "Show join/part messages" turned off.

To disable the prints for only certain channels, right click on the channel tab and click *Settings -> Hide Join/Part Messages*

### Why doesn't DCC send work behind a router?

If you are behind a IP-NAT or ADSL router, you will most likely have an address like 192.168.0.1. This address is not usable on the Internet, and must be translated.

When offering a DCC file, HexChat will tell the receiver your address. If it says 192.168.0.1, the receiver will not be able to connect. One way to make it send your "real" address is to enable the "Get my IP from IRC Server" option in HexChat. This option is available in *Settings → Preferences → Network setup*. When you turn it ON, you will have to re-login to the server before it'll take effect.

You will also need to forward some ports for use in DCC send. You may pick almost any port range you wish, for example, in HexChat set:

First DCC send port: 4990 Last DCC send port: 5000

This will allow you to send up to ten files at the same time, which should be plenty for most people. Lastly, configure your router/modem to forward ports 4990-5000 to your PC's address. You'll have to consult your router/modem's manual on how to do this.

### How do I execute multiple commands in one line?

There are a few ways to do this:

- If this is during connection the network list (Ctrl+s) has a 'connect commands' section as well as most login types you would need.
- /LOAD -e <textfile>, where <textfile> is a file in your config dir containing commands on each line.
- Separate your commands with CTRL-SHIFT-u-a. This will appear as a little box with numbers on it (or an invisible character).
- You can create two UserCommands, with the same name, and then execute the UserCommand. It will be executed in the same order as it's written in the UserCommands GUI.

### I get this error: "Unknown file type abc.yz. Maybe you need to install the Perl or Python plugin?"

If you get this error when trying to load a Perl or Python script, it means the plugin or language for running those scripts isn't loaded.

On Unix: - Install the packages including these plugins (i.e. hexchat-perl) and perl. - When building yourself check ./configure output and install devel packages.

On Windows - Select the plugins in the installer. - Install the language linked on our downloads page.

### How do I play sound files on certain events?

In the menus, go to: *Settings → Preferences → Sound*. Select the event you want to make a sound on, then type in a sound filename (or use the Browse button). On Windows, sound files must be in .wav format.

### How do I auto-load scripts at startup?

The root of your HexChat config is:

- Windows: %APPDATA%\HexChat
- Unix/Linux: ~/.config/hexchat

Referred to as <config> from now. HexChat automatically loads, at startup:

- <config>/addons/*.pl Perl scripts
- <config>/addons/*.py Python scripts
- <config>/addons/*.lua Lua scripts
- <config>/addons/*.dll Plugins (Windows)
- <config>/addons/*.so Plugins (Unix)

### How do I minimize HexChat to the System Tray (Notification Area)?

On both Unix and Windows there is an included tray plugin. To enable minimizing to tray on exit go to *Settings* → *Preferences* → *Alerts*.

### How do I start HexChat with...?

To see the various launch options such as setting configdir or minimize level run:

> hexchat –help

### Where are the log files saved to?

- Unix

  > ~/.config/hexchat/logs

- Windows

  > %APPDATA%\HexChat\logs

### Why do config files and logs appear as a single line?

HexChat supports many platforms and to be consistent it uses Unix line endings.

This is normally not a problem at all but Microsoft Notepad does not support these so they are rendered as a single line. Any other editor should support them.

### How do I rotate log files every so often?

By default settings, no rotation occurs, your log files will just keep getting larger.

Go to *Settings* → *Preferences* → *Logging* and change the log filename to any one of these:

> %Y-%m-%d/%n-%c.log -> 2006-12-30/FreeNode-#channel.log
>
> %n/%Y-%m-%d/%c.log -> FreeNode/2006-12-30/#channel.log
>
> %n/%c.log -> FreeNode/#channel.log (no rotation)

%Y, %m and %d represents the current year, month and day respectively. %n is the network name, e.g. “FreeNode” or “UnderNet”, and finally, %c is the channel. In these examples, a new log filename and folder would be created after midnight.

The format can also be a full path if you want to save logs to an external drive for example.

For the full list of formatting codes, please refer to the Unix or Windows documentation on *strftime*.

### Where did the Real Name field go?

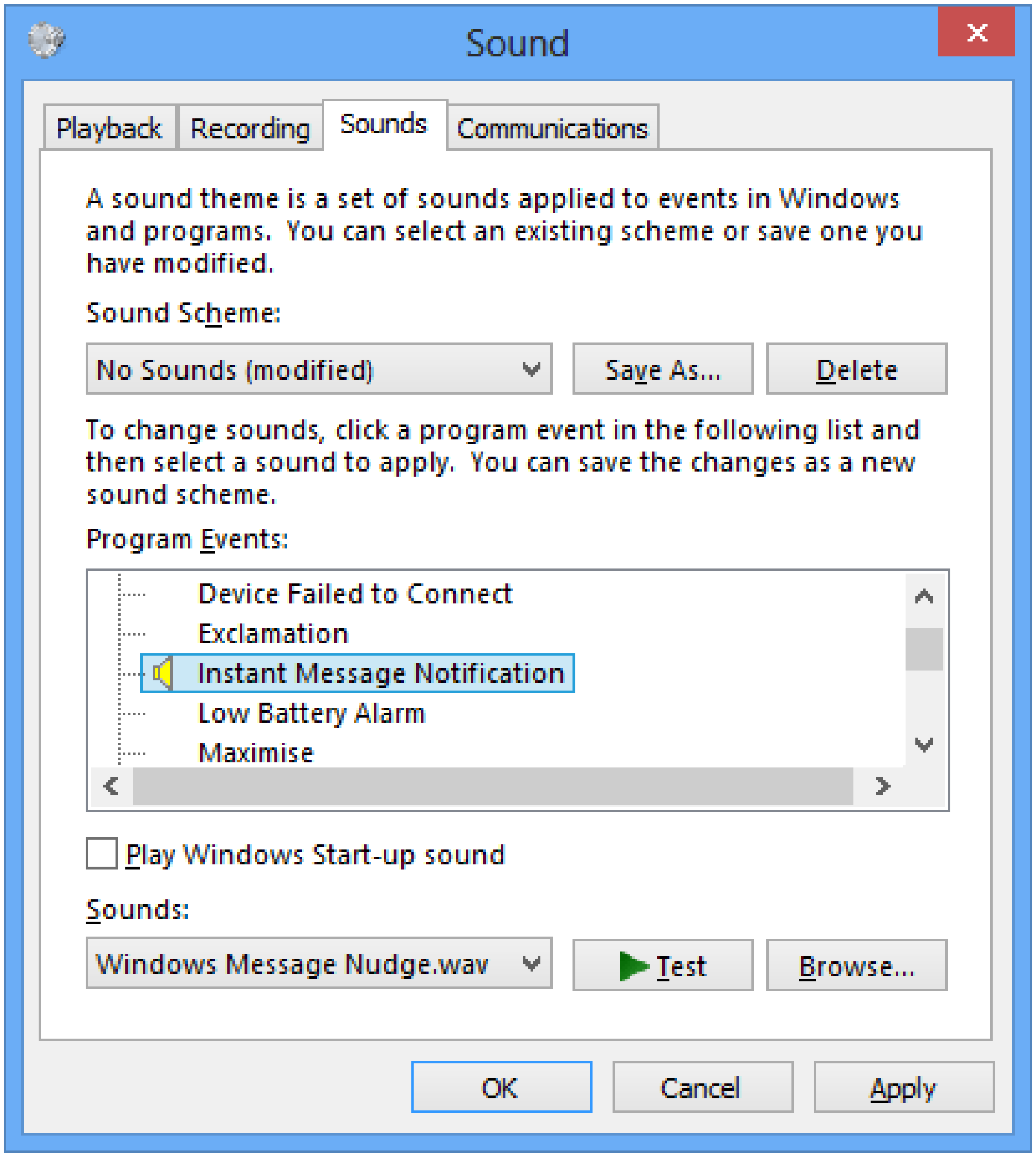
The Real name field used to be accessible via the Network List, which is the very first screen that a new user sees. Newcomers, who are not familiar with IRC terminology, might be afraid of their personal data. In order to avoid alienating such people, we decided to remove this setting from the Network List. Now you can access this setting under *Settings → Preferences → Advanced* instead, or if you prefer the command line, you can use the following command:

> /set irc_real_name Stewie Griffin

### Why don't beep sound alerts work?

On Windows, HexChat is using the *Instant Message Notification* system sound for making beep alerts, and if it's unspecified, it attempts to produce a simple beep effect. In case you don't hear beeps when alerts occur, you need to set this system sound to the desired sound effect. To do this, go to *Control Panel → Hardware and Sound → Change system sounds*.

On Unix it uses libcanberra by default to play the "message-new-instant" freedesktop sound. Use the *canberra-gtk-play* command to test playing events or custom sound files as that is likely where the problem lies.



A sound theme is a set of sounds applied to events in Windows and programs. You can select an existing scheme or save one you have modified.

Sound Scheme:

No Sounds (modified)

To change sounds, click a program event in the following list and then select a sound to apply. You can save the changes as a new sound scheme.

Program Events:

- Device Failed to Connect
- Exclamation
- Instant Message Notification
- Low Battery Alarm
- Maximise

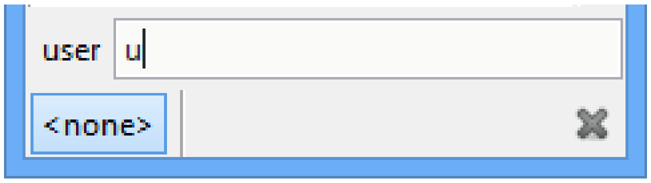
☐ Play Windows Start-up sound

Sounds:

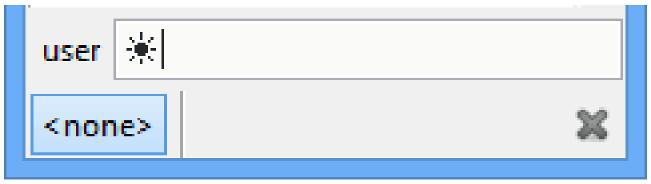
Windows Message Nudge.wav

### How do I type Unicode characters?

Press *Ctrl* + *Shift* + *U* at once. When you release the keys, *u* will appear in your input box.


user u

Now you can enter the 4-digit code of the desired glyph. When you're done, just press *Space* or *Return*, and the glyph will appear as well.



**Note:** Windows alt codes will not work.

#### How do I add a network to the official list?

There are no official requirements to be added to the network list, but there are some simple guidelines of what the network SHOULD have.

- A maintained website listing servers.
- Has been around for a while.
- A decent amount of active users (100+).
- Doesn't block large groups of users.

You should make a pull request on GitHub if you want the quickest results, the list is stored in *src/common/servlist.c*. You should include an SSL server if supported but you should not include IPv6 only servers and mark SASL as default if supported. Follow the syntax of other networks in the file.

## 1.3 Changelog

### 1.3.1 HexChat ChangeLog

#### 2.12.4 (2016-12-10)

- fix issue with timers causing ping timeouts
- fix building against OpenSSL 1.1
- fix /exec output printing invalid utf8
- replace doat plugin with an internal command
- change how tab colors interact with plugins

#### 2.12.3 (2016-10-22)

- fix crash with bad translations
- fix crash and leaks in mpcinfo plugin
- add mhop command
- change ping timeout to 60 by default
- update translations

### 2.12.2 (2016-10-08)

- fix input box theme with Adwaita 3.20
- fix return value of hexchat_pluginpref_get_int()
- fix tab color changing when print events are eaten
- fix network name not being sanitized for scrollback files
- fix building sysinfo on OS X <= 10.9
- fix resume with DCC GET
- fix possible assertion when decoding incoming text
- fix possible crashes when plugins modify the UI during context close
- add "chanmodes" to channel list in plugin api
- lua:
  - add automatic return and = handling in console
  - fix pluginpref usage
- fishlim:
  - fix saving nicks containing [ or ]
  - add commands: /topic+, /msg+, and /notice+
  - add support for /me
  - add /keyx command to do DH1080 key exchanges
- improve efficiency of various timers
- reduce updates of user count in titlebar/userlist
- download extra redist for perl on Windows
- update appdata file
- update translations
- update dependencies on Windows

### 2.12.1-2 (2016-05-06)

- fix crash in dependencies on Windows (no HexChat changes)

### 2.12.1 (2016-05-01)

- add lua plugin
- change desktop file to open urls in existing instance on Unix
- misc chanopt fixes
  - fix not being saved properly
  - fix values other than 0/1/2 being set
  - fix flags in plugin API

- misc identd fixes
  - fix ipv6 support on Windows
  - fix respecting USE_GLOBAL network flag
  - fix connections not always closed
  - add responses for INVALID-PORT and NO-USER
- misc challengeauth fixes
  - fix lower casing username
  - don't respond to challenge notices when disabled
- re-add support for old versions of libnotify
- bundle luajit and lgi on Windows
- update network list

### 2.12.0 (2016-03-12)

- add support for native Windows 8+ spell check
- add support for native Windows 8.1+ and OSX notifications
- add support for IRCv3.2
  - add cap 3.2
  - add sasl 3.2
  - add chghost
  - add cap-notify
- add support for twitch.tv/membership cap
- add support for SNI (Server Name Indication)
- add ability to do DnD reordering in some settings dialogs
- add option to disable middle-click closing tabs
- rewrite sysinfo plugin
  - add support for OSX
  - add multi-gpu info on win32
  - add hdd info on win32
  - add ability to print single peice of info
  - change formatting, unified across platforms
  - fix handling unsupported CPU arch on unix
- rewrite identd plugin
  - add support for unix
  - add ability to change port
  - fix handling multiple connections
  - fix threading issues

- rewrite update plugin
  - fix locking up UI during check
  - show graphical dialog on update
  - host update file on https://dl.hexchat.net
- rewrite checksum plugin
  - fix support for very large files
  - remove openssl dependency
- remove DH-{AES,BLOWFISH} mechanisms (insecure)
- remove “IRC” encoding, replaced with UTF-8
- remove “System Default” encoding, replaced with UTF-8
- remove configure option to disable ipv6
- remove msproxy and socks5 library support (unused)
- change tab-complete to favor other user nicks over own
- change url detection to support unicode
- change decoding to not attempt ISO-8859-1 fixing corruption
- change pluginpref to escape values
- change installer on Windows to not overwrite gtkrc file
- minor changes to icons
- fix numerous crashes (but not #600)
  - fix crash on printing very long lines
  - fix crash on failing to open log file
  - fix crash when using unsorted userlist
  - fix crash when timestamp format set to nothing
  - fix crash when tab-completeing long strings
  - fix crash with long values in pluginpref API
  - fix various other unsafe string handling throughout
- fix poor performance with nick indent enabled
- fix UTF-8 text in winamp plugin
- fix fishlim plugin handling networks with server-time
- fix logging hostname of users in new queries
- fix Key Press event sending non-UTF-8 text to plugins
- fix VERSION response on windows 10
- fix get_info(‘win_ptr’) from python
- fix running portable-mode from another directory
- fix duplicate timestamps on selection
- fix –cfgdir argument

- fix mode-button text being cut off
- fix scrollback timestamps with server-time
- fix url handler accepting quoted paths with spaces
- fix using correct encoding when jumping networks
- improve DCC handling large files
- improve python detection in configure
- improve scrollback file handling (corruption, line endings)
- improve building in cygwin
- improve build options on unix to be more secure
- update to VS 2015 on windows
- update deps on windows (openssl, python 3.5, etc)
- update translations
- update network list

### 2.10.2 (2014-11-25)

- verify hostnames of certificates
- use more secure openssl options (No SSLv2/3)
- detect utf8 urls in chat
- fix using multiple client certs at the same time
- fix checking for Perl on some distros
- fix friends list not properly updating
- fix building with format-security
- fix more non-ascii path issues on Windows
- fix opening utf8 urls on Windows and OSX
- update deps on Windows
- update translations

### 2.10.1 (2014-07-28)

- add configure check for python 3.4
- improve spell check handling apostrophes
- disable unsupported tray feature on Elementary OS
- disable installing on Vista (would just crash on x64)
- fix possible crash in banlist
- fix some crashes in pluginpref API
- fix crash in spell check
- fix notify list being sent to wrong networks

- fix scrolling the chanview on Unix
- fix building plugin support on some platforms (OS X 10.8)
- fix FiSHLiM decryption when identify-msg is enabled
- fix expanding *~user* on Unix
- fix missing license files on Windows
- fix silent installer on Windows
- update translations
- update network list
- update Python versions on Windows

### 2.10.0 (2014-06-01)

- fix SASL on InspIRCd networks
- fix building on OpenBSD
- fix crash when using invalid timestamps on Windows
- fix Lag Meter reporting invalid numbers
- fix drag and drop on User List/Channel Switcher
- fix various Unicode issues on Windows
- add fullscreen display mode
- add `/getbool` command
- add support for userhost-in-names capability
- add –command command line flag on Windows
- add message parameter to `/query`
- add help messages to user commands
- add `plugin_pref` to Perl
- add regex search to the Channel List on Windows
- add option to hide nick change messages
- redesign various settings windows (notably key bindings)
- render colors and attributes in the Input Box and Topic Bar
- bind middle-click to close tabs
- build the GTK+ PixBuf theme engine on Windows
- change `/list` to open the Channel List window
- change the format key bindings are stored in (will convert automatically)
- enable italics
- rewrite `/dns` adding support for Windows and replacing the plugin
- rebrand the Perl plugin
- replace the Search window with a Search Bar

- replace Text Box transparency with full window transparency (may come back)
- remove all deprecated GTK+ usage
- remove all direct use of X11, fixing XWayland support
- hide unsupported channel modes from the Topic Bar
- improve spell check (handle contractions)
- improve rendering and selection in the Text Box
- improve OS X integration (custom theme, app menu, dock icon, spell check, app bundle, key bindings)
- improve the Windows installer (downloads requirements when ran)
- update Visual Studio to 2013
- update Perl to 5.20 on Windows
- update Python to 3.4 on Windows
- update OpenSSL on Windows to fix “Heartbleed”
- update translations

### 2.9.6.1 (2013-09-15)

- fix some utilities causing crash #740

### 2.9.6 (2013-09-11)

- redesign edit window in network list
- rename favorites to autojoin
- improve URL detection yet again (this time with more IPv6!)
- implement `/exec -o` on Windows
- improvements to the DCC window
- improvements to sysinfo on Unix, including `-e` to print info instead of saying
- add support for BLOWFISH, AES, and EXTERNAL SASL mechanisms
- add reload command and button in plugingui
- add support for server-time and znc.in/server-time[-iso] capabilities
- add attributes to `hook_print`/`server` and `emit_print` for information such as server-time
- add support for QuakeNet’s challangeauth
- add chanopt for stripping colors
- add copy option to banlist entries
- add autoconnect option to context menu of networks
- add option for omitting alerts while window is focused
- add python3 support along with various bugfixes
- add libcanberra support on Unix
- add tracking of users accounts

- add `%u` to userlist popups for accounts
- add channelkey to channel lists in plugin api
- add `MONITOR` support for the friends list
- add `QUIET` and `UNQUIET` commands
- add support for the away-notify, account-notify, and extended-join capabilities
- add notifications for friends away status (requires away-notify)
- add events for quiet, unquiet, and quietlist
- add Ctrl+N (New Server Window) keybinding
- add Ctrl+Home/End keybinding for scrolling to top/bottom
- add theme manager to Unix build system
- fix compilation on FreeBSD
- fix running as root
- fix splitting ctcps and notices
- fix alerts and scrollback chanopts
- fix crash when attaching/detaching tabs
- fix sending limited channel messages (op messages) to the wrong tab
- change `/load -e` to load from config dir
- remove Ctrl+L (Clear Text) keybinding
- remove custom sound applications
- remove away announce, replaced by away-notify on supported servers (alternative python script)
- update network list

## 2.9.5 (2013-04-01)

- fix Checksum plugin with DCC download directory set
- fix false positives with Update Checker
- fix sound directory option on Unix
- fix loading custom icons
- fix tray icon not reappearing if the tray crashes
- fix restoring maximized windows from tray
- fix `/QUERY -nofocus`
- fix reconnecting to channels with keys
- fix compilation on FreeBSD
- fix showing the join dialog when autojoining channels
- fix Plugin-Tray menu not closing on Windows
- fix close dialog minimizing to tray before selection
- fix Python plugin compilation on Ubuntu 13.04

- fix Theme Manager crashing with read-only files
- fix channel tree indentation without server tab or with icons
- add auto-away support to Plugin-Tray
- add Plugin-Tray option to disable blinking
- add option to always show notices on current tab
- add support for notification filtering in GNOME 3.8
- add support for channel keys in URLs
- add option to color nicks in the user list the same way as in the chat area
- add ability to automatically switch to last activity on change-page hotkey
- add ability to save divider position between combined user list and channel tree
- add global real name option to Preferences
- add Safe Mode shortcut to the Start Menu group on Windows
- add helpful links to the setup wizard on Windows
- make the source tree compliant with Debian policies
- install SVG icon on Unix
- enable Plugin-Tray menu on Windows
- enable IPv6 by default on Unix
- show `/WHOIS` response on current tab by default
- redesign the Ban List window to show invites, bans, exemptions and quiets
- make user list icons slightly smaller
- close all utility windows with the Esc key
- improve URL and username detection in the chat area
- make `/JOIN` focus the existing channel if already joined
- change default DCC download directory to ~/Downloads on Unix
- allow Plugins and Scripts utility to be opened in a tab
- only beep when the HexChat window is not active
- use the certs subfolder of the config folder for loading custom certificates
- disable tray icon when using Unity
- remove Lua and Tcl
- remove HexTray in favor of built-in Plugin-Tray
- remove installer theming on Windows
- cease support for Perl 5.12 and 5.14 on Windows
- rebuild every dependency with Visual C++ on Windows
- stop using the WDK on Windows and depend on the Visual C++ Redistributable
- update GTK+ to 2.24 on Windows
- update default text events

- update translations
- update the network list

## 2.9.4 (2012-11-11)

- fix alerts when omit alerts in away option is set
- fix dialog icon in userlist popup
- fix opening links on Mac
- fix default network in the Network List
- fix initial folder in file dialogs
- fix positioning the nick change dialog
- fix error message for busy servers
- fix filename encoding errors
- fix Fedora spec file
- fix Raw Log content being impossible to copy when auto-copy is disabled
- fix rough icon rendering in most windows on Windows
- fix config folder when specified with -d argument
- add built-in support for SASL authentication via CAP
- add support for identify-msg/multi-prefix server capabilities
- add text events for CAP related messages
- add support for the SysInfo plugin on Unix
- add option to change update check frequency and delay for first check
- add option to change GUI language on Windows
- add Ignore entry to userlist popup
- add Afrikaans, Asturian, Danish, Gujarati, Indonesian, Kinyarwanda and Malayalam translations
- add ChangeLog and ReadMe links to Start Menu during installation on Windows
- add manual page on Unix
- add icon support for 3 levels above op user mode
- change default colors, text events and user list/channel tree icons
- make Esc key close the Raw Log window
- use Consolas as the default font where available
- open dialog window for double-clicking in the user list by default
- variable separation, cleanup and renaming
- check in the installers whether Windows release is supported by HexChat
- display previous value after `/SET`
- reorganize the Settings menu and add new options
- redesign the About dialog

- show certain help messages in GTK+ dialogs instead of command line
- disable faulty one instance option
- build system cosmetics on Unix
- reorganize repo file structure
- rebranding
- update translations
- update the network list

### 2.9.3 (2012-10-14)

- fix various URL detection bugs
- fix default folders for file transfers in portable mode
- fix Autotools warnings with recent releases
- add `/ADDSERVER` command
- add option to save URLs to disk on-the-fly
- add option to omit alerts when marked as being away
- add default icons for channel tree and option to turn them off
- change certain default colors
- enhance Non-BMP filtering performance
- accept license agreement by default on Windows
- update the network list

### 2.9.2 (2012-10-05)

- fix compilation on Red Hat and Fedora
- fix portable to non-portable migrations on Windows
- fix ban message in HexTray
- fix icon in Connection Complete dialog
- fix determining if the log folder path is full or relative
- fix desktop notification icons on Unix
- fix URL grabber saving an unlimited number of URLs by default
- fix URL grabber memory leaks under certain circumstances
- fix URL grabber trying to export URL lists to system folders by default
- fix opening URLs without `http(s)://`
- add support for regenerating text events during compilation on Windows
- add support for the theme manager on Unix
- add Unifont to the default list of alternative fonts
- add option to retain colors in the topic

- allow the installer to preserve custom GTK+ theme settings on Windows
- use the icons subfolder of the config folder for loading custom icons
- use port 6697 for SSL connections by default
- install the SASL plugin by default on Windows
- `/lastlog` improvements
- build system cosmetics on Unix
- open links with just left click by default
- enable timestamps and include seconds by default
- make libproxy an optional dependency on Unix
- update German translation
- update the network list

### 2.9.1 (2012-07-27)

- fix installing/loading plugins on Unix
- fix restoring the HexChat window via shortcuts on Windows
- fix HexTray icon rendering for certain events
- fix the Show marker line option in Preferences
- fix `/lastlog` regexp support on Windows
- add support for the Checksum, Do At, FiSHLiM and SASL plugins on Unix
- add option to retain colors when displaying scrollback
- add MS Gothic to the default list of alternative fonts
- rebranding and cleanup
- eliminate lots of compiler warnings
- Unix build system fixes and cosmetics
- make Git ignore Unix-specific intermediate files
- use better compression for Windows installers
- switch to GTK+ file dialogs on Windows
- restructure the Preferences window
- use the addons subfolder of the config folder for auto-loading plugins/scripts
- improve the dialog used for opening plugins/scripts
- remember user limits in channel list between sessions
- remember last search pattern during sessions
- update XChat to r1521

### 2.9.0 (2012-07-14)

- rebranding
- migrate code to GitHub
- update XChat to r1515
- fix x64 Perl interface installation for Perl 5.16
- improve URL detection with new TLDs and file extensions

### 1508-3 (2012-06-17)

- add XChat Theme Manager
- fix problems with Turkish locale

### 1508-2 (2012-06-15)

- add support for Perl 5.16
- update Do At plugin
- fix drawing of chat area bottom
- avoid false hits when restoring from tray via shortcut
- migrate from NMAKE to Visual Studio

### 1508 (2012-06-02)

- remove Real Name from Network List
- search window improvements
- restore XChat-WDK from tray via shortcut if X-Tray is used

### 1507 (2012-05-13)

- update OpenSSL to 1.0.1c
- FiSHLiM updates

### 1506 (2012-05-04)

- update OpenSSL to 1.0.1b
- update German translation

#### 1503 (2012-03-16)

- update OpenSSL to 1.0.1
- URL grabber updates
- FiSHLiM updates

#### 1500 (2012-02-16)

- add option for specifying alternative fonts
- fix crash due to invalid timestamp format
- X-Tray cosmetics

#### 1499-7 (2012-02-08)

- fix update notifications
- fix compilation on Linux
- add IPv6 support to built-in identd

#### 1499-6 (2012-01-20)

- add DNS plugin

#### 1499-5 (2012-01-20)

- built-in fix for client crashes
- update OpenSSL to 1.0.0g

#### 1499-4 (2012-01-18)

- add Non-BMP plugin to avoid client crashes

#### 1499-3 (2012-01-15)

- rework and extend plugin config API
- add `ADD`/`DEL`/`LIST` support to X-SASL

#### 1499-2 (2012-01-11)

- add X-SASL plugin

#### 1499 (2012-01-09)

- fix saving FiSHLiM keys
- update OpenSSL to 1.0.0f

#### 1498-4 (2011-12-05)

- fix updates not overwriting old files
- display WinSys output in one line for others
- use Strawberry Perl for building

#### 1498-3 (2011-12-02)

- add plugin config API
- add Exec plugin
- add WinSys plugin
- perform periodic update checks automatically

#### 1498-2 (2011-11-25)

- add FiSHLiM plugin
- add option to allow only one instance of XChat to run

#### 1498 (2011-11-23)

- separate x86 and x64 installers (uninstall any previous version!)
- downgrade GTK+ to 2.16
- re-enable the transparent background option
- various X-Tray improvements
- add WMPA plugin
- add Do At plugin
- automatically save set variables to disk by default
- update OpenSSL to 1.0.0e

#### 1496-6 (2011-08-09)

- add option to auto-open new tab upon `/MSG`
- fix the update checker to use the git repo
- disable update checker cache

### 1496-5 (2011-08-07)

- fix attach/detach keyboard shortcut
- add multi-language support to the spell checker

### 1496-4 (2011-07-27)

- recognize Windows 8 when displaying OS info
- update OpenSSL certificate list
- fix X-Tray blinking on unselected events
- fix X-Tray keyboard shortcut handling
- cease support for Perl 5.10
- use Strawberry Perl for 5.12 DLLs

### 1496-3 (2011-06-16)

- add option for changing spell checker color

### 1496-2 (2011-06-05)

- add support for custom license text

### 1496 (2011-05-30)

- display build type in CTPC VERSION reply
- add support for Perl 5.14

### 1494 (2011-04-16)

- update Visual Studio to 2010 SP1
- update OpenSSL to 1.0.0d
- ship MySpell dictionaries in a separate installer

### 1489 (2011-01-26)

- fix unloading the Winamp plugin
- enable the Favorite Networks feature
- add Channel Message event support to X-Tray
- add mpcInfo plugin

#### 1486 (2011-01-16)

- fix a possible memory leak in the update checker
- fix XChat-Text shortcut creation
- fix XChat version check via the plugin interface
- add option for limiting the size of files to be checksummed
- add X-Tray as an install option
- disable Plugin-Tray context menu completely

#### 1479-2 (2011-01-10)

- improve command-line argument support
- add auto-copy options
- enable XChat-Text
- disable faulty tray menu items

#### 1479 (2010-12-29)

- update GTK+ to 2.22.1
- update OpenSSL to 1.0.0c
- update Python to 2.7.1
- replace X-Tray with Plugin-Tray

#### 1469-3 (2010-10-20)

- add Checksum plugin
- menu integration for Update Checker and Winamp

#### 1469-2 (2010-10-09)

- fix DCC file sending
- native open/save dialogs
- make the version info nicer
- register XChat-WDK as IRC protocol handler
- add option to run XChat-WDK after installation
- disable erroneous uninstall warnings
- disable Plugin-Tray, provide X-Tray only
- cease support for Perl 5.8
- replace EasyWinampControl with Winamp

### 1469 (2010-10-08)

- use Visual C++ 2010 for all WDK builds
- build Enchant with WDK and update it to 1.6.0
- fix SSL validation
- fix opening the config folder from GUI in portable mode
- further improve dialog placement for closing network tabs

### 1468-2 (2010-10-02)

- update GTK+ to 2.22
- spelling support
- more config compatibility with official build
- improve dialog placement for closing network tabs
- remove themes from the installer
- disable toggle for favorite networks until it's usable
- disable transparent backgrounds
- hide mnemonic underlines until Alt key pressed
- fix XP lagometer and throttlemeter rendering

### 1468 (2010-09-19)

- update Perl to 5.12.2
- update Tcl to 8.5.9
- fix scrollback shrinking
- enable advanced settings pane
- retain emoticon settings
- add `/IGNALL` command

### 1464-6 (2010-09-06)

- fix Perl interface breakage
- update checker plugin

### 1464-5 (2010-08-30)

- primitive update checker

### 1464-4 (2010-08-30)

- selectable tray icon
- selectable theme for portable
- selectable plugins

### 1464-3 (2010-08-29)

- black theme for portable

### 1464-2 (2010-08-29)

- make Perl version selectable during install

### 1464 (2010-08-26)

- Perl interface updates

### 1462 (2010-08-25)

- update XChat to r1462
- build system cleanup

### 1459-3 (2010-08-23)

- more installer changes (uninstall any previous version!)

### 1459-2 (2010-08-23)

- universal installer
- update build dependencies

### 1459 (2010-08-19)

- portable mode and installer fixes

### 1457 (2010-08-17)

- disable GUI warnings

### 1455-2 (2010-08-17)

- unified installer for standard and portable

#### 1455 (2010-08-15)

- support for gtkwin_ptr in the Perl interface

#### 1454 (2010-08-14)

- gtkwin_ptr for plugins introduced

#### 1452 (2010-08-14)

- fix taskbar alerts on x86
- upgrade Perl to 5.12 and make 5.8/5.10 builds available separately

#### 1451-6 (2010-08-12)

- include Lua-WDK with the installer

#### 1451-5 (2010-08-12)

- switch to Inno Setup (uninstall any previous version!)
- add Lua support

#### 1451-4 (2010-08-11)

- enable the XDCC plugin

#### 1451-3 (2010-08-11)

- enable Python support

#### 1451-2 (2010-08-11)

- enable SSL support
- fix simultaneous connections
- re-enable identd by default

#### 1451 (2010-08-10)

- update XChat to r1451
- disable identd by default
- remove DNS plugin

#### 1444 (2010-07-30)

- update XChat to r1444
- downgrade Tcl to 8.5
- add Tcl support to the x64 build

#### 1441 (2010-06-15)

- update XChat to r1441
- enable transfer of files bigger than 4 GB

#### 1439 (2010-05-30)

- update XChat to r1439 (2.8.8)

#### 1431-6 (2010-05-30)

- re-enable the transparent background option
- add branding to Plugin-Tray
- installer updates

#### 1431-5 (2010-05-29)

- fix installer
- add DNS plugin status messages

#### 1431-4 (2010-05-28)

- disable the transparent background option
- downgrade GTK+ to more stable 2.16

#### 1431-3 (2010-05-23)

- add portable build support

#### 1431-2 (2010-05-22)

- replace X-Tray with Plugin-Tray

#### 1431 (2010-05-21)

- update XChat to r1431
- include a lot of XChat translations added since 2.8.6

#### 1412-3 (2010-05-02)

- fix GTK function call

#### 1412-2 (2010-05-02)

- re-enable taskbar alerts on x64

#### 1412 (2010-05-02)

- update XChat to r1412
- update GTK+ and friends
- update Visual Studio to 2010
- fix Perl warning message
- include GTK L10n with the installer

#### 1409-9 (2010-04-18)

- fix loading of scrollback

#### 1409-8 (2010-04-03)

- fix X-Tray on x64

#### 1409-7 (2010-04-02)

- disable taskbar notification options

#### 1409-6 (2010-03-31)

- display version numbers everywhere

#### 1409-5 (2010-03-31)

- add DNS plugin
- add EasyWinampControl plugin
- disable Plugin-Tray settings

#### 1409-4 (2010-03-30)

- add X-Tray

#### 1409-3 (2010-03-29)

- plugin linkage fixes

#### 1409-2 (2010-03-29)

- enable IPv6 support
- enable NLS support
- enable Perl support
- enable Tcl support

#### 1409 (2010-03-29)

- initial release

CHAPTER 2

# Settings

## 2.1 Config Files

Config files are stored as plaintext files (which shouldn't be edited by hand). They are located in:

- Windows: `%APPDATA%\HexChat`
- Unix: `~/.config/hexchat`

**Note:** Type the path into your file browser to expand them into a full directory.

Custom directories can be set with the *-d* or *–cfgdir* command line argument.

On Windows there is a portable mode option which makes HexChat store settings inside of a `\config` subdir within the main installation folder. Though this option should only be used if you must for a portable drive or if you lack administrative priviliges for installation.

## 2.2 Network List

The network list contains a list of networks, basic user information, and per network settings. It can be accessed in *HexChat* → *Network List* or by the keyboard shortcut `Control-s`.

### 2.2.1 User Information

This is where basic information is set for the user such as nicknames, these will be used as defaults for networks but can be overridden. The Realname field is entirely optional. The username is often used for options like Server and SASL password.

### 2.2.2 Networks

HexChat comes with a list of default networks but you can easily add your own or edit existing ones. The sorting and names of networks do matter as some commands reference the network by the name here, e.g. **`/newserver`**. The order they are sorted in this list also determines the order of auto-connecting. To edit the order just select one and hit `Shift-up` or `Shift-down`.

#### Servers

Per network you can maintain a list of servers in case of one fails. The syntax for these servers *hostname/port*. The port is entirely optional and can be prefixed by + to signify SSL. If no port is given the default port used is 6667 and 6697 for SSL.

#### Your Details

Unticking *Use global user information* allows unique nicknames or usernames between networks.

#### Connecting

Ticking *Auto connect to this network at startup* combined with *Favorite channels* will allow you to quickly connect and join chats.

The password fields allow you to login to a password protected server or services. They take the syntax *username:password* or just the password if username has been set as mentioned above.

**See also:**

See the FAQ if you have trouble identifying before join.

## 2.3 Channel Options

Some options can be specific to channels. To access these you can right click on any tab or use the command **`/chanopt`**. Any option other than *0* or *1* is considered *unset* and will use the globally set defaults.

## 2.4 Preferences

### 2.4.1 Keyboard Shortcuts

There are two types of keyboard shortcuts in HexChat hardcoded ones which can be found by looking around in the menu, e.g. `Control-s`, and configurable ones in *Settings* → *Keyboard Shortcuts*. These have help messages to guide you through setting up custom bindings.

### 2.4.2 Url Handlers

Url handlers add alternative browsers to you your right click menu on urls (they do not set the default). They take the syntax:

```
!program %s or !"C:\Program Files\Program\program.exe" %s
```

**Note:** If HexChat does not find the executable it will not add it to the menu.

### 2.4.3 Auto Replace

Located in *Settings → Auto Replace* this setting allows you to replace text while typing. The Text column is what it is to detect and the Replace with column is what will replace it. This column accepts color codes from Text Events

For Example:

> text: –> replace: %C8→%O

This will replace that arrow with a green unicode arrow upon pressing enter or space. The keys that check for replace are defined in *Settings → Keyboard Shortcuts*. A trick to avoid replacing it `Shift-Space` since that is not defined by default.

### 2.4.4 CTCP Replies

Custom CTCP replies can be set in *Settings → CTCP Replies* and accept the same format as User Commands with the addition of *%s* for the nick replying to.

**Note:** To hide the default VERSION reply you must **`/set irc_hide_version on`**

## 2.5 Set Command

The set command can be used to change options. The usage is:

**`/set option_name [number|string]`**

Togglable options take *1* for on and *0* for off but as an alternative you can type:

**`/set toggle_option on`**

If no value is given it will show the current value of a setting. You can also use wildcards to find or show multiple values for example

**`/set gui_*`**

Some options such as *gui_tray* require running this after changing:

**`/gui apply`**

This can be avoided by using the preferences window instead, which is recommended, also using the gui will warn if a setting requires restart.

## 2.6 List of Settings

| | |
|---|---|
| **away_auto_unmark** | Toggle automatically unmarking away before message send. |
| **away_omit_alerts** | Toggle omitting alerts when marked as being away. |
| **away_reason** | Default away reason. |
| **away_show_message** | Toggle announcing of away messages. |
| **away_show_once** | Show identical away messages only once. |
| **away_size_max** | How many users can be away in userlist before they are not colored. |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **away_timeout** | How often in seconds to check for max size for colors in userlist. |
| **away_track** | Toggle color change for away users in userlist. |
| **completion_amount** | How may nicks starting with input there should be before all are shown in text box. (E.g. if you have 'k' and completion_amount is set to 6, and there are 6 more people beginning with 'k' in the userlist, then all of the nicks starting with that are shown in the text box. To always cycle nicks, set to 123456 (or any other high number). |
| **completion_auto** | Toggle automatic nick completion. |
| **completion_sort** | Toggle nick completion sorting in "last talk" order. |
| **completion_suffix** | Suffix to be appended to nicks after completion. |
| **dcc_auto_chat** | Toggle auto accept for DCC chats. |
| **dcc_auto_recv** | How to accept DCC transfers.<br>• 0=Ask for confirmation<br>• 1=Ask for download folder<br>• 2=Save without interaction |
| **dcc_auto_resume** | Toggle auto resume of DCC transfers. |
| **dcc_blocksize** | The blocksize for DCC transfers. |
| **dcc_completed_dir** | Directory to move completed files to. |
| **dcc_dir** | Directory to download files to from DCC. |
| **dcc_fast_send** | Toggle speed up of DCC transfers by not waiting to heard if last part was received before sending next (currently disabled on Win32). |
| **dcc_global_max_get_cps** | Max file transfer speed for all downloads combined in bytes per second. |
| **dcc_global_max_send_cps** | Max file transfer speed for all uploads combined in bytes per second. |
| **dcc_ip** | DCC IP address to bind to. |
| **dcc_ip_from_server** | Get address from IRC server. |
| **dcc_max_get_cps** | Max file transfer speed for one download in bytes per second. |
| **dcc_max_send_cps** | Max file transfer speed for one upload in bytes per second. |
| **dcc_permissions** | What permissions to set on received files. (It's a CHMOD value in decimal, e.g. to CHMOD a file to 644, which is octal, you need to set dcc_permissions to 420, which is it's decimal equivalent) |
| **dcc_port_first** | First DCC port in range (leave ports at 0 for full range). |
| **dcc_port_last** | Last DCC port in range (leave ports at 0 for full range). |
| **dcc_remove** | Toggle automatic removal of finished/failed DCCs. |
| **dcc_save_nick** | Toggle saving of nicks in filenames. |
| **dcc_send_fillspaces** | Replace spaces in filenames with underscores. |
| **dcc_stall_timeout** | Time in seconds to wait before timing out during a DCC send. |
| **dcc_timeout** | Time in seconds to wait before timing out a DCC transfer waiting to be accepted. |
| **flood_ctcp_num** | Number of CTCPs within flood_ctcp_time to be considered a flood. |
| **flood_ctcp_time** | Time in seconds for use with flood_ctcp_num. |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **flood_msg_num** | Number of messages within flood_msg_time to be considered a flood. |
| **flood_msg_time** | Time in seconds for use with flood_msg_num. |
| **gui_autoopen_chat** | Toggle auto opening of Direct Chat Window on DCC Chat. |
| **gui_autoopen_dialog** | Toggle auto opening of dialog windows. |
| **gui_autoopen_recv** | Toggle auto opening of transfer window on DCC Recv. |
| **gui_autoopen_send** | Toggle auto opening of transfer window on DCC Send. |
| **gui_chanlist_maxusers** | Maximum number of users in channels to be listed in List of Channels. |
| **gui_chanlist_minusers** | Minimum number of users in channels to be listed in List of Channels. |
| **gui_compact** | Toggle compact mode (more or less spacing between user list/channel tree rows). |
| **gui_dialog_height** | New dialog height in pixels. |
| **gui_dialog_left** | The X co-ordinance of dialogs when opened. |
| **gui_dialog_top** | The Y co-ordinance of dialogs when opened. |
| **gui_dialog_width** | New dialog width in pixels. |
| **gui_hide_menu** | Hide or unhide menu bar. |
| **gui_input_icon** | Toggle user mode icon in the nick box. |
| **gui_input_nick** | Toggle the nick box in the input box. |
| **gui_input_spell** | Enable or disable spell checking. |
| **gui_input_style** | Toggle use of text box colors and fonts in input box. |
| **gui_join_dialog** | Toggle join dialog after connect. |
| **gui_lagometer** | Toggle types of Lag-O-Meters.<br>• 0=Off<br>• 1=Graph<br>• 2=Text<br>• 3=Both |
| **gui_lang** | Set GUI language. Possible values are from 0 to 50 (Win32 only). |
| **gui_mode_buttons** | Toggle mode buttons. |
| **gui_pane_left_size** | Change size left pane. |
| **gui_pane_right_size** | Change size right pane. |
| **gui_pane_divider_position** | Saves position of divider when channel switcher and user list are on the same side. |
| **gui_pane_right_size_min** | FIXME |
| **gui_quit_dialog** | Toggle quit dialog. |
| **gui_slist_fav** | Toggle showing favorites only in network list. |
| **gui_slist_select** | The number of the server to select by default in the server list starting at 0. (E.g. to select the 67th server, set it to 66) |
| **gui_slist_skip** | Toggle server list on startup. |
| **gui_tab_chans** | Open channels in tabs instead of windows. |
| **gui_tab_dialogs** | Open dialogs in tabs instead of windows. |
| **gui_tab_dots** | Toggle dotted lines in the channel tree. |
| **gui_tab_icons** | Toggle channel tree icons. |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **gui_tab_layout** | Use treeview or tabs.<br>• 0=Tabs<br>• 2=Treeview |
| **gui_tab_newtofront** | When to focus new tabs.<br>• 0=Never<br>• 1=Always<br>• 2=Only on requested tabs |
| **gui_tab_pos** | Set position of tabs.<br>• 1=Left-Upper<br>• 2=Left<br>• 3=Right-Upper<br>• 4=Right<br>• 5=Top<br>• 6=Bottom<br>• 7=Hidden |
| **gui_tab_server** | Open an extra tab for server messages. |
| **gui_tab_small** | Set small tabs.<br>• 0=Off<br>• 1=Small tabs<br>• 2=Extra small tabs |
| **gui_tab_sort** | Toggle alphabetical sorting of tabs. |
| **gui_tab_trunc** | Number or letters to shorten tab names to. |
| **gui_tab_utils** | Open utils in tabs instead of windows. |
| **gui_throttlemeter** | Toggle types of throttle meters.<br>• 0=Off<br>• 1=Graph<br>• 2=Text<br>• 3=Both |
| **gui_topicbar** | Toggle topic bar. |
| **gui_tray** | Enable system tray icon. |
| **gui_tray_away** | Automatically mark away/back when the tray is toggled. |
| **gui_tray_blink** | Toggle tray icon blinking or using static images. |
| **gui_tray_close** | Close to tray. |
| **gui_tray_minimize** | Minimize to tray. |
| **gui_tray_quiet** | Only show tray balloons when hidden or iconified. |
| **gui_ulist_buttons** | Toggle userlist buttons. |
| **gui_ulist_count** | Toggle displaying user count on top of the user list. |
| **gui_ulist_doubleclick** | Command to run upon double click of user in userlist. |
| **gui_ulist_hide** | Hides userlist. |
| **gui_ulist_icons** | Toggle use of icons instead of text symbols in user list. |
| **gui_ulist_pos** | Set userlist position.<br>• 1=Left-Upper<br>• 2=Left-Lower<br>• 3=Right-Upper<br>• 4=Right-Lower |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **gui_ulist_resizable** | Toggle resizable userlist. |
| **gui_ulist_show_hosts** | Toggle user's hosts displaying in userlist. (requires irc_who_join) |
| **gui_ulist_sort** | How to sort users in the userlist.<br>• 0=A-Z with Ops first<br>• 1=A-Z<br>• 2=A-Z with Ops last<br>• 3=Z-A<br>• 4=Unsorted |
| **gui_ulist_style** | Toggle use of text box colors and fonts in userlist. |
| **gui_url_mod** | How to handle URLs when clicked. (And what to hold.)<br>• 0=Left Click Only<br>• 1=Shift<br>• 2=Caps Lock<br>• 4=CTRL<br>• 8=ALT |
| **gui_usermenu** | Toggle editable usermenu. |
| **gui_win_height** | Main window height in pixels. |
| **gui_win_left** | The X co-ordinance of main window when opened. |
| **gui_win_modes** | Show channel modes in title bar. |
| **gui_win_save** | Toggles saving of state on exit. |
| **gui_win_state** | Default state of the main window.<br>• 0=Not Maximized<br>• 1=Maximized |
| **gui_win_swap** | Swap the middle and left panes (allows side-by-side userlist/tree). |
| **gui_win_top** | The Y co-ordinance of main window when opened. |
| **gui_win_ucount** | Show number of users in title bar. |
| **gui_win_width** | Main window width in pixels. |
| **identd** | Toggle internal IDENTD (Win32 only). |
| **input_balloon_chans** | Show tray balloons on channel messages. |
| **input_balloon_hilight** | Show tray balloons on highlighted messages. |
| **input_balloon_priv** | Show tray balloons on private messages. |
| **input_balloon_time** | How long balloon messages should be displayed. (2.8.8+) |
| **input_beep_chans** | Toggle beep on channel messages. |
| **input_beep_hilight** | Toggle beep on highlighted messages. |
| **input_beep_priv** | Toggle beep on private messages. |
| **input_command_char** | Character used to execute commands. (E.g. if set to '[' then you would use commands like '[me jumps around') |
| **input_filter_beep** | Toggle filtering of beeps sent by others. |
| **input_flash_chans** | Toggle whether or not to flash taskbar on channel messages. |
| **input_flash_hilight** | Toggle whether or not to flash taskbar on highlighted messages. |
| **input_flash_priv** | Toggle whether or not to flash taskbar on private messages. |
| **input_perc_ascii** | Toggle interpreting of %nnn as ASCII value. |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **input_perc_color** | Toggle interpreting of %C, %B as color, bold, etc. |
| **input_tray_chans** | Blink tray icon on channel messages. |
| **input_tray_hilight** | Blink tray icon on highlighted messages. |
| **input_tray_priv** | Blink tray icon on private messages. |
| **irc_auto_rejoin** | Toggle auto rejoining when kicked. |
| **irc_ban_type** | The default ban type to use for all bans. (requres irc_who_join)<br>• 0=*!*@*.host<br>• 1=*!*@domain<br>• 2=*!*user@*.host<br>• 3=*!*user@domain |
| **irc_conf_mode** | Toggle hiding of join, part and quit messages. (More info)<br>• 0=Show join/part/quits<br>• 1=Hide join/part/quits |
| **irc_extra_hilight** | Extra words to highlight on. |
| **irc_hide_version** | Toggle hiding of VERSION reply. |
| **irc_id_ntext** | $4 in the channel message, channel message hilight and private message events if unidentified. |
| **irc_id_ytext** | $4 in the channel message, channel message hilight and private message events if identified. |
| **irc_invisible** | Toggle invisible mode (+i). |
| **irc_join_delay** | How long to delay auto-joining a channel after connect. |
| **irc_logging** | Toggle logging. |
| **irc_logmask** | Mask used to create log filenames (strftime details: Windows Unix). |
| **irc_nick1** | First choice nick. |
| **irc_nick2** | Second choice nick. |
| **irc_nick3** | Third choice nick. |
| **irc_nick_hilight** | What nicks to highlight when they talk. |
| **irc_notice_pos** | Placement of Notices:<br>• 0 = Automatic<br>• 1 = Open extra (notices) tab<br>• 2 = Always place in front tab |
| **irc_no_hilight** | Nicks not to highlight on. |
| **irc_part_reason** | Default reason when leaving channel. |
| **irc_quit_reason** | Default quit reason. |
| **irc_raw_modes** | Toggle RAW channel modes. |
| **irc_real_name** | Real name to be sent to server. |
| **irc_servernotice** | Toggle receiving of server notices. |
| **irc_skip_motd** | Toggle skipping of server MOTD. |
| **irc_user_name** | Username to be sent to server. |
| **irc_wallops** | Toggle receiving wallops. |
| **irc_who_join** | Toggle running WHO after joining channel. |
| **irc_whois_front** | Toggle whois results being sent to currently active tab. |
| **net_auto_reconnect** | Toggle auto reconnect to server. |
| **net_auto_reconnectonfail** | Toggle auto reconnect upon failed connection. (Unix only command, not available on Windows) |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **net_bind_host** | Network address to bind HexChat to. |
| **net_ping_timeout** | How long server ping has to be to timeout. |
| **net_proxy_auth** | Toggle proxy authentication. |
| **net_proxy_host** | Proxy host to use. |
| **net_proxy_pass** | Password to use if proxy authentication is turned on. |
| **net_proxy_port** | Port to use for proxy host. |
| **net_proxy_type** | Type of proxy to use.<br>• 0=Disabled<br>• 1=Wingate<br>• 2=Socks4<br>• 3=Socks5<br>• 4=HTTP<br>• 5=MS Proxy (ISA) |
| **net_proxy_use** | What to use proxies for (if set).<br>• 0=All<br>• 1=IRC Only<br>• 2=DCC Only |
| **net_proxy_user** | Username to use if proxy authentication is turned on. |
| **net_reconnect_delay** | How many seconds to wait before reconnection. |
| **net_throttle** | Toggle flood protection (to keep from getting kicked). |
| **notify_timeout** | How often in seconds to check for users in your notify list. |
| **notify_whois_online** | Toggle performing WHOIS on users on your notify list when they come online. |
| **perl_warnings** | Toggle perl warnings. |
| **sound_dir** | Directory where sounds are located. |
| **stamp_log** | Toggle timestamps in logs. |
| **stamp_log_format** | Format to use for log timestamps (strftime details: Windows Unix). |
| **stamp_text** | Toggle timestamps in text box. |
| **stamp_text_format** | Format to use for timestamps in textbox (strftime details: Windows Unix). |
| **text_autocopy_color** | Toggle automatic copying of color information. |
| **text_autocopy_stamp** | Toggle automatic copying of time stamps. |
| **text_autocopy_text** | Toggle automatic copying of selected text. |
| **text_background** | Sets the background image for text box. |
| **text_color_nicks** | Toggle colored nicks. |
| **text_font** | All fonts to be used (main and alternative fonts combined, shouldn't be edited manually). |
| **text_font_main** | Primary font to be used. |
| **text_font_alternative** | Alternative fonts to be used for glyphs not supported by the primary font. |
| **text_indent** | Toggle text indentation. |
| **text_max_indent** | Max pixels to indent text with. |
| **text_max_lines** | Max number or scrollback lines. |
| **text_replay** | Reloads conversation buffers on next startup. |
| **text_search_case_match** | Toggle performing a case-sensitive search. |
| **text_search_backward** | Toggle searching from newest text line to the oldest. |

Continued on next page

Table 2.1 – continued from previous page

| | |
|---|---|
| **text_search_highlight_all** | Toggle highlighting all occurences and underlining of the current occurence. |
| **text_search_follow** | Toggle search for newly arriving messages. |
| **text_search_regexp** | Toggle regarding search string as a regular expression. |
| **text_show_marker** | Toggle red marker line feature. |
| **text_show_sep** | Toggle separator line. |
| **text_spell_langs** | List of languages to have spelling for, by language codes, separated by commas. |
| **text_stripcolor_msg** | Toggle stripping colors from messages. |
| **text_stripcolor_replay** | Toggle stripping colors from scrollback. |
| **text_stripcolor_topic** | Toggle stripping colors from topic. |
| **text_thin_sep** | Use thin separator line instead of thick line. |
| **text_transparent** | Toggle transparent background. |
| **text_wordwrap** | Toggle wordwrap. |
| **url_grabber** | Toggle URL grabber. |
| **url_grabber_limit** | Limit the number of URLs handled by the url grabber. |
| **url_logging** | Toggle logging URLs to *<config>/url.log*. |

CHAPTER 3

# Commands

Commands in HexChat are prefixed with / and to escape them you can type it twice e.g. //

HexChat will first try to run plugin commands, then user commands, then client commands, and lastly send it directly to the server.

## 3.1 User Commands

User commands can be used to create aliases, to run multiple commands at a time, or more complex custom commands. They are set in *Settings* → *User Commands*.

An alias is just a shortcut refering to an existing command, for example */j* refers to */join &2*

Naming two user commands the same thing will run both in the order they are listed.

For more complex commands you can use these codes:

- **%c** Current channel
- **%e** Current network
- **%m** Machine info
- **%n** Your nick
- **%t** Time/date
- **%v** HexChat version
- **%<num>** Word
- **&<num>** Word from end of line

## 3.2 List of Commands

To get a full list of commands and what they do type **`/help -l`**.

CHAPTER 4

# Appearance

## 4.1 Theme Files

We distribute themes as `.hct` files. These are just simply ZIP archives that you can *extract* manually into your config folder with HexChat closed. Some themes can be found here.

On unix for example you can extract these with this command while HexChat is closed:

```
unzip ~/Downloads/monokai.hct -d ~/.config/hexchat
```

The theme manager is an optional external application, which is included with the installer on Windows and buildable on Unix, that simply helps install themes.

## 4.2 Theming

### 4.2.1 Colors

Colors are defined in *Settings* → *Preferences* → *Colors*. Text Colors set the palette for events to use. The rest like background color directly affect parts of the UI.

mIRC colors (00-15) are what you refer to when sending colored text over IRC for others to see and vice versa, because of this they should somewhat follow a set of standards so clients can agree 04 is red.

Local colors (16-31) are to be used by HexChat only and can be anything you wish, these are typically what you use in your events.

### 4.2.2 Text Events

Text events control the look of every event you see. They can be customized in *Settings* → *Text Events* using these codes to format it:

- **%C<fg>,<bg>** Color code (e.g. %C02,00 is blue on grey)
- **%R** Reverse color
- **%U** Underlined text
- **%B** Bold text
- **%I** Italic text (2.10.0+)
- **%H** Hide text

- **%O** Normal text
- **%%** Escaped %
- **$t** Text separator (tab character)
- **$aXXX** Ascii value
- **$<num>** Event information

**Note:** Always hit enter after editing a field.

## 4.2.3 Icons

HexChat comes with built in icons for the tray, user list, and channel tree (which can be disabled in Preferences). You can use custom icons by placing icons (16x16 recommended) in an `icons` subdir, which may need to be created, within your config folder. The icons must be named exactly as follows including file extensions:

- User List
  - ulist_netop.png
  - ulist_founder.png
  - ulist_owner.png
  - ulist_op.png
  - ulist_halfop.png
  - ulist_voice.png
- Channel Tree
  - tree_channel.png
  - tree_dialog.png
  - tree_server.png
  - tree_util.png
- Tray Icon
  - tray_fileoffer.png
  - tray_highlight.png
  - tray_message.png
  - hexchat.png

## 4.2.4 Gtk Theme

To customize more than just text color such as the window borders you must install a gtk theme. Here are some simple instructions for using them on Windows.

The only requirement for themes on Windows is that they are for gtk2 and they use the Pixbuf engine. Themes requiring other engines will work but not look correct. HexChat has also only shipped this engine since 2.10.0. If you use < 2.12.0 you need to install HexChat with the *Gtk theme* option unchecked as this will avoid the installer overwriting it on updates.

Themes can be either installed globally (changing all gtk2 applications) or to HexChat specifically. The latter requires write permissions to HexChat's install dir (e.g. in Program Files). Global themes are set in *%USERPROFILE%\.gtkrc-2.0* but will not be covered here.

This example will use the Vertex theme. Which contains an unoffensive dark theme and *mostly* uses the Pixbuf engine. Zips are on the releases page.

Note that *<installdir>* refers to the full path where HexChat is installed which by default is *C:\Program Files\HexChat*. Also these directories may need to be created.

1. Create the directory *<installdir>\share\themes\vertex\gtk-2.0*
2. Extract contents of *vertex-gtk\gtk-2.0-dark\* into this directory.
3. Create an empty file at *<installdir>\etc\gtk-2.0\gtkrc* (with no file extension!)
4. Edit it and add `gtk-theme-name = "vertex"`

## 4.3 Buttons, Menus, and Popups

### 4.3.1 Userlist Popup

Popups are shown when you right click on a nickname, either in the userlist or in the main chat itself. These can be edited in *Settings → Userlist Popup*

The Name column can take either just the name of the entry, *SUB/ENDSUB* for submenus, *SEP* for separators, and *TOGGLE* for toggleable options. Suffix the name with *~GTK_ICON_NAME~* to show an icon (see Gtk Stock Items) and prefix a character with _ for keyboard shortcuts (e.g. N_ame will bind a).

The Command column can take any command with text formatted using the same codes as text events and on top of that they also have their own codes:

- **%a** all selected nicks
- **%c** current channel
- **%h** selected nick's hostname
- **%m** machine info
- **%n** your nickname
- **%s** selected nickname
- **%t** time/date
- **%u** selected nick's account (2.9.6+)

As a sidenote the **gui_ulist_doubleclick** setting can run a command using these codes when double-clicking a nick in the userlist.

### 4.3.2 Userlist Buttons

Buttons are shown below the userlist, can be edited in *Settings → Userlist Buttons*, and take the same syntax as Userlist Popup for commands.

### 4.3.3 Usermenu

In order to add custom entries to your menu you need to first enable the usermenu with the command **`/set gui_usermenu on`** which may require a restart. Once this is enabled you can go to *Usermenu* → *Edit this Menu* to add any command you would like. For menu entries it supports the same as Userlist Popups.

CHAPTER 5

# Addons

HexChat ships with a handful of useful plugins that extend the functionality of the client. In order to auto-load custom ones you just place them in an `addons` subdir within your config folder, manually with **`/load`** and **`/unload`**, or with *Window* → *Plugins and Scripts*.

With the included scripting interfaces you can also use easier to create but equally powerful scripts in languages such as Python. These are loaded/unloaded in the same manner as plugins.

## 5.1 Exec

The exec plugin provides the /exec command for Windows, it is built in on Unix.

With Exec you can perform commands as if you ran them in the command line. The output will be printed at once, in the end of execution. If the command takes more than 10 seconds to complete, it gets aborted to avoid locking down HexChat. Usage:

```
/exec ping google.com
```

## 5.2 FISHLiM

Adds FiSH encryption support. You set a password for a conversation/channel, and then all your messages get encrypted. Only those who know the password will be able to read your messages. Usage is simple: first you set the password:

```
/setkey yoursecretkey
```

Then you let your fellow chatters know this password. Once they also set the password on their sides, they'll receive messages decrypted and send them encrypted, too.

You can remove the key with:

```
/delkey #channel
```

## 5.3 Update Checker

Automatically checks for updates available. Can be manually checked via *Help* → *Check for Updates* or by the command:

```
/updchk
```

## 5.4 Sysinfo

Prints out basic system information on both Windows and Unix. You can either activate in *Window* → *Display System Info* or type:

**`/sysinfo`**

## 5.5 Checksum

Automatically calculates the SHA-256 checksum of files sent and recieved via DCC.

## 5.6 Winamp

Displays your currently playing song via *Window* → *Display Current Song* or by command:

**`/winamp`**

**Note:** Foobar2000 can also be used with the foo_winamp_spam plugin.

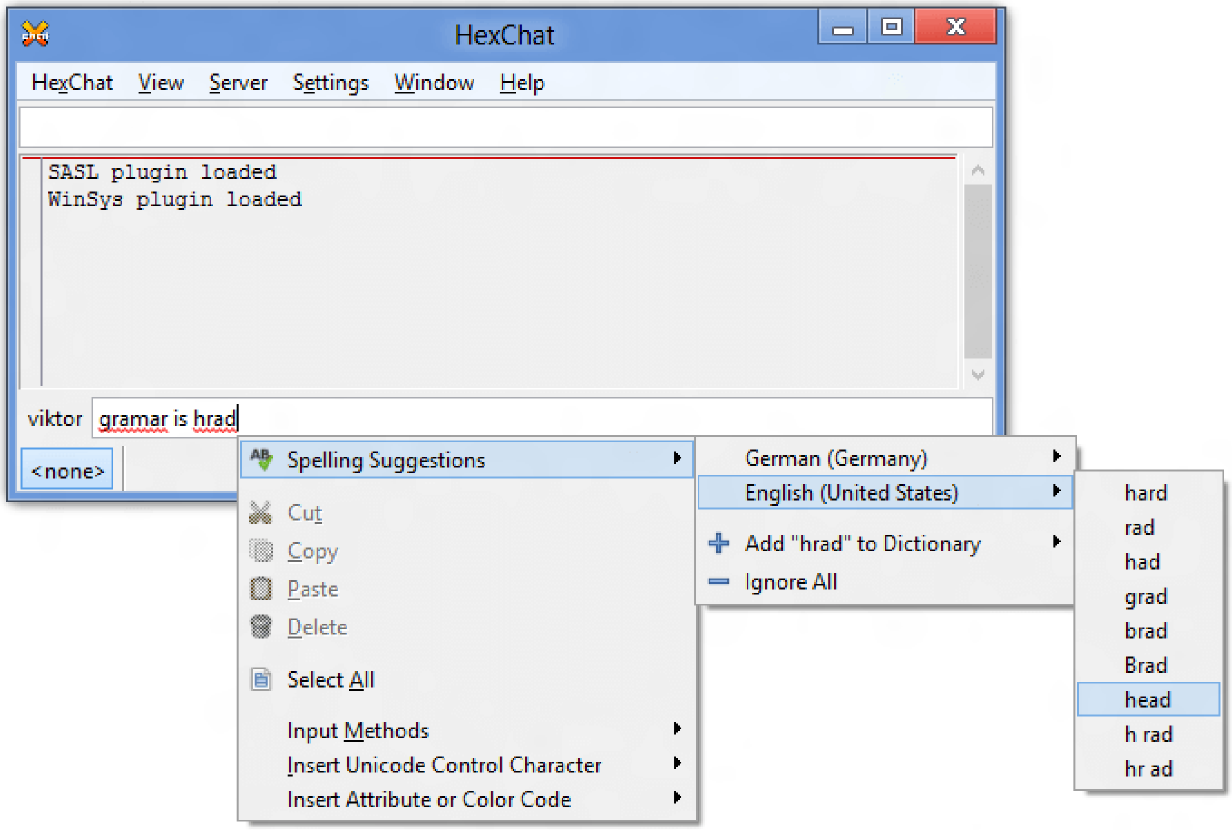
CHAPTER 6

# Tips & Tricks

## 6.1 Spell Check

### 6.1.1 Windows

HexChat for Windows uses MySpell for spelling via the Enchant library. The required libraries are included in the installer, but the dictionaries are big so they are distributed in a separate archive. They're from Debian Lenny.



```
SASL plugin loaded
WinSys plugin loaded
```

If you want to have spelling, do the following:

1. Download the Spelling Dictionaries self-extracting archive
2. Specify the languages you wish to have spelling for in *Settings* → *Preferences* → *Interface* → *Input box*. You need to specify their language codes, see `%LOCALAPPDATA%\enchant\myspell` for hints. By default,

HexChat uses the LC_ALL environmental variable, or if it's unset, it falls back to en_US.

**Note:** For portable installs, you can use the `share\myspell\dicts` subfolder instead of `%LOCALAPPDATA%\enchant\myspell` (both will work, but the former one can be carried on a pendrive unlike the latter one).

### 6.1.2 Unix

Install enchant and your spelling dictionaries via your package manager (myspell-en-us on the Debians or hunspell-en-US everywhere else for English). Then make sure to enable spelling under *Settings* → *Preferences* → *Interface* → *Input box*.

In 2.10.0+ you can then list your languages in *Settings* → *Preferences* → *Interface* → *Input box* by their language codes (i.e. de_DE for german), separated by commas.

In older versions the language can be overriden by starting HexChat with "*LANG=en_US hexchat*" (You can edit your hexchat.desktop file to do this every launch)

## 6.2 Localization

In order to start HexChat in a different language (for which a translation exists here) you can use the regional settings of Windows, or set the LC_ALL user environmental variable. The value of the variable must be the two letter country code for your country. If in doubt, have a look at the sharelocale folder. You have to restart HexChat for the changes to apply.

You can also use a batch file to affect only HexChat:

```
@echo off
set LC_ALL=en
start hexchat.exe
```

This sets the language to English. You may use *fr* for French, *de* for German, etc. Save the code above as `run.bat`, and copy it to the HexChat install folder. You can then start HexChat in the desired language by running the batch file.

## 6.3 Special Glyphs

There are many symbols which may not be supported by the main font you selected to use in HexChat, especially Asian glyphs and special characters, like a peace sign. In this case, you'll see "lego blocks" instead of them.

To circumvent this, you need to have alternative fonts for glyphs not supported by your current font. On Unix this is handled automatically. On Windows you can specify them in *Settings* → *Preferences* → *Chatting* → *Advanced* → *Alternative fonts*. By default, it is set to *Arial Unicode MS,Segoe UI Emoji,Lucida Sans Unicode,Meiryo,Symbola,Unifont*, which should cover most characters (note that Unifont does not come with Windows).

There are many available fonts that try to cover most of unicode:

- Unifont
- Symbola
- Quivira

In case you still get lego blocks, you'll need to add additional fonts to the list which support those obscure glyphs. Feel free to extend the list. You only need to specify font names, other info (such as size, weight, style etc.) should be omitted, otherwise those entries will be ignored. All font names must be separated by a comma and there mustn't be spaces before and/or after commas.

Please bear in mind that for some reason certain fonts that can display a certain glyph when used as the main font may not work when specified as an alternative font so you might have to play around it a bit.

## 6.4 Client Certificates

Client Certificates allows you to identify to networks services using a certificate. Please do not mistake it for server certificate which allows you to connect to network with invalid certificate, right now HexChat can't do it.

To use one you need to put your certificate file inside `certs` directory in HexChat's config folder.

Certificate should be named after the network where it will be used, for example if you want to use it on *Rizon*, certificate file should look like this: *Rizon.pem*. If that does not exist every network will try *client.pem*. It can not be encrypted and require a password.

An example of creating a cert on unix:

```
openssl req -x509 -nodes -days 365 -newkey rsa:4096 -keyout ~/.config/hexchat/certs/client.pem -out
```

On networks that support it you can use SASL EXTERNAL in the network list. If a network does not support this but does support normal SASL usually that would be the better option.

### 6.4.1 Note on Custom Server Certificates

On Windows it is possible to edit *cert.pem* file in HexChat main installation directory and add custom certificate there. But this method isn't very effective as *cert.pem* is overwritten each time HexChat installer is used.

## 6.5 Notice Placement

Other than channel messages and private messages, IRC has a notice type of message. This is intended to be used as a reply, something that will not cause the other client to send any acknowledgement back. When HexChat displays these messages, it shows them in a tab that it figures is appropriate.

### 6.5.1 Why replies from ChanServ may not appear in the current tab

When HexChat decides where to print a notice, it does so in the following order:

1. In a query window you have with that user
2. In the front tab, if the tab is a channel, the other user is on that channel, and you are on the correct network
3. In the last joined channel you have in common with the other user
4. The current tab, if you are on the same network
5. The last tab you looked at that shares the correct network with the other user

This means that if you issue a **`/cs info #yourchannel`** from your channel, the reply may show up elsewhere if ChanServ isn't in your channel, but is in some other channel.

### 6.5.2 How to make notices show up in a consistent location

The simplest method is to set the location in *Settings* → *Preferences* → *Channel switcher* → *Placement of notices*, and select "in an extra tab" or "in the front tab". The former will cause all server notices to go into a (snotices) tab, and all user notices to go into a (notices) tab. The latter will always print the notices where you are, this can cause odd positioning of channel notices but you will never miss them.

If you know who will notice you before hand, you can simply query the user before they notice you. This way, all notices from that user will show up in the query tab. In the case of ChanServ, this may allow an easier archive of commands you have done anyway.

For other locations, a separate script would be required. While not currently implemented, it would be possible with a script to treat all notices like private messages (open a new query window when received), or place them in a specific existing tab, such as the server tab. At this point, the choice is up to you (or whoever designs the script).

## 6.6 How the marker line works

The marker line is a very useful tool to keep track of what you have and have not read in a channel but it's behavior is non-obvious at times. It just follows a few simple rules though.

A line is created when new information is printed in a context that is not currently visible. This means the window is in the background, another tab is selected, or you are scrolled up.

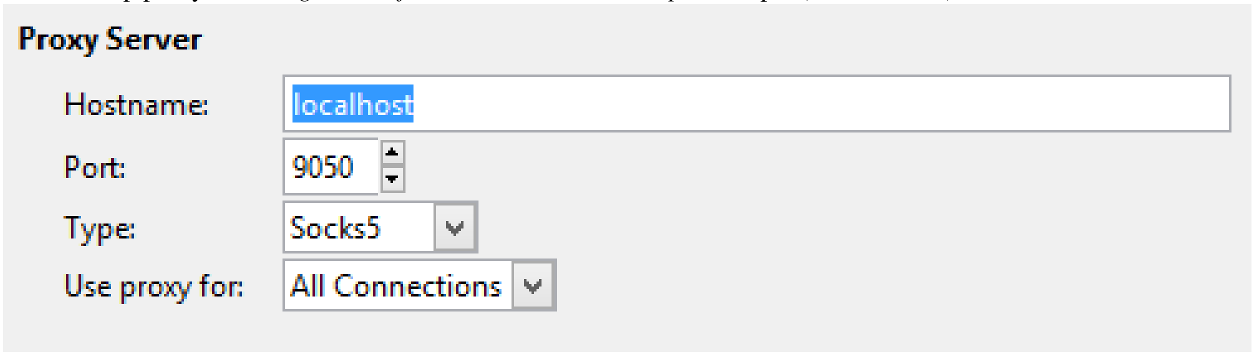
This line by design only automatically resets when it is seen. One common issue here is that the marker line is at the very top of your scrollback so you very unlikely to see it. This can happen with bnc playback for example where you get a lot of messages at once.

HexChat has two shortcuts to reset the marker line also. Ctrl+M will reset the the marker line directly. Ctrl+Shift+M will scroll to where the marker was which is quite useful if you actually care about the scrollback.

Once a marker line is "reset" it does not instantly get created at the bottom it will only be created if it matches the conditions mentioned above (not being visible).

## 6.7 Tor

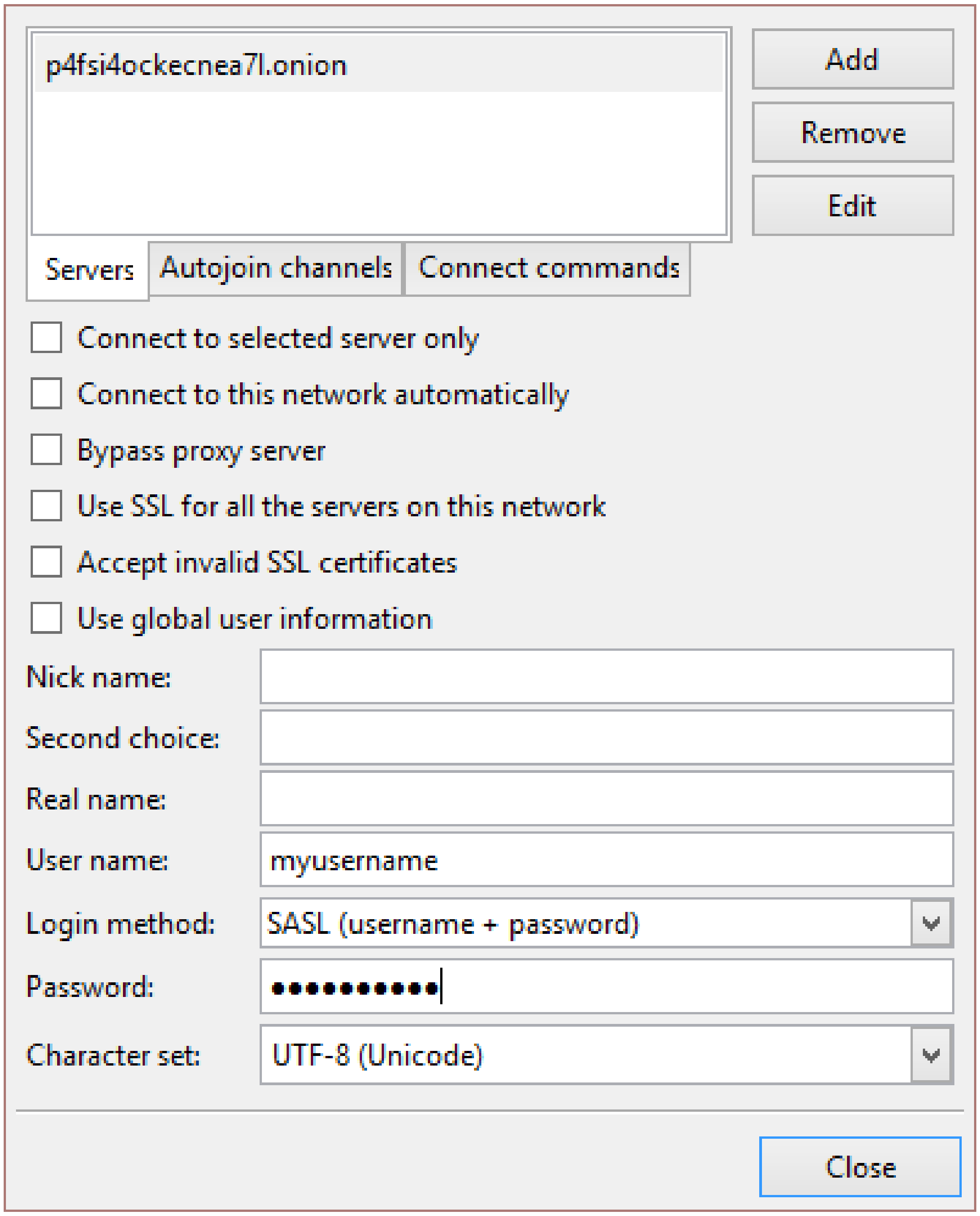
1. Find a network that allows tor (most don't). Example: freenode
2. Get tor working. Refer to the tutorial from official tor website (instructions for Windows and Linux). For windows - Browser bundle is an easy way to test.
3. Set up proxy in *Settings* → *Preferences* → *Network Setup*. Example (with defaults):



**Proxy Server**

| | |
|---|---|
| Hostname: | localhost |
| Port: | 9050 |
| Type: | Socks5 |
| Use proxy for: | All Connections |

4. Setup the network in *HexChat* → *Network List*. (Note: use the updated information from freenode's site.) Example:


p4fsi4ockecnea7l.onion

☐ Connect to selected server only
☐ Connect to this network automatically
☐ Bypass proxy server
☐ Use SSL for all the servers on this network
☐ Accept invalid SSL certificates
☐ Use global user information

| | |
|---|---|
| Nick name: | |
| Second choice: | |
| Real name: | |
| User name: | myusername |
| Login method: | SASL (username + password) |
| Password: | ●●●●●●●●●●● |
| Character set: | UTF-8 (Unicode) |

## 6.8 Twitch

Twitch.tv uses irc for chat so you can use a regular client for chat but it is a very customized irc that has some extra requirements.

In the Network List add a new network and for the server use *irc.chat.twitch.tv* with SSL. You must have your nickname match your twitch account. For the login method choose *Server Password* and generate a password on this website http://twitchapps.com/tmi

To enhance your experience I recommend using the twitch.lua script.

CHAPTER 7

# Contributor Documentation

## 7.1 How to Help

### 7.1.1 Translation

Translation is handled by Transifex. Simply register on the site and apply for membership to a translation team for your language. Approval may take some time.

**Note:** Your Transifex email will be public in the created files.

### 7.1.2 Documentation

For simple edits of the documentation just go to the page and click *Show/Edit on Github* on the left side, fork the repo, edit it, and submit a pull request.

For more complex additions I direct you to documentation for the projects we use:

- reStructuredText for the markup language.
- Pandoc for converting existing docs.
- Github and ReadTheDocs for hosting.
- Sphinx for generating the docs.

### 7.1.3 Bug Reports

We use Github Issues for our bug reports. Be sure to search on there for similar issues before posting your own, if it already exists pointing out you also have the issue never hurts. With the bug report include at minimum the information from *help -> about*.

**Note:** Issues is not a forum for asking questions, please direct those to IRC for now.

### 7.1.4 IRC Support

Anybody who enjoys helping others and understands much of what's on this site is welcome to help new users in our official irc channel on freenode, #hexchat.

## 7.2 Developers

### 7.2.1 Building

#### Windows

##### Software

Download and install (in their default install paths):

- Visual Studio 2015 Community
- Inno Setup 5.5 Unicode
- Inno Download Plugin
- 7-Zip
- gendef (extract to *C:\gtk-build*)
- msgfmt (extract to *C:\gtk-build*)
- WinSparkle (extract to *C:\gtk-build*)

##### Source code

Download the HexChat source code and extract it to somewhere. You will work in the extracted *hexchat* folder from now.

##### GTK+

Create a folder for GTK+, referred to as *YourDepsPath* from now (*C:\gtk-build\gtk* by default). Specify the absolute path to *YourDepsPath* in *win32\hexchat.props* with the *YourDepsPath* property. Download:

- GTK+ x86 bundle
- GTK+ x64 bundle

Extract them to *win32* and *x64* in *YourDepsPath*.

**See also:**

If you would like to build GTK+ yourself, read this guide.

##### Language interfaces

You can skip this step, but then you won't be able to generate the installer. Download:

- Perl 5.20 x86 (install to *C:\gtk-build\perl-5.20\Win32*)
- Python 2.7 x86 (install to *C:\gtk-build\python-2.7\Win32*)

- Python 3.5 x86 (install to *C:\gtk-build\python-3.5\Win32*)
- Perl 5.20 x64 (install to *C:\gtk-build\perl-5.20\x64*)
- Python 2.7 x64 (install to *C:\gtk-build\python-2.7\x64*)
- Python 3.5 x64 (install to *C:\gtk-build\python-3.5\x64*)

You can use other paths, but then you must update the related properties in *win32\hexchat.props* accordingly.

#### Building

Open PowerShell as administrator and run:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

- If you're on 32-bit Windows, this is *C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe*
- If you're on 64-bit Windows, this is *C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe* (notice that this is the 32-bit PowerShell executable)

Open *win32\hexchat.sln*, right click on the *release/installer* (or *release/copy* if you skipped the language interfaces) project and set it as the startup project. Now you can compile from under the *Build* menu to your taste.

If everything went fine, the resulting binaries will be found in *hexchat-build\Win32* and/or *hexchat-build\x64*. It was easy, wasn't it?

### Unix

First of all, you have to install the build dependencies. Package names differ across distros, so be creative and check your *configure* output if you get an error.

Most package-managers can get the dependencies for you:

- dnf: dnf-builddep hexchat
- apt: apt-get build-dep hexchat && apt-get install build-essential autoconf-archive libluajit-5.1-dev
- apt (Older Ubuntu): apt-get install automake autoconf-archive imagemagick intltool libcanberra-dev libdbus-glib-1-dev libglib2.0-dev libgtk2.0-dev libnotify-dev libpci-dev libperl-dev libproxy-dev libssl-dev libtool libluajit-5.1-dev python-dev build-essential

HexChat has its source code hosted using Git, so you have to install Git as well. When it's ready, you can start the actual compilation, which is basically:

```
git clone https://github.com/hexchat/hexchat.git
cd hexchat
./autogen.sh
make
sudo make install
```

This will compile with defaults. See `./configure --help` for more info about flags.

#### Building Theme Manager

The theme manager isn't built by default on Unix. To do so get hexchat's source as mentioned above and install MonoDevelop with your package manager of choice then run this:

```
./configure --with-theme-manager
make && sudo make install
```

## OS X

### Homebrew

HexChat can be installed via Homebrew. The Homebrew method is fairly simple and is automatically configured to support Python and Perl so that scripts can be loaded from ~/.config/hexchat/addons. First install Homebrew, then do the following:

```
brew install homebrew/gui/hexchat --with-python
# or to build directly from git:
# brew install homebrew/gui/hexchat --HEAD --with-python

# then launch it
/usr/local/bin/hexchat
```

### JHBuild

JHBuild is the alternative method. It uses the quartz backend as well as a more native theme. It can be more hands on and complex than the automated Homebrew method.

1. Follow the instructions on Gnome's site for Building on OSX

**Note:** Some of these builds may fail and require dropping to a shell to fix them. Most are trivial but the packages are always changing so you are on your own.

Once everything is set up we can build hexchat:

```
git clone https://github.com/hexchat/hexchat.git && cd hexchat
./autogen.sh --prefix=$JHBUILD_PREFIX
make && make install
hexchat
```

If you want to make the HexChat.app bundle it will take some more work.

1. Follow the instructions on Gnome's site for Bundling on OSX
2. Build and install these packages
    - Modified gtk-quartz-engine
    - xamarin-gtk-theme
    - enchant with these patches

```
cd osx && ./makebundle.sh
```

### 7.2.2 Plugin Interface

#### Introduction

Plugins for HexChat are written in C. The interface aims to keep 100% binary compatability. This means that if you upgrade HexChat, you will not need to recompile your plugins, they'll continue to work. The interface doesn't depend on any structures and offsets, so compiler versions shouldn't have an impact either. The only real requirement of a HexChat plugin is that it define an *hexchat_plugin_init* symbol. This is your entry point function, see the example below. You should make all your global variables and functions *static*, so that a symbol is not exported. There is no harm in exporting these symbols, but they are not necessary and only pollute the name-space. Plugins are compiled as shared objects (.so files), for example:

Most UNIX systems:

> gcc -Wl –export-dynamic -Wall -O1 -shared -fPIC myplugin.c -o myplugin.so

OS X:

> gcc -no-cpp-precomp -g -O2 -Wall -bundle -flat_namespace -undefined suppress -o myplugin.so myplugin.c

See the Windows section on how to compile a plugin using Visual Studio.

All strings passed to and from plugins are encoded in UTF-8, regardless of locale.

#### Sample plugin

This simple plugin auto-ops anyone who joins a channel you're in. It also adds a new command */AUTOOPTOGGLE*, which can be used to turn the feature ON or OFF. Every HexChat plugin must define an *hexchat_plugin_init* function, this is the normal entry point. *hexchat_plugin_deinit* is optional.

```
#include "hexchat-plugin.h"

#define PNAME "AutoOp"
#define PDESC "Auto Ops anyone that joins"
#define PVERSION "0.1"

static hexchat_plugin *ph;      /* plugin handle */
static int enable = 1;

static int
join_cb (char *word[], void *userdata)
{
        if (enable)
        {
                /* Op ANYONE who joins */
                hexchat_commandf (ph, "OP %s", word[1]);
        }
        /* word[1] is the nickname, as in the Settings->Text Events window in HexChat */

        return HEXCHAT_EAT_NONE;        /* don't eat this event, HexChat needs to see it! */
}

static int
autooptoggle_cb (char *word[], char *word_eol[], void *userdata)
{
        if (!enable)
        {
                enable = 1;
                hexchat_print (ph, "AutoOping now enabled!\n");
        }
        else
        {
                enable = 0;
                hexchat_print (ph, "AutoOping now disabled!\n");
        }

        return HEXCHAT_EAT_ALL;     /* eat this command so HexChat and other plugins can't process it
}

void
hexchat_plugin_get_info (char **name, char **desc, char **version, void **reserved)
{
        *name = PNAME;
        *desc = PDESC;
        *version = PVERSION;
}

int
hexchat_plugin_init (hexchat_plugin *plugin_handle, char **plugin_name, char **plugin_desc, char **pl
{
        /* we need to save this for use with any hexchat_* functions */
        ph = plugin_handle;

        /* tell HexChat our info */
        *plugin_name = PNAME;
        *plugin_desc = PDESC;
        *plugin_version = PVERSION;

        hexchat_hook_command (ph, "AutoOpToggle", HEXCHAT_PRI_NORM, autooptoggle_cb, "Usage: AUTOOPTO
        hexchat_hook_print (ph, "Join", HEXCHAT_PRI_NORM, join_cb, 0);

        hexchat_print (ph, "AutoOpPlugin loaded successfully!\n");

        return 1;       /* return 1 for success */
}
```

## What's *word* and *word_eol*?

They are arrays of strings. They contain the parameters the user entered for the particular command. For example, if you executed:

```
/command NICK hi there
```

- word[1] is command
- word[2] is NICK
- word[3] is hi
- word[4] is there
- word_eol[1] is command NICK hi there
- word_eol[2] is NICK hi there
- word_eol[3] is hi there

- word_eol[4] is there

These arrays are simply provided for your convenience. You are **not** allowed to alter them. Both arrays are limited to 32 elements (index 31). *word[0]* and *word_eol[0]* are reserved and should not be read.

### Lists and Fields

Lists of information (DCCs, Channels, User list, etc.) can be retreived with *hexchat_list_get*. All fields are **read only** and must be copied if needed for a long time after calling *hexchat_list_str*. The types of lists and fields available are:

| "channels" | list of channels, querys and their servers | |
|---|---|---|
| Name | Description | Type |
| channel | Channel or query name | string |
| channelkey | Channels key or NULL (2.9.6+) | string |
| chanmodes | Channel modes e.g. "beI,k,l" (2.12.2+) | string |
| chantypes | Channel types e.g. "#!&" | string |
| context | (hexchat_context *) pointer. Can be used with hexchat_set_context | string |
| flags | • 1 = Connected<br>• 2 = Connecting<br>• 4 = Marked away<br>• 8 = End of MOTD<br>• 16 = Has WHOX<br>• 32 = Has IDMSG<br>• 64 = Hide Join/Parts<br>• 128 = Hide Join/Parts unset<br>• 256 = Beep on Message<br>• 512 = Blink Tray<br>• 1024 = Blink Taskbar<br>• 2048 = Logging<br>• 4096 = Logging unset<br>• 8192 = Scrollback<br>• 16384 = Scrollback unset<br>• 32768 = Strip colors<br>• 65536 = Strip colors unset | int |
| id | Unique server ID | int |
| lag | Lag in milliseconds | int |
| maxmodes | Maximum modes per line | int |
| network | Name of network | string |
| nickprefixes | Nickname prefixes e.g. "@+" | string |
| nickmodes | Nickname mode chars e.g. "ov" | string |
| queue | Number of bytes in the send-queue | int |
| server | Server name to which this channel belongs | string |
| type | • 1 = Server<br>• 2 = Channel<br>• 3 = Dialog<br>• 4 = Notice<br>• 5 = SNotice | int |
| users | Number of users in this channel | int |

| "dcc" | list of DCC file transfers | |
|---|---|---|
| Name | Description | Type |
| address32 | Address of the remote user (ipv4 address) | int |
| cps | Bytes per second (speed) | int |
| destfile | Destination full pathname | string |
| file | File name | string |
| nick | Nickname of person who the file is from/to | string |
| port | TCP port number | int |
| pos | Bytes sent/received | int |
| poshigh | Bytes sent/received, high order 32 bits | int |
| resume | Point at which this file was resumed (or zero if it was not resumed) | int |
| resumehigh | Point at which this file was resumed, high order 32 bits | int |
| size | File size in bytes, low order 32 bits (cast it to unsigned) | int |
| sizehigh | File size in bytes, high order 32 bits | int |
| status | • 0 = Queued<br>• 1 = Active<br>• 2 = Failed<br>• 3 = Done<br>• 4 = Connecting<br>• 5 = Aborted | int |
| type | • 0 = Send<br>• 1 = Recieve<br>• 1 = ChatRecv<br>• 1 = ChatSend | int |

| "ignore" | current ignore list | |
|---|---|---|
| Name | Description | Type |
| mask | Ignore mask. .e.g. *!*@*.aol.com | string |
| flags | • 0 = Private<br>• 1 = Notice<br>• 2 = Channel<br>• 3 = CTCP<br>• 4 = Invite<br>• 5 = Unignore<br>• 6 = NoSave<br>• 7 = DCC | int |

| "notify" | list of people on notify | |
|---|---|---|
| Name | Description | Type |
| networks | Networks to which this nick applies. Comma separated. May be NULL. | string |
| nick | Nickname | string |
| flags | Bit field of flags. 0=Is online. | int |
| on | Time when user came online. | time_t |
| off | Time when user went offline. | time_t |
| seen | Time when user the user was last verified still online. | time_t |

Fields are only valid for the context when hexchat_list_get() was called (i.e. you get information about the user ON THAT ONE SERVER ONLY). You may cycle through the "channels" list to find notify information for every server.

| "users" | list of users in the current channel | |
|---|---|---|
| Name | Description | Type |
| account | Account name or NULL (2.9.6+) | string |
| away | Away status (boolean) | int |
| lasttalk | Last time the user was seen talking | time_t |
| nick | Nick name | string |
| host | Host name in the form: user@host (or NULL if not known). | string |
| prefix | Prefix character, .e.g: @ or +. Points to a single char. | string |
| realname | Real name or NULL | string |
| selected | Selected status in the user list, only works for retrieving the user list of the focused tab | int |

Example:

```
list = hexchat_list_get (ph, "dcc");

if (list)
{
        hexchat_print (ph, "--- DCC LIST ------------------\nFile  To/From   KB/s   Position\n");

        while (hexchat_list_next (ph, list))
        {
                hexchat_printf (ph, "%6s %10s %.2f  %d\n",
                        hexchat_list_str (ph, list, "file"),
                        hexchat_list_str (ph, list, "nick"),
                        hexchat_list_int (ph, list, "cps") / 1024,
                        hexchat_list_int (ph, list, "pos"));
        }

        hexchat_list_free (ph, list);
}
```

### Plugins on Windows (Win32)

All you need is Visual Studio setup as explained in Building. Your best bet is to use an existing plugin (such as the currently unused SASL plugin) in the HexChat solution as a starting point. You should have the following files:

- hexchat-plugin.h - main plugin header
- plugin.c - Your plugin, you need to write this one :)
- plugin.def - A simple text file containing the following:

```
EXPORTS
    hexchat_plugin_init
    hexchat_plugin_deinit
    hexchat_plugin_get_info
```

Leave out *hexchat_plugin_deinit* if you don't intend to define that function. Then compile your plugin in Visual Studio as usual.

**Caveat:** plugins compiled on Win32 **must** have a global variable called *ph*, which is the *plugin_handle*, much like in the sample plugin above.

## Controlling the GUI

A simple way to perform basic GUI functions is to use the */GUI* command. You can execute this command through the input box, or by calling *hexchat_command (ph, "GUI .....");*.

- **GUI ATTACH:** Same function as "Attach Window" in the HexChat menu.
- **GUI DETACH:** Same function as "Detach Tab" in the HexChat menu.
- **GUI APPLY:** Similar to clicking OK in the settings window. Execute this after /SET to activate GUI changes.
- **GUI COLOR *n*:** Change the tab color of the current context, where *n* is a number from 0 to 3.
- **GUI FOCUS:** Focus the current window or tab.
- **GUI FLASH:** Flash the taskbar button. It will flash only if the window isn't focused and will stop when it is focused by the user.
- **GUI HIDE:** Hide the main HexChat window completely.
- **GUI ICONIFY:** Iconify (minimize to taskbar) the current HexChat window.
- **GUI MSGBOX *text*:** Displays a asynchronous message box with your text.
- **GUI SHOW:** Show the main HexChat window (if currently hidden).

You can add your own items to the menu bar. The menu command has this syntax:

```
MENU [-eX] [-i<ICONFILE>] [-k<mod>,<key>] [-m] [-pX] [-rX,group] [-tX] {ADD|DEL} <path> [command] [un
```

For example:

```
MENU -p5 ADD FServe
MENU ADD "FServe/Show File List" "fs list"
MENU ADD FServe/-
MENU -k4,101 -t1 ADD "FServe/Enabled" "fs on" "fs off"
MENU -e0 ADD "FServe/Do Something" "fs action"
```

In the example above, it would be recommended to execute *MENU DEL FServe* inside your *hexchat_plugin_deinit* function. The special item with name "-" will add a separator line.

Parameters and flags:

- **-eX:** Set enable flag to X. -e0 for disable, -e1 for enable. This lets you create a disabled (shaded) item.
- **-iFILE:** Use an icon filename FILE. Not supported for toggles or radio items.
- **-k<mod>,<key>:** Specify a keyboard shortcut. "mod" is the modifier which is a bitwise OR of: 1-SHIFT 4-CTRL 8-ALT in decimal. "key" is the key value in decimal, e.g. -k5,101 would specify SHIFT-CTRL-E.
- **-m:** Specify that this label should be treated as Pango Markup language. Since forward slash ("/") is already used in menu paths, you should replace closing tags with an ASCII 003 instead e.g.: hexchat_command (ph, "MENU -m ADD "<b>Bold Menu<03b>"");
- **-pX:** Specify a menu item's position number. e.g. -p5 will cause the item to be inserted in the 5th place. If the position is a negative number, it will be used as an offset from the bottom/right-most item.

- **-rX,group:** Specify a radio menu item, with initial state X and a group name. The group name should be the exact label of another menu item (without the path) that this item will be grouped with. For radio items, only a select command will be executed (no unselect command).
- **-tX:** Specify a toggle menu item with an initial state. -t0 for an "unticked" item and -t1 for a "ticked" item.

If you want to change an item's toggle state or enabled flag, just *ADD* an item with exactly the same name and command and specify the *-tX -eX* parameters you need.

It's also possible to add items to HexChat's existing menus, for example:

```
MENU ADD "Settings/Sub Menu"
MENU -t0 ADD "Settings/Sub Menu/My Setting" myseton mysetoff
```

However, internal names and layouts of HexChat's menu may change in the future, so use at own risk.

Here is an example of Radio items:

```
MENU ADD "Language"
MENU -r1,"English" ADD "Language/English" cmd1
MENU -r0,"English" ADD "Language/Spanish" cmd2
MENU -r0,"English" ADD "Language/German" cmd3
```

You can also change menus other than the main one (i.e popup menus). Currently they are:

| Root Name | Menu |
|---|---|
| $TAB | Tab menu (right click a channel/query tab or treeview row) |
| $TRAY | System Tray menu |
| $URL | URL link menu |
| $NICK | Userlist nick-name popup menu |
| $CHAN | Menu when clicking a channel in the text area |

Example:

```
MENU -p0 ADD "$TAB/Cycle Channel" cycle
```

You can manipulate HexChat's system tray icon using the */TRAY* command:

```
Usage:
TRAY -f <timeout> <file1> [<file2>] Flash tray between two icons. Leave off file2 to use default Hex
TRAY -f <filename>                  Set tray to a fixed icon.
TRAY -i <number>                    Flash tray with an internal icon.
TRAY -t <text>                      Set the tray tooltip.
TRAY -b <title> <text>              Set the tray balloon.
```

Icon numbers:

- 2: Message
- 5: Highlight
- 8: Private
- 11:File

For tray balloons on Linux, you'll need libnotify.

Filenames can be *ICO* or *PNG* format. *PNG* format is supported on Linux/BSD and Windows XP. Set a timeout of -1 to use HexChat's default.

### Handling UTF-8/Unicode strings

The HexChat plugin API specifies that strings passed to and from HexChat must be encoded in UTF-8.

What does this mean for the plugin programmer? You just have to be a little careful when passing strings obtained from IRC to system calls. For example, if you're writing a file-server bot, someone might message you a filename. Can you pass this filename directly to open()? Maybe! If you're lazy... The correct thing to do is to convert the string to "system locale encoding", otherwise your plugin will fail on non-ascii characters.

Here are examples on how to do this conversion on Unix and Windows. In this example, someone will CTCP you the message "SHOWFILE <filename>".

```
static int
ctcp_cb (char *word[], char *word_eol[], void *userdata)
{
        if(strcmp(word[1], "SHOWFILE") == 0)
        {
                get_file_name (nick, word[2]);
        }

        return HEXCHAT_EAT_HEXCHAT;
}

static void
get_file_name (char *nick, char *fname)
{
        char buf[256];
        FILE *fp;

        /* the fname is in UTF-8, because it came from the HexChat API */

#ifdef _WIN32

        wchar_t wide_name[MAX_PATH];

        /* convert UTF-8 to WIDECHARs (aka UTF-16LE) */
        if (MultiByteToWideChar (CP_UTF8, 0, fname, -1, wide_name, MAX_PATH) < 1)
        {
                return;
        }

        /* now we have WIDECHARs, so we can _wopen() or CreateFileW(). */
        /* _wfopen actually requires NT4, Win2000, XP or newer. */
        fp = _wfopen (wide_name, "r");

#else

        char *loc_name;

        /* convert UTF-8 to System Encoding */
        loc_name = g_filename_from_utf8 (fname, -1, 0, 0, 0);
        if(!loc_name)
        {
                return;
        }

        /* now open using the system's encoding */
        fp = fopen (loc_name, "r");
        g_free (loc_name);
#endif

        if (fp)
        {
                while (fgets (buf, sizeof (buf), fp))
                {
                        /* send every line to the user that requested it */
                        hexchat_commandf (ph, "QUOTE NOTICE %s :%s", nick, buf);
                }
                fclose (fp);
        }
}
```

## Types and Constants

**hexchat_plugin**
**hexchat_list**
**hexchat_hook**
**hexchat_context**
**hexchat_event_attrs**

**HEXCHAT_PRI_HIGHEST**
**HEXCHAT_PRI_HIGH**
**HEXCHAT_PRI_NORM**
**HEXCHAT_PRI_LOW**
**HEXCHAT_PRI_LOWEST**

**HEXCHAT_EAT_NONE**
**HEXCHAT_EAT_XCHAT**
**HEXCHAT_EAT_PLUGIN**
**HEXCHAT_EAT_ALL**

**HEXCHAT_FD_READ**
**HEXCHAT_FD_WRITE**
**HEXCHAT_FD_EXCEPTION**
**HEXCHAT_FD_NOTSOCKET**

## Functions

### General Functions

void **hexchat_command** (*hexchat_plugin* **ph*, const char **command*)
Executes a command as if it were typed in HexChat's input box.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **command** – Command to execute, without the forward slash "/".

void **hexchat_commandf** (*hexchat_plugin* **ph*, const char **format*, ...)
Executes a command as if it were typed in HexChat's input box and provides string formatting like *printf*.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).

- **format** – The format string.

void **hexchat_print** (*hexchat_plugin *ph*, const char **text*)

Prints some text to the current tab/window.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **text** – Text to print. May contain mIRC color codes.

void **hexchat_printf** (*hexchat_plugin *ph*, const char **format*, ...)

Prints some text to the current tab/window and provides formatting like *printf*.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **format** – The format string.

int **hexchat_emit_print** (*hexchat_plugin *ph*, const char **event_name*, ...)

Generates a print event. This can be any event found in the *Settings* → *Text Events* window. The vararg parameter list **must** always be NULL terminated. Special care should be taken when calling this function inside a print callback (from `hexchat_hook_print()`), as not to cause endless recursion.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **event_name** – Text event to print.

**Returns** 0 on Failure, 1 on Success

**Example:**

```
hexchat_emit_print (ph, "Channel Message", "John", "Hi there", "@", NULL);
```

int **hexchat_emit_print_attrs** (*hexchat_plugin *ph*, *hexchat_event_attrs *attrs*, const char **event_name*, ...)

Generates a print event. This is the same as `hexchat_emit_print()` but it passes an `hexchat_event_attrs *` to hexchat with the print attributes.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **attrs** – Print attributes. This should be obtained with `hexchat_event_attrs_create()` and freed with `hexchat_event_attrs_free()`.
- **event_name** – Text event to print.

**Returns** 0 on Failure, 1 on Success

New in version 2.9.6.

**Example:**

```
hexchat_event_attrs *attrs;

attrs = hexchat_event_attrs_create (ph);
attrs->server_time_utc = 1342224702;
hexchat_emit_print (ph, attrs, "Channel Message", "John", "Hi there", "@", NULL);
hexchat_event_attrs_free (ph, attrs);
```

void **hexchat_send_modes** (*hexchat_plugin *ph*, const char **targets[]*, int *ntargets*, int *modes_per_line*, char *sign*, char *mode*)

Sends a number of channel mode changes to the current channel. For example, you can Op a whole group of people in one go. It may send multiple MODE lines if the request doesn't fit on one. Pass 0 for *modes_per_line* to use the current server's maximum possible. This function should only be called while in a channel context.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **targets** – Array of targets (strings). The names of people whom the action will be performed on.
- **ntargets** – Number of elements in the array given.
- **modes_per_line** – Maximum modes to send per line.
- **sign** – Mode sign, '-' or '+'.
- **mode** – Mode char, e.g. 'o' for Ops.

**Example:** (Ops the three names given)

```
const char *names_to_Op[] = {"John", "Jack", "Jill"};
hexchat_send_modes (ph, names_to_Op, 3, 0, '+', 'o');
```

int **hexchat_nickcmp** (*hexchat_plugin *ph*, const char **s1*, const char **s2*)

Performs a nick name comparision, based on the current server connection. This might be an RFC1459 compliant string compare, or plain ascii (in the case of DALNet). Use this to compare channels and nicknames. The function works the same way as *strcasecmp*.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **s1** – String to compare.
- **s2** – String to compare *s1* to.

**Quote from RFC1459:** >Because of IRC's scandanavian origin, the characters {}| are considered to be the lower case equivalents of the characters [], respectively. This is a critical issue when determining the equivalence of two nicknames.

**Returns** An integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

char* **hexchat_strip** (*hexchat_plugin *ph*, const char **str*, int *len*, int *flags*)

Strips mIRC color codes and/or text attributes (bold, underlined etc) from the given string and returns a newly allocated string.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **str** – String to strip.
- **len** – Length of the string (or -1 for NULL terminated).
- **flags** – Bit-field of flags: - 0: Strip mIRC colors. - 1: Strip text attributes.

**Returns** A newly allocated string or NULL for failure. You must free this string with `hexchat_free()`.

**Example:**

```
    {
        char *new_text;

        /* strip both colors and attributes by using the 0 and 1 bits (1 BITWISE-OR 2) */
        new_text = hexchat_strip (ph, "\00312Blue\003 \002Bold!\002", -1, 1 | 2);

        if (new_text)
        {
                /* new_text should now contain only "Blue Bold!" */
                hexchat_printf (ph, "%s\n", new_text);
                hexchat_free (ph, new_text);
        }
    }
```

void **hexchat_free** (*hexchat_plugin *ph*, void **ptr*)

Frees a string returned by **hexchat_*** functions. Currently only used to free strings from `hexchat_strip()`.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **ptr** – Pointer to free.

*hexchat_event_attrs* ***hexchat_event_attrs_create** (*hexchat_plugin *ph*)

Allocates a new `hexchat_event_attrs`. The attributes are initially marked as unused.

**Returns** A pointer to the allocated `hexchat_event_attrs`. Should be freed by `hexchat_event_attrs_free()`.

New in version 2.9.6.

void **hexchat_event_attrs_free** (*hexchat_plugin *ph*, *hexchat_event_attrs *attrs*)

Frees an `hexchat_event_attrs`.

**Parameters**

- **attrs** – Attributes previously allocated by `hexchat_event_attrs_create()`.

New in version 2.9.6.

### Getting Information

const char* **hexchat_get_info** (*hexchat_plugin *ph*, const char **id*)

Returns information based on your current context.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **id** – ID of the information you want. List of ID's(case sensitive):
  - **away:** away reason or NULL if you are not away.
  - **channel:** current channel name.
  - **charset:** character-set used in the current context.
  - **configdir:** HexChat config directory, e.g.: `/home/user/.config/hexchat`. This string is encoded in UTF-8.
  - **event_text <name>:** text event format string for *name*.
  - **gtkwin_ptr:** (GtkWindow *).

- **host:** real hostname of the server you connected to.
- **inputbox:** the input-box contents, what the user has typed.
- **libdirfs:** library directory. e.g. /usr/lib/hexchat. The same directory used for auto-loading plugins. This string isn't necessarily UTF-8, but local file system encoding.
- **modes:** channel modes, if known, or NULL.
- **network:** current network name or NULL.
- **nick:** your current nick name.
- **nickserv:** nickserv password for this network or NULL.
- **server:** current server name (what the server claims to be). NULL if you are not connected.
- **topic:** current channel topic.
- **version:** HexChat version number.
- **win_ptr:** native window pointer. Unix: (GtkWindow *) Win32: HWND.
- **win_status:** window status: "active", "hidden" or "normal".

**Returns** A string of the requested information, or NULL. This string must not be freed and must be copied if needed after the call to `hexchat_get_info()`.

int **`hexchat_get_prefs`** (*hexchat_plugin* **ph*, const char **name*, const char ***string*, int **integer*)

Provides HexChat's setting information (that which is available through the **`/SET`** command). A few extra bits of information are available that don't appear in the **`/SET`** list, currently they are:

- **state_cursor:** Current input box cursor position (characters, not bytes).
- **id:** Unique server id

**Parameters**

- **`ph`** – Plugin handle (as given to *hexchat_plugin_init*).
- **`name`** – Setting name required.
- **`string`** – Pointer-pointer which to set.
- **`integer`** – Pointer to an integer to set, if setting is a boolean or integer type.

**Returns**

- 0: Failed.
- 1: Returned a string.
- 2: Returned an integer.
- 3: Returned a boolean.

**Example:**

```
{
    int i;
    const char *str;

    if (hexchat_get_prefs (ph, "irc_nick1", &amp;str, &amp;i) == 1)
    {
            hexchat_printf (ph, "Current nickname setting: %s\n", str);
```

```
        }
    }
```

*hexchat_list** **hexchat_list_get** (*hexchat_plugin *ph*, const char **name*)

Retreives lists of information.

**Parameters**

- **name** – Name from the List and Fields Table

**Returns** hexchat_list to be used by the following functions.

const char* const* **hexchat_list_fields** (*hexchat_plugin *ph*, const char **name*)

Lists fields in a given list.

**Parameters**

- **name** – Name from the List and Fields Table

int **hexchat_list_next** (*hexchat_plugin *ph*, *hexchat_list *xlist*)

Selects the next list item in a list.

**Parameters**

- **xlist** – `hexchat_list` returned by `hexchat_list_get()`

const char* **hexchat_list_str** (*hexchat_plugin *ph*, *hexchat_list *xlist*, const char **name*)

Gets a string field from a list.

**Parameters**

- **name** – Name from the List and Fields Table
- **xlist** – `hexchat_list` returned by `hexchat_list_get()`

int **hexchat_list_int** (*hexchat_plugin *ph*, *hexchat_list *xlist*, const char **name*)

Gets a int field from a list.

**Parameters**

- **name** – Name from the List and Fields Table
- **xlist** – `hexchat_list` returned by `hexchat_list_get()`

time_t **hexchat_list_time** (*hexchat_plugin *ph*, *hexchat_list *xlist*, const char **name*)

Gets a time field from a list.

**Parameters**

- **name** – Name from the List and Fields Table
- **xlist** – `hexchat_list` returned by `hexchat_list_get()`

void **hexchat_list_free** (*hexchat_plugin *ph*, *hexchat_list *xlist*)

Frees a list.

**Parameters**

- **xlist** – `hexchat_list` returned by `hexchat_list_get()`

### Hook Functions

*hexchat_hook** **hexchat_hook_command**(*hexchat_plugin *ph*, const char **name*, int *pri*, int (**callb*)(char *word[], char *word_eol[], void *user_data), const char **help_text*, void **userdata*)

Adds a new **/command**. This allows your program to handle commands entered at the input box. To capture text without a "/" at the start (non-commands), you may hook a special name of "". i.e **hexchat_hook_command(ph, "", ...)**.

Commands hooked that begin with a period ('.') will be hidden in **/HELP** and **/HELP -l**.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **name** – Name of the command (without the forward slash).
- **pri** – Priority of this command. Use *HEXCHAT_PRI_NORM*.
- **callb** – Callback function. This will be called when the user executes the given command name.
- **help_text** – String of text to display when the user executes **/HELP** for this command. May be NULL if you're lazy.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to *hexchat_unhook()*.

**Example:**

```
static int
onotice_cb (char *word[], char *word_eol[], void *userdata)
{
    if (word_eol[2][0] == 0)
    {
            hexchat_printf (ph, "Second arg must be the message!\n");
            return HEXCHAT_EAT_ALL;
    }

    hexchat_commandf (ph, "NOTICE @%s :%s", hexchat_get_info (ph, "channel"), word_eol[2]);
    return HEXCHAT_EAT_ALL;
}

hexchat_hook_command (ph, "ONOTICE", HEXCHAT_PRI_NORM, onotice_cb, "Usage: ONOTICE <message> Sen
```

*hexchat_hook** **hexchat_hook_fd**(*hexchat_plugin *ph*, int *fd*, int *flags*, int (**callb*)(int fd, int flags, void *user_data), void **userdata*)

Hooks a socket or file descriptor. WIN32: Passing a pipe from MSVCR71, MSVCR80 or other variations is not supported at this time.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init ()*).
- **fd** – The file descriptor or socket.
- **flags** – One or more of HEXCHAT_FD_* constants tells HexChat that the provided *fd* is not a socket, but an "MSVCRT.DLL" pipe.
- **callb** – Callback function. This will be called when the socket is available for reading/writing or exception (depending on your chosen *flags*)
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to *hexchat_unhook()*.

*hexchat_hook** **hexchat_hook_print** (*hexchat_plugin* **ph*, const char **name*, int *pri*, int (**callb*)(char *word[], void *user_data), void **userdata*)

Registers a function to trap any print events. The event names may be any available in the *Settings* → *Text Events* window. There are also some extra "special" events you may hook using this function. Currently they are:

- "Open Context": Called when a new hexchat_context is created.
- "Close Context": Called when a hexchat_context pointer is closed.
- "Focus Tab": Called when a tab is brought to front.
- **"Focus Window": Called a toplevel window is focused, or the main** tab-window is focused by the window manager.
- "DCC Chat Text": Called when some text from a DCC Chat arrives. It provides these elements in the *word[]* array:

```
word[1] Address
word[2] Port
word[3] Nick
word[4] The Message
```

- "Key Press": Called when some keys are pressed in the input box. It provides these elements in the *word[]* array:

```
word[1] Key Value
word[2] State Bitfield (shift, capslock, alt)
word[3] String version of the key
word[4] Length of the string (may be 0 for unprintable keys)
```

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **name** – Name of the print event (as in *Text Events* window).
- **pri** – Priority of this command. Use *HEXCHAT_PRI_NORM*.
- **callb** – Callback function. This will be called when this event name is printed.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to *hexchat_unhook()*.

**Example:**

```
static int
youpart_cb (char *word[], void *userdata)
{
    hexchat_printf (ph, "You have left channel %s\n", word[3]);
    return HEXCHAT_EAT_HEXCHAT;      /* don't let HexChat do its normal printing */
}

hexchat_hook_print (ph, "You Part", HEXCHAT_PRI_NORM, youpart_cb, NULL);
```

*hexchat_hook** **hexchat_hook_print_attrs**(*hexchat_plugin *ph*, const char **name*, int *pri*, int (**callb*)(char *word[], *hexchat_event_attrs* *attrs, void *user_data), void **userdata*)

Registers a function to trap any print events. This is the same as `hexchat_hook_print()` but the callback receives an `hexchat_event_attrs *` with attributes related to the print event.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **name** – Name of the print event (as in *Text Events* window).
- **pri** – Priority of this command. Use *HEXCHAT_PRI_NORM*.
- **callb** – Callback function. This will be called when this event name is printed.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to `hexchat_unhook()`.

New in version 2.9.6.

*hexchat_hook** **hexchat_hook_server**(*hexchat_plugin *ph*, const char **name*, int *pri*, int (**callb*)(char *word[], char *word_eol[], void *user_data), void **userdata*)

Registers a function to be called when a certain server event occurs. You can use this to trap *PRIVMSG*, *NOTICE*, *PART*, a server numeric, etc. If you want to hook every line that comes from the IRC server, you may use the special name of *RAW LINE*.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **name** – Name of the server event.
- **pri** – Priority of this command. Use *HEXCHAT_PRI_NORM*.
- **callb** – Callback function. This will be called when this event is received from the server.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to `hexchat_unhook()`.

**Example:**

```
static int
kick_cb (char *word[], char *word_eol[], void *userdata)
{
    hexchat_printf (ph, "%s was kicked from %s (reason=%s)\n", word[4], word[3], word_eol[5]);
    return HEXCHAT_EAT_NONE;        /* don't eat this event, let other plugins and HexChat see i
}

hexchat_hook_server (ph, "KICK", HEXCHAT_PRI_NORM, kick_cb, NULL);
```

*hexchat_hook** **hexchat_hook_server_attrs**(*hexchat_plugin *ph*, const char **name*, int *pri*, int (**callb*)(char *word[], char *word_eol[], *hexchat_event_attrs* *attrs, void *user_data), void **userdata*)

Registers a function to be called when a certain server event occurs. This is the same as `hexchat_hook_server()` but the callback receives an `hexchat_event_attrs *` with attributes related to the server event.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **name** – Name of the server event.

- **pri** – Priority of this command. Use *HEXCHAT_PRI_NORM*.
- **callb** – Callback function. This will be called when this event is received from the server.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to *hexchat_unhook()*.

New in version 2.9.6.

*hexchat_hook* ***hexchat_hook_timer**(*hexchat_plugin* **ph*, int *timeout*, int (**callb*)(void *user_data), void **userdata*)

Registers a function to be called every "timeout" milliseconds.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **timeout** – Timeout in milliseconds (1000 is 1 second).
- **callb** – Callback function. This will be called every "timeout" milliseconds.
- **userdata** – Pointer passed to the callback function.

**Returns** Pointer to the hook. Can be passed to *hexchat_unhook()*.

**Example:**

```
static hexchat_hook *myhook;

static int
stop_cb (char *word[], char *word_eol[], void *userdata)
{
    if (myhook != NULL)
    {
            hexchat_unhook (ph, myhook);
            myhook = NULL;
            hexchat_print (ph, "Timeout removed!\n");
    }

    return HEXCHAT_EAT_ALL;
}

static int
timeout_cb (void *userdata)
{
    hexchat_print (ph, "Annoying message every 5 seconds! Type /STOP to stop it.\n");
    return 1;       /* return 1 to keep the timeout going */
}

myhook = hexchat_hook_timer (ph, 5000, timeout_cb, NULL);
hexchat_hook_command (ph, "STOP", HEXCHAT_PRI_NORM, stop_cb, NULL, NULL);
```

void* **hexchat_unhook**(*hexchat_plugin* **ph*, *hexchat_hook* **hook*)

Unhooks any hook registered with **hexchat_hook_print/server/timer/command**. When plugins are unloaded, all of its hooks are automatically removed, so you don't need to call this within your *hexchat_plugin_deinit* function.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **hook** – Pointer to the hook, as returned by **hexchat_hook_***.

**Returns** The userdata you originally gave to **hexchat_hook_***.

### Context Functions

*hexchat_context** **hexchat_find_context** (*hexchat_plugin* **ph*, const char **servname*, const char **channel*)

Finds a context based on a channel and servername. If *servname* is NULL, it finds any channel (or query) by the given name. If *channel* is NULL, it finds the front-most tab/window of the given *servname*. If NULL is given for both arguments, the currently focused tab/window will be returned.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **servname** – Server name or NULL.
- **channel** – Channel name or NULL.

**Returns** Context pointer (for use with `hexchat_set_context()`) or NULL.

*hexchat_context** **hexchat_get_context** (*hexchat_plugin* **ph*)

Returns the current context for your plugin. You can use this later with `hexchat_set_context()`.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).

**Returns** Context pointer (for use with `hexchat_set_context()`).

int **hexchat_set_context** (*hexchat_plugin* **ph*, *hexchat_context* **ctx*)

Changes your current context to the one given.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **ctx** – Context to change to (obtained with `hexchat_get_context()` or `hexchat_find_context()`).

**Returns**

- 1: Success.
- 0: Failure.

### Plugin Preferences

int **hexchat_pluginpref_set_str** (*hexchat_plugin* **ph*, const char **var*, const char **value*)

Saves a plugin-specific setting with string value to a plugin-specific config file.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **var** – Name of the setting to save.
- **value** – String value of the the setting.

**Returns**

- 1: Success.
- 0: Failure.

**Example:**

```
int
hexchat_plugin_init (hexchat_plugin *plugin_handle, char **plugin_name, char **plugin_desc, char
{
    ph = plugin_handle;
    *plugin_name = "Tester Thingie";
    *plugin_desc = "Testing stuff";
    *plugin_version = "1.0";

    hexchat_pluginpref_set_str (ph, "myvar1", "I want to save this string!");
    hexchat_pluginpref_set_str (ph, "myvar2", "This is important, too.");

    return 1;       /* return 1 for success */
}
```

In the example above, the settings will be saved to the plugin_tester_thingie.conf file, and its content will be: >myvar1 = I want to save this string! myvar2 = This is important, too.

You should never need to edit this file manually.

int **hexchat_pluginpref_get_str** (*hexchat_plugin *ph*, const char **var*, char **dest*)
Loads a plugin-specific setting with string value from a plugin-specific config file.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **var** – Name of the setting to load.
- **dest** – Array to save the loaded setting's string value to.

**Returns**

- 1: Success.
- 0: Failure.

int **hexchat_pluginpref_set_int** (*hexchat_plugin *ph*, const char **var*, int *value*)
Saves a plugin-specific setting with decimal value to a plugin-specific config file.

**Parameters**

- **ph** – Plugin handle (as given to *hexchat_plugin_init*).
- **var** – Name of the setting to save.
- **value** – Decimal value of the the setting.

**Returns**

- 1: Success.
- 0: Failure.

**Example:**

```
static int
saveint_cb (char *word[], char *word_eol[], void *user_data)
{
    int buffer = atoi (word[2]);

    if (buffer > 0 && buffer < INT_MAX)
    {
            if (hexchat_pluginpref_set_int (ph, "myint1", buffer))
            {
                    hexchat_printf (ph, "Setting successfully saved!\n");
```

```
                }
                else
                {
                        hexchat_printf (ph, "Error while saving!\n");
                }
        }
        else
        {
                hexchat_printf (ph, "Invalid input!\n");
        }

        return HEXCHAT_EAT_HEXCHAT;
}
```

You only need such complex checks if you're saving user input, which can be non-numeric.

int **`hexchat_pluginpref_get_int`** (*hexchat_plugin *ph*, const char **var*)

Loads a plugin-specific setting with decimal value from a plugin-specific config file.

**Parameters**

- **`ph`** – Plugin handle (as given to *hexchat_plugin_init*).
- **`var`** – Name of the setting to load.

**Returns** The decimal value of the requested setting upon success, -1 for failure.

int **`hexchat_pluginpref_delete`** (*hexchat_plugin *ph*, const char **var*)

Deletes a plugin-specific setting from a plugin-specific config file.

**Parameters**

- **`ph`** – Plugin handle (as given to *hexchat_plugin_init*).
- **`var`** – Name of the setting to delete.

**Returns**

- 1: Success.
- 0: Failure.

If the given setting didn't exist, it also returns 1, so 1 only indicates that the setting won't exist after the call.

int **`hexchat_pluginpref_list`** (*hexchat_plugin *ph*, char **dest*)

Builds a comma-separated list of the currently saved settings from a plugin-specific config file.

**Parameters**

- **`ph`** – Plugin handle (as given to *hexchat_plugin_init*).
- **`dest`** – Array of size 4096 to save the list to.

**Returns**

- 1: Success.
- 0: Failure (nonexistent, empty or inaccessible config file).

**Example:**

```
static void
list_settings ()
{
        char list[4096];
        char buffer[512];
        char *token;

        hexchat_pluginpref_list (ph, list);
        hexchat_printf (ph, "Current Settings:\n");
        token = strtok (list, ",");

        while (token != NULL)
        {
                hexchat_pluginpref_get_str (ph, token, buffer);
                hexchat_printf (ph, "%s: %s\n", token, buffer);
                token = strtok (NULL, ",");
        }
}
```

In the example above we query the list of currently stored settings, then print them one by one with their respective values. We always use *hexchat_pluginpref_get_str ()*, and that's because we can read an integer as string (but not vice versa).

#### Plugin GUI

void* **`hexchat_plugingui_add`**(*hexchat_plugin* **ph*, const char **filename*, const char **name*, const char **desc*, const char **version*, char **reserved*)

Adds a fake plugin to the GUI in *Window → Plugins and Scripts*. This does not need to be done for your actual plugin and is only used for interfaces to other languages like our python plugin.

**Returns** Handle to be used with `hexchat_plugingui_remove()`

void **`hexchat_plugingui_remove`**(*hexchat_plugin* **ph*, void **handle*)

Removes the fake plugin from the GUI. Again not to be used to remove your own plugin.

**Parameters**

- **`handle`** – Handle returned by `hexchat_plugingui_add()`

### 7.2.3 Scripting

There are multiple scripting languages supported by HexChat but currently only Python (2.7 & 3.5), Perl (5.20), and Lua are officially supported with Lua being the recommended option.

#### HexChat D-Bus Interface

For more help you can see the HexChat plugin interface documentation and see some examples in Python and C.

**Warning:** The dbus interface may change in the future.

You can use the "/org/hexchat/Remote" object with interface "org.hexchat.plugin", but his context can be changed by other clients at any moment and you may receive signal asked by other clients. So for more complex usage it's better to get your own remote object. Using "Connect" method on interface "org.hexchat.connection"

##### Available methods on *org.hexchat.connection* interface:

- "Connect"
  - Parameters:

        - gchar*: filename
        - gchar*: name
        - gchar*: description
        - gchar*: version
    - Returns:
        - gchar*: Your own object's path.
- "Disconnect"
    - No parameter, no return value. It frees your remote object.

#### Available methods on *org.hexchat.plugin* interface:

- "Command"
    - Parameters:
        - gchar*: the command name without the "/". (e.g. "nick pseudo")
- "Print"
    - Parameters:
        - gchar*: text to print on the HexChat window.
- "FindContext"
    - Parameters:
        - gchar*: the server name. Can be NULL.
        - gchar*: the channel name. Can be NULL.
    - Returns:
        - guint: context ID.
- "GetContext"
    - Returns:
        - guint: current context's ID.
- "SetContext"
    - Parameters:
        - guint: context ID to switch, returned by "FindContext" or "GetContext"
    - Returns:
        - gboolean:
            - 1: Success.
            - 0: Failure.
- "GetInfo"
    - Parameters:
        - gchar*: ID of the information you want.
    - Returns:

        * gchar*: information you requested.

- "GetPrefs"
    - Parameters:
        * gchar*: Setting name required.
    - Returns:
        * int:
            · 0: Failed.
            · 1: Returned a string.
            · 2: Returned an integer.
            · 3: Returned a boolean.
        * gchar*: the information requested if it's a string.
        * int: the information requested if it's a integer or boolean.
- "HookCommand"
    - Parameters:
        * gchar*: Name of the command (without the forward slash).
        * int: Priority of this command.
        * gchar*: String of text to display when the user executes /help for this command. May be NULL if you're lazy.
        * int: Value to returns when the command is catched. See HEXCHAT_EAT_*.
    - Returns:
        * guint: The ID of the hook.
- "HookServer"
    - Parameters:
        * gchar*: Name of the server event.
        * int: Priority of this command.
        * int: Value to returns when the command is catched. See HEXCHAT_EAT_*.
    - Returns:
        * guint: The ID of the hook.
- "HookPrint"
    - Parameters:
        * gchar*: Name of the print event.
        * int: Priority of this command.
        * int: Value to returns when the command is catched. See HEXCHAT_EAT_*.
    - Returns:
        * guint: The ID of the hook.
- "Unhook"

    - Parameters:
        * guint: ID of the hook to unhook. (the return value of "HookCommand", "HookServer" or "HookPrint")

- "ListGet"
    - Parameters:
        * gchar*: The list name.
    - Returns:
        * guint: List ID.
- "ListNext"
    - Parameters:
        * guint: List ID returned by "ListGet".
    - Returns:
        * gboolean: says if there is no more item in the list.
- "ListStr"
    - Parameters:
        * guint: List ID returned by "ListGet".
        * gchar*: Name of the information needed.
    - Returns:
        * gchar*: The information requested.

Warning: "context" attribute of "channels" list should be get with "ListInt"

- "ListInt"
    - Parameters:
        * guint: List ID returned by "ListGet".
        * gchar*: Name of the information needed.
    - Returns:
        * guint: The information requested.
- "ListTime"
    - Parameters:
        * guint: List ID returned by "ListGet".
        * gchar*: Name of the information needed.
    - Returns:
        * guint64: The information requested.
- "ListFields"
    - Parameters:
        * gchar*: The list name.
    - Returns:

    - gchar**: information names in this list.

- “ListFree”
  - Parameters:
    - guint: List ID returned by “ListGet”.
- “EmitPrint”
  - Parameters:
    - gchar*: Text event to print.
    - gchar**: NULL terminated array of string.
  - Returns:
    - gboolean:
      - 1: Success.
      - 0: Failure.
- “Nickcmp”
  - Parameters:
    - gchar*: String to compare.
    - gchar*: String to compare.
  - Returns:
    - int: An integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.
- “Strip”
  - Parameters:
    - gchar*: String to strip.
    - int: Length of the string (or -1 for NULL terminated).
    - int: Bit-field of flags:
      - 0: Strip mIRC colors.
      - 1: Strip text attributes.
  - Returns:
    - gchar*: stripped string.
- “SendModes”
  - Parameters:
    - gchar**: NULL terminated array of targets (strings). The names of people whom the action will be performed on.
    - int: Maximum modes to send per line.
    - gchar: Mode sign, ‘-‘ or ‘+’.
    - gchar: Mode char, e.g. ‘o’ for Ops.

#### Available signals:

- "ServerSignal"
  - Parameters:
    - gchar**: word returned by HexChat.
    - gchar**: word_eol returned by HexChat.
    - guint: the ID of the hook (the return value of "HookServer").
    - guint: the ID of the context where the event come from.
- "CommandSignal"
  - Parameters:
    - gchar**: word returned by HexChat.
    - gchar**: word_eol returned by HexChat.
    - guint: the ID of the hook (the return value of "HookCommand").
    - guint: the ID of the context where the event come from.
- "PrintSignal"
  - Parameters:
    - gchar**: word returned by HexChat.
    - guint: the ID of the hook (the return value of "HookPrint").
    - guint: the ID of the context where the event come from.
- "UnloadSignal"
  - Emitted when the user asks to unload your program. Please exit(0); when received!

## HexChat Javascript Interface

### Information

The javascript plugin does not come with HexChat, more information on it can be found Here

This page is simply for hosting its API docs and is a work in progress.

### Functions

**Generic Fuctions**

`print` (*object*)

Prints text to the current context.

**Arguments**

- `object` – Object to be converted to a string

`command` (*string*)

Runs a command in the current context.

**Hook Functions**

**`hook_print`**(*name*, *callback*[, *userdata*, *priority* ])
Calls specified callback anytime the specified event happens. The Print Events can be found in *Settings* → *Text Events*.

**Returns** Hook handler

**`hook_server`**(*name*, *callback*[, *userdata*, *priority* ])
Calls specified callback anytime the specified event happens.

**Arguments**

- **`name`** – IRC numeric or named event

**Returns** Hook handler

**`hook_command`**(*name*, *callback*[, *userdata*, *priority* ])
Calls specified callback anytime the specified command is run.

**Returns** Hook handler

**`hook_special`**(*name*, *callback*[, *userdata*, *priority* ])
Calls specified callback anytime the specified special event happens.

Special Events:

- Open Context
- Close Context
- Focus Tab
- Focus Window
- DCC Chat Text
- Key Press

**Returns** Hook handler

**`hook_timer`**(*timeout*, *callback*[, *userdata* ])
Calls specified callback in your specified interval (in milliseconds).

If you return *true* the callback will continue to be called, otherwise it is removed.

**Returns** Hook handler

**`hook_unload`**(*callback*[, *userdata* ])
Calls specified callback when the script is unloaded.

**Returns** Hook handler

**`unhook`**(*handler*)
Removes any hook registered above.

**Plugin Preferences** You can use pluginpref to easily store and retrieve settings.

**`set_pluginpref`**(*name*, *value*)
Stores settings in addon_*SCRIPT_NAME*.conf in the config dir.

**Returns**

- False: Failure
- True: Success

**get_pluginpref**(*name*)
This will return the value of the variable of that name. If there is none by this name it will return undefined.

**Returns** String or Integer of stored setting or None if it does not exist.

**Note:** Strings of numbers are always returned as Integers.

**del_pluginpref**(*name*)
Deletes the specified variable.

**Returns**

- False: Failure
- True: Success (or never existing),

**list_pluginpref**()
Returns a list of all currently set preferences.

## HexChat Lua Interface

### Scripts

Just like many other language plugins, the Lua plugin provides the commands `/load`, `/unload`, and `/reload` which operate on Lua scripts. You can forcibly load a Lua script with any extension by using `/lua load`, `/lua unload` or `/lua reload` instead. All files ending in `.lua` and `.luac` in the addons directory will be automatically loaded on startup.

Every script gets its own isolated state. The state is initialized with the Lua standard library, and has a global table called `hexchat` providing the API, which is described below.

### Commands

**/lua** `load <filename>`

**/load** `<filename>`

Loads a script with the given filename. `/load` will only load files ending in `.lua` and `.luac`.

**/lua** `unload <filename>`

**/unload** `<filename>`

Unloads a script with the given filename.

**/reload** `<filename>`‘

**/lua** `reload <filename>`‘

Reloads a script with the given filename.

**/lua** `list`

Lists loaded Lua scripts.

**/lua** `exec <code>`

Executes given code in the interpreter.

**/lua** `console`

Opens an interactive console. Messages to that tab are intercepted and interpreted as code.

**/lua** `inject <filename> <code>`

Executes given code in the context of a given script (which has to be loaded).

**/lua** `reset`

Reloads the interpreter (but not the scripts).

### Environment

**Note:** This only *directly* applies to **Windows** since other platforms rely on the Lua version and libraries installed on your system. It is most likely still of use, however.

The embedded Lua implementation is LuaJIT . LuaJIT by itself provides a few extensions, such as the the `bit` and `ffi` modules. It generally follows Lua 5.1 and has a few backwards compatible additions from Lua 5.2, which are explained on the extensions page.

Hexchat then provides the global `hexchat` table, which is described in detail in the *API* section below.

Additionally, the `lgi` module is available, which provides GObject-introspection, since it is used by Hexchat internally. For looking up mappings, the Python GObject Introspection API reference can be used, as it is very similar.

#### References

**LuaJIT** http://luajit.org/luajit.html

**LuaJIT Extensions** http://luajit.org/extensions.html

**BitOp Library** http://bitop.luajit.org/api.html

**FFI Library** http://luajit.org/ext_ffi.html

**LGI project page** https://github.com/pavouk/lgi

**LGI User Guide** https://github.com/pavouk/lgi/blob/master/docs/guide.md

**Gnome API Reference** https://developer.gnome.org/references

**Python GObject Introspection API Reference** http://lazka.github.io/pgi-docs/index.html

### API

The HexChat API is accessible through the `hexchat` table.

#### General functions

**hexchat.register(name, version, description)** Upon initialization every script should introduce itself by calling this function. Failure to do so will result in the script being unloaded immediately.

**hexchat.command(cmd)** Executes the command `cmd` in the current context, as if `/cmd` was typed by the user.

See `/help` for valid commands and their documentation.

**hexchat.print(...)** Prints zero or more values to the current tab. This function also replaces the global `print` function.

**hexchat.emit_print(event, ...)** Emits a text event (can be found in Settings->Text Events) into the current tab. `...` are the strings that are passed as arguments to the event. At the moment due to internal limitations, only 5 arguments are passed as there aren't any text events with more arguments.

**hexchat.send_modes(targets, mode[, max])** Sets multiple modes in the current context, possibly grouping them together up to the server limit. `targets` needs to be an array of strings, `mode` needs to be a string of 2 characters: + or − followed by the mode letter. `max` is the number of modes on one line, if omitted the serverside limit is used.

**hexchat.nickcmp(a, b)** Compares 2 strings case-insensitively, in accordance with current server's casemapping. Returns a negative number if `a` is less than `b`, zero if they are equal, and a positive number if `a` is more than `b`.

**hexchat.strip(string[, keep_colors[, keep_attrs]])** Removes color codes from the given string. If `keep_colors` is a truthy value, colors are not removed. If `keep_attrs` is a truthy value, attributes such as bold or underline are not removed. Returns the resulting string.

**hexchat.get_info(id)** Returns information about the current context. `id` is a string determining the information you want. It can be one of the following (case sensitive):

| ID | Return value |
|---|---|
| away | Away reason or `nil` if you are not away |
| channel | Current context's name |
| charset | Character set used in the current context |
| configdir | HexChat config directory, e.g. `/home/user/.config/hexchat` |
| event_text <name> | Text event format string for `<name>` |
| host | Real hostname of the server you connected to |
| inputbox | The input box contents, what the user has typed |
| libdirfs | Library directory. e.g. `/usr/lib/hexchat`. The same directory is used for autoloading plugins |
| modes | Channel modes, or `nil` if not known |
| network | Current network name, or `nil` if not known |
| nick | Your current nickname |
| password | Password for this network or `nil` |
| server | Current server name (what the server claims to be) or `nil` if you are not connected |
| topic | Current channel topic |
| version | HexChat version number |
| win_status | Window status: `active`, `hidden` or `normal` |
| win_ptr | A light userdata pointer to the native window. GtkWindow on unix, HWND on windows |
| gtkwin_ptr | A light userdata pointer to a GtkWindow. (Usable within lgi) |

**hexchat.iterate(list)** Iterate through the list `list`. To be used with generic for-loops in the following fashion:

```lua
for chan in hexchat.iterate("channels") do
    print(chan.server .. ": " .. chan.channel)
end
```

List of possible values of `list`, along with respective keys:

| `channels` | List of channels, queries and their servers |
|---|---|
| Name | Description |
| channel | Tab name |
| channelkey | Channel's key or `nil` |
| chanmodes | Channel modes, e.g. `beI,k,l` (2.12.2+) |
| chantypes | Channel types, e.g. `#!&` |
| context | Tab's context object |
| flags | • 2^0 = Connected<br>• 2^1 = Connecting<br>• 2^2 = Marked away<br>• 2^3 = End of MOTD<br>• 2^4 = Has WHOX<br>• 2^5 = Has IDMSG<br>• 2^6 = Hide Join/Parts<br>• 2^7 = Hide Join/Parts unset<br>• 2^8 = Beep on Message<br>• 2^9 = Beep on Message unset<br>• 2^10 = <unused><br>• 2^11 = Logging<br>• 2^12 = Logging unset<br>• 2^13 = Scrollback<br>• 2^14 = Scrollback unset<br>• 2^15 = Strip colors<br>• 2^16 = Strip colors unset<br>• 2^17 = Flash tray<br>• 2^18 = Flash tray unset<br>• 2^19 = Flash taskbar<br>• 2^20 = Flash taskbar unset |
| id | Unique server ID |
| lag | Lag in milliseconds |
| maxmodes | Maximum modes per line |
| network | Network name |
| nickprefixes | Nickname prefixes, e.g. `@+` |
| nickmodes | Nickname mode chars, e.g. `ov` |
| queue | Number of bytes in the send-queue |
| server | Server name to which this channel belongs |
| type | • 1 = Server<br>• 2 = Channel<br>• 3 = Dialog<br>• 4 = Notice<br>• 5 = SNotice |
| users | Number of users in this channel |

| `users` | Current contexts users |
|---|---|
| Name | Description |
| account | Account name or `nil` |
| away | Away message or `nil` |
| host | Host name in form of `user@host` or `nil` |
| lasttalk | Time stamp of last time they spoke |
| nick | Nick name |
| prefix | Prefix character such as `@` |
| realname | Real name |
| selected | If they are selected in the userlist |

| `dcc` | List of DCC file transfers |
|---|---|
| Name | Description |
| address32 | Address of the remote user (ipv4 address) |
| cps | Bytes per second (speed) |
| destfile | Destination full pathname |
| file | File name |
| nick | Nickname of person who the file is from/to |
| port | TCP port number |
| pos | Bytes sent/received |
| poshigh | Bytes sent/received, high order 32 bits |
| resume | Point at which this file was resumed (or zero if it was not resumed) |
| resumehigh | Point at which this file was resumed, high order 32 bits |
| size | File size in bytes, low order 32 bits (cast it to unsigned) |
| sizehigh | File size in bytes, high order 32 bits |
| status | • 0 = Queued<br>• 1 = Active<br>• 2 = Failed<br>• 3 = Done<br>• 4 = Connecting<br>• 5 = Aborted |
| type | • 0 = Send<br>• 1 = Recieve<br>• 1 = ChatRecv<br>• 1 = ChatSend |

| `ignore` | Current ignore list |
|---|---|
| Name | Description |
| mask | Ignore mask, .e.g. `\*\!\*\@\*.aol.com` |
| flags | • 2^0 = Private<br>• 2^1 = Notice<br>• 2^2 = Channel<br>• 2^3 = CTCP<br>• 2^4 = Invite<br>• 2^5 = Unignore<br>• 2^6 = NoSave<br>• 2^7 = DCC |

| `notify` | List of people on notify |
|---|---|
| Name | Description |
| networks | Networks to which this nick applies. Comma separated. May be `nil` |
| nick | Nickname |
| flags | Bit field of flags. 0=Is online. |
| on | Unix timestamp of when user came online. |
| off | Unix timestamp of when user went offline. |
| seen | Unix timestamp of when user the user was last verified still online. |

**hexchat.props** A table containing the values of a `"channels"` list for the current context.

**Preferences**

**hexchat.prefs** You can access HexChat's settings via this pseudo-table, see `/set` for a list of keys. Note that you cannot modify the table. Instead, you should use `hexchat.command("set -quiet <key> <value>")`

There are also these special preferences:

- `id` - Unique server id
- `state_cursor` - Location of cursor in input box

**Hooks** Some hooks are executed in a priority order, and hooks executed earlier can prevent later hooks from being invoked. The following constants determine priorities of such hooks and are passed to the hooking function:

- `hexchat.PRI_HIGHEST` - The highest priority.
- `hexchat.PRI_HIGH`
- `hexchat.PRI_NORM` - The default priority.
- `hexchat.PRI_LOW`
- `hexchat.PRI_LOWEST` - The lowest priority.

The following constants determine whether to pass the event on after the hook has finished. One of these has to be returned from the callback:

- `hexchat.EAT_NONE` - Let other hooks see the event.
- `hexchat.EAT_HEXCHAT` - Let other hooks see the event, but prevent HexChat itself from seeing it.
- `hexchat.EAT_PLUGIN` - Don't let remaining hooks see the event, but let HexChat know about it.
- `hexchat.EAT_ALL` - Consume this event completely, don't let anyone else know about it.

All hooking functions return an object which can be later used to remove the hook, but the hooks are also removed automatically when the script is unloaded or reloaded.

Unlike the C and Python APIs, there isn't a userdata value passed to the hooks. Instead you should use upvalues, closures, and/or anonymous functions.

**hexchat.hook_command(command, callback[, help[, priority]])** Hooks the function `callback` to be executed whenever `/command` is entered. `help` is the helptext for the `/help` command. Returns a hook object. The callback receives an array of words, and an array of word_eols as arguments.

If `command` is nil, then all non-command text is hooked instead, including `/say`.

**hexchat.hook_print(event, callback[, priority])** Hooks the function `callback` to be executed whenever the text event `event` is to be printed. Returns a hook object. The callback receives the array of event's arguments as its only argument.

There are also a few extra events you can hook using this function:

- `Open Context` - Emitted when a new context is created.
- `Close Context` - Emitted when a context is closed.
- `Focus Tab` - Emitted when a tab is brought to the front.
- `Focus Window` - Emitted when a toplevel window is focused, or the main tab-window is focused by the window manager.
- **`DCC Chat Text` - Emitted when some text from a DCC Chat arrives. It provides these elements in the word list:**
  - Address
  - Port
  - Nick
  - Message
- **`Key Press` - Emitted when some keys are pressed in the input box. It provides these elements in the word list:**
  - Key Value
  - Modifier bitfield (Shift, CapsLock, Alt, etc)
  - String version of the key
  - Length of the string (may be 0 for unprintable keys)

**hexchat.hook_server(command, callback[, priority])** Hooks the function `callback` to be executed whenever `command` is received from the server. Returns a hook object. The callback receives an array of words, and an array of word_eols as arguments.

If `command` is nil, then the callback is called for every received line.

**hexchat.hook_timer(interval, callback)** Hooks the function `callback` to be executed after `inverval` milliseconds. Returns a hook object. As long as the callback returns a truthy value, it is scheduled to happen after the same preiod of time.

**hexchat.hook_unload(callback)** Hooks the function `callback` to be executed when the current script is unloaded. Returns a hook object.

**hook:unhook() and hexchat.unhook(hook)** Removes the given hook. A hook can only be removed once.

**Contexts** A context corresponds to a HexChat window or tab. Some of the functions in `hexchat.*` will do something in the current tab. Using contexts you can perform such actions in other tabs instead. Two context objects can be tested for equality using the == operator, which will return true if the contexts refer to the same tab. All methods of a context object will error if the object is invalidated and points to a tab that is closed.

**hexchat.get_context()** Returns a context object for the current context.

**hexchat.find_context(server_name, channel_name)** Finds a context object for a tab on the given channel of the given channel. If `server_name` is nil, it searches for the given channel or query across all servers. If `channel_name` is nil, finds the frontmost tab of the given server. If both are `nil`, returns current context. In any case, if the specified tab was not found, the function returns nil.

**ctx:set() and hexchat.set_context(ctx)** Makes `ctx` the "current" context. All `hexchat.*` functions will be using this context. This setting only persists within one event. Next time any of the callbacks is called, the current context will be set to the actual one.

Returns a boolean indicating whether the context was successfully set. Failure will only occur if the context has been closed.

**ctx:find_context(server_name, channel_name)** Identical to `hexchat.find_context`, except the defaults are based on the current context.

**ctx:print(...)** Prints zero or more values in the given context.

**ctx:emit_print(event, ...)** Emits a text event into the given context. See `hexchat.emit_print`.

Returns a boolean indicating success.

**ctx:command(cmd)** Executes the command `/cmd` in the given context. See `hexchat.command`.

**ctx:nickcmp(a, b)** Compares 2 strings using casemapping from the given context. See `hexchat.nickcmp`.

**ctx:get_info(id)** Returns information about the given context. See `hexchat.get_info`.

**ctx:iterate(list)** Iterate through a list within the given context. See `hexchat.iterate`.

**Plugin preferences**

**hexchat.pluginprefs** To persistently store your script's settings, you can use this pseudo-table. The values inside will persist across script reloads, HexChat restarts, and reboots. Currently, you can only store and read strings and numbers associated to string keys, and iterate through the table with `pairs()`.

**Attributes** Attributes correspond to extra metadata for messages, such as server-time (currently the only supported attribute). Some functions have attributes-enhanced versions.

**hexchat.attrs()** Returns a new attributes object. It has only one field: `server_time_utc`.

**hexchat.emit_print_attrs(attrs, event, ...) and ctx:emit_print_attrs(attrs, event, ...)** Analogous to `hexchat.emit_print` and `ctx:emit_print` respectively, but passes an extra attributes argument.

**hexchat.hook_print_attrs(event, callback[, priority])** Identical to `hexchat.hook_print`, except that the callback receives an additional second argument with an attributes object and that the aforementioned extra events cannot be hooked.

**hexchat.hook_server_attrs(command, callback[, priority])** Identical to `hexchat.hook_server`, except that the callback receives an additional third argument with an attributes object.

## HexChat Perl Interface

### Introduction

This is the Perl interface for HexChat. If there are any problems, questions, comments or suggestions please email them to the address on the bottom of this page.

### Constants

#### Priorities

- **HexChat::PRI_HIGHEST**
- **HexChat::PRI_HIGH**
- **HexChat::PRI_NORM**
- **HexChat::PRI_LOW**
- **HexChat::PRI_LOWEST**

#### Return values

- **HexChat::EAT_NONE** - pass the event along
- **HexChat::EAT_HEXCHAT** - don't let HexChat see this event
- **HexChat::EAT_PLUGIN** - don't let other scripts and plugins see this event but HexChat will still see it
- **HexChat::EAT_ALL** - don't let anything else see this event

#### Timer and fd hooks

- **HexChat::KEEP** - keep the timer going or hook watching the handle
- **HexChat::REMOVE** - remove the timer or hook watching the handle

#### hook_fd flags

- **HexChat::FD_READ** - invoke the callback when the handle is ready for reading
- **HexChat::FD_WRITE** - invoke the callback when the handle is ready for writing
- **HexChat::FD_EXCEPTION** - invoke the callback if an exception occurs
- **HexChat::FD_NOTSOCKET** - indicate that the handle being hooked is not a socket

**Exports** The following tags are supported:

- `:all` - exports everything
- `:constants` - exports all the constants
- `:hooks` - exports `hook_*` functions as well us `unhook`
- `:util` - everything else

By default only the constants are exported.

### Functions

**HexChat::register( $name, $version, [$description,[$callback]] )**

- `$name` - The name of this script
- `$version` - This script's version
- `$description` - A description for this script
- `$callback` - This is a function that will be called when the is script unloaded. This can be either a reference to a function or an anonymous sub reference.

This is the first thing to call in every script.

**HexChat::hook_server( $message, $callback, [%options] )**

**HexChat::hook_command( $command, $callback, [%options] )**

**HexChat::hook_print( $event,$callback, [%options] )**

**HexChat::hook_timer( $timeout,$callback, [%options | $data] )**

**HexChat::hook_fd( $handle, $callback, [ %options ] )** These functions can be to intercept various events. `hook_server` can be used to intercept any incoming message from the IRC server. `hook_command` can be used to intercept any command, if the command doesn't currently exist then a new one is created. `hook_print` can be used to intercept any of the events listed in *Setttings -> Text Events*. `hook_timer` can be used to create a new timer

- **$message** - server message to hook such as PRIVMSG
- **$command** - command to intercept, without the leading /
- **$event** - one of the events listed in *Settings -> Text Events*
- **$timeout** - timeout in milliseconds
- **$handle** - the I/O handle you want to monitor with hook_fd. This must be something that has a fileno. See perldoc -f fileno or fileno
- **$callback** - callback function, this is called whenever the hooked event is trigged, the following are the conditions that will trigger the different hooks. This can be either a reference to a function or an anonymous sub reference.
- **%options** - a hash reference containing addional options for the hooks

Valid keys for %options:

| | |
|---|---|
| data | Additional data that is to be associated with the hook. For timer hooks this value can be provided either as `HexChat::hook_timer( $timeout, cb,{data=> data})` or `HexChat::hook_timer( $timeout, $cb, $data )`. However, this means that hook_timer cannot be provided with a hash reference containing data as a key.<br>Example:<br>`my $options = { data => [@arrayOfStuff] };`<br>`HexChat::hook_timer( $timeout, $cb, $options );`<br>In this example, the timer's data will be `[@arrayOfStuff]` and not `{ data => [@arrayOfStuff] }`. This key is valid for all of the hook functions. Default is `undef`. |
| priority | Sets the priority for the hook. It can be set to one of the `HexChat::PRI_*` constants. This key only applies to server, command and print hooks. Default is `HexChat::PRI_NORM`. |
| help_text | Text displayed for /help $command. This key only applies to command hooks. Default is "". |
| flags | Specify the flags for a fd hook. See hook fd flags section for valid values. On Windows if the handle is a pipe you specify `HexChat::FD_NOTSOCKET` in addition to any other flags you might be using. This key only applies to fd hooks. Default is `HexChat::FD_READ`. |

**When callbacks are invoked** Each of the hooks will be triggered at different times depending on the type of hook.

| Hook Type | When the callback will be invoked |
|---|---|
| server hooks | A `$message` message is received from the server. |
| command hooks | The `$command` command is executed, either by the user or from a script. |
| print hooks | X-Chat is about to print the message for the `$event` event. |
| timer hooks | Called every `$timeout` milliseconds (1000 milliseconds are 1 second). The callback will be executed in the same context where the `hook_timer` was called, or if the context no longer exists then it will execute in a random context. |
| fd hooks | Depends on the flags that were passed to `hook_fd`. See `hook_fd` flags in the section above. |

The value return from these hook functions can be passed to `HexChat::unhook` to remove the hook.

**Callback Arguments** All callback functions will receive their arguments in `@_` like every other Perl subroutine.

- Server and command callbacks

  `$_[0]` - array reference containing the IRC message or command and arguments broken into words example: /command arg1 arg2 arg3

  - `$_[0][0]` - command
  - `$_[0][1]` - arg1
  - `$_[0][2]` - arg2

  - `$_[0][3]` - arg3

  `$_[1]` - array reference containing the Nth word to the last word example: /command arg1 arg2 arg3

  - `$_[1][0]` - command arg1 arg2 arg3
  - `$_[1][1]` - arg1 arg2 arg3
  - `$_[1][2]` - arg2 arg3
  - `$_[1][3]` - arg3

  `$_[2]` - the data that was passed to the hook function

- Print callbacks

  `$_[0]` - array reference containing the values for the text event, see *Settings -> Text Events* `$_[1]` - the data that was passed to the hook function

- Timer callbacks

  `$_[0]` - the data that was passed to the hook function

- fd callbacks

  `$_[0]` - the handle that was passed to hook_fd `$_[1]` - flags indicating why the callback was called `$_[2]` - the data that was passed to the hook function

**Callback return values** All server, command and print callbacks should return one of the `HexChat::EAT_*` constants. Timer callbacks can return `HexChat::REMOVE` to remove the timer or `HexChat::KEEP` to keep it going.

**Miscellaneous Hook Related Information** For server hooks, if `$message` is "RAW LINE" then `$cb` will be called for every IRC message that HexChat receives.

For command hooks if `$command` is "" then `$cb` will be called for messages entered by the user that is not a command.

For print hooks besides those events listed in *Settings -> Text Events*, these additional events can be used.

| Event | Description |
|---|---|
| "Open Context" | a new context is created |
| "Close Context" | a context has been closed |
| "Focus Tab" | when a tab is brought to the front |
| "Focus Window" | when a top level window is focused or the main tab window is focused by the window manager |
| "DCC Chat Text" | when text from a DCC Chat arrives. $_[0] will have these values<br>• `$_[0][0]` - Address<br>• `$_[0][1]` - Port<br>• `$_[0][2]` - Nick<br>• `$_[0][3]` - Message |
| "Key Press" | used for intercepting key presses<br>• `$_[0][0]` - key value<br>• `$_[0][1]` - state bitfield, 1 - shift, 4 - control, 8 - alt<br>• `$_[0][2]` - string version of the key which might be empty for unprintable keys<br>• `$_[0][3]` - length of the string in `$_[0][2]` |

**HexChat::unhook( $hook )**

- **$hook** - the hook that was previously returned by one of the `HexChat::hook_*` functions

This function is used to removed a hook previously added with one of the `HexChat::hook_*` functions.

It returns the data that was passed to the HexChat::hook_* function when the hook was added.

**HexChat::print( $text | @lines, [$channel,[$server]] )**

- **$text** - the text to print
- **@lines** - array reference containing lines of text to be printed all the elements will be joined together before printing
- **$channel** - channel or tab with the given name where `$text` will be printed
- **$server** - specifies that the text will be printed in a channel or tab that is associated with `$server`

The first argument can either be a string or an array reference of strings. Either or both of `$channel` and `$server` can be `undef`.

If called as HexChat::print( $text ), it will always return true. If called with either the channel or the channel and the server specified then it will return true if a context is found and false otherwise. The text will not be printed if the context is not found. The meaning of setting `$channel` or `$server` to `undef` is the same as find_context.

**HexChat::printf( $format, LIST )**

- **$format** - a format string, see "perldoc -f sprintf" for further details
- **LIST** - list of values for the format fields

**HexChat::command( $command | @commands, [$channel,[$server]] )**

- **$command** - the command to execute, without the leading /

- **@commands** - array reference containing a list of commands to execute
- **$channel** - channel or tab with the given name where `$command` will be executed
- **$server** - specifies that the command will be executed in a channel or tab that is associated with `$server`

The first argument can either be a string or an array reference of strings. Either or both of `$channel` and `$server` can be `undef`.

If called as HexChat::command( $command ), it will always return true. If called with either the channel or the channel and the server specified then it will return true if a context is found and false otherwise. The command will not be executed if the context is not found. The meaning of setting `$channel` or `$server` to `undef` is the same as find_context.

**HexChat::commandf( $format, LIST )**

- **$format** - a format string, see "perldoc -f sprintf" for further details
- **LIST** - list of values for the format fields

**HexChat::find_context( [$channel, [$server]] )**

- **$channel** - name of a channel
- **$server** - name of a server

Either or both of `$channel` and `$server` can be `undef`. Calling `HexChat::find_context()` is the same as calling `HexChat::find_context( undef, undef)` and `HexChat::find_context( $channel )` is the same as `HexChat::find_context( $channel, undef ).`

If `$server` is `undef`, find any channel named `$channel`. If `$channel` is `undef`, find the front most window or tab named `$server`.If both `$channel` and `$server` are `undef`, find the currently focused tab or window.

Return the context found for one of the above situations or `undef` if such a context cannot be found.

**HexChat::get_context()** Returns the current context.

**HexChat::set_context( $context | $channel,[$server] )**

- **$context** - context value as returned from `get_context`, `find_context` or one of the fields in the list of hashrefs returned by `list_get`
- **$channel** - name of a channel you want to switch context to
- **$server** - name of a server you want to switch context to

See `find_context` for more details on `$channel` and `$server`.

Returns true on success, false on failure.

**HexChat::get_info( $id )**

- **$id** - one of the following case sensitive values

| ID | Return value | Associated Command(s) |
|---|---|---|
| away | away reason or `undef` if you are not away | AWAY, BACK |
| channel | the original name of the tab, not affected by SETTAB | SETTAB |
| charset | character-set used in the current context | CHARSET |
| configdir | HexChat config directory encoded in UTF-8. Examples: /home/user/.config/hexchat C:UsersuserAppdataRoamingHexChat | |
| event_text <Event Name> | text event format string for <Event name> Example: `my $channel_msg_format = HexChat::get_info( "event_text Channel Message" );` | |
| host | real hostname of the current server | |
| id | connection id | |
| inputbox | contents of the inputbox | SETTEXT |
| libdirfs | the system wide directory where HexChat will look for plugins. this string is in the same encoding as the local file system | |
| modes | the current channels modes or `undef` if not known | MODE |
| network | current network name or `undef`, this value is taken from the Network List | |
| nick | current nick | NICK |
| nickserv | nickserv password for this network or `undef`, this value is taken from the Network List | |
| server | current server name (what the server claims to be) `undef` if not connected | |
| state_cursor | current inputbox cursor position in characters | SETCURSOR |
| topic | current channel topic | TOPIC |
| version | HexChat's version number | |
| win_status | status of the HexChat window, possible values are "active", "hidden" and "normal" | GUI |
| win_ptr | native window pointer, GtkWindow * on Unix, HWND on Win32. On Unix if you have the Glib module installed you can use my `$window = Glib::Object->new\_from\_pointer( HexChat::get_info( "win\_ptr" ) );` to get a Gtk2::Window object. Additionally when you have detached tabs, each of the windows will return a different win_ptr for the different `Gtk2::Window` objects. See char_count.pl for a longer example of a script that uses this to show how many characters you currently have in your input box. | |
| gtkwin_ptr | similar to win_ptr except it will always be a GtkWindow * | |

This function is used to retrieve certain information about the current context. If there is an associated command then that command can be used to change the value for a particular ID.

**HexChat::get_prefs( $name )**

- **$name** - name of a HexChat setting (available through the /set command)

This function provides a way to retrieve HexChat's setting information.

Returns `undef` if there is no setting called called `$name`.

**HexChat::emit_print( $event, LIST )**

- **$event** - name from the Event column in *Settings –> Text Events*
- **LIST** - this depends on the Description column on the bottom of *Settings –> Text Events*

This functions is used to generate one of the events listed under *Settings –> Text Events*.

Note: when using this function you **must** return `HexChat::EAT_ALL` otherwise you will end up with duplicate events. One is the original and the second is the one you emit.

Returns true on success, false on failure.

**HexChat::send_modes( $target | @targets, $sign, $mode, [ $modes_per_line ] )**

- **$target** - a single nick to set the mode on
- **@targets** - an array reference of the nicks to set the mode on
- **$sign** - the mode sign, either '+' or '-'
- **$mode** - the mode character such as 'o' and 'v', this can only be one character long
- **$modes_per_line** - an optional argument maximum number of modes to send per at once, pass 0 use the current server's maximum (default)

Send multiple mode changes for the current channel. It may send multiple MODE lines if the request doesn't fit on one.

Example:

```
use strict;
use warnings;
use HexChat qw(:all);

hook_command( "MODES", sub {
    my (undef, $who, $sign, $mode) = @{$_[0]};
    my @targets = split /,/, $who;
    if( @targets > 1 ) {
       send_modes( \@targets, $sign, $mode, 1 );
    } else {
       send_modes( $who, $sign, $mode );
    }
    return EAT_HEXCHAT;
});
```

**HexChat::nickcmp( $nick1, $nick2 )**

- **$nick1, $nick2** - the two nicks or channel names that are to be compared

The comparsion is based on the current server. Either an RFC1459 compliant string compare or plain ascii will be using depending on the server. The comparison is case insensitive.

Returns a number less than, equal to or greater than zero if `$nick1` is found respectively, to be less than, to match, or be greater than `$nick2`.

**HexChat::get_list( $name )**

- **$name** - name of the list, one of the following: “channels”, “dcc”, “ignore”, “notify”, “users”

This function will return a list of hash references. The hash references will have different keys depend on the list. An empty list is returned if there is no such list.

“channels” - list of channels, querys and their server

| Key | Description |
|---|---|
| channel | tab name |
| channelkey | the key used to get into the channel |
| chantypes | channel types supported by the server, typically "#&" |
| context | can be used with set_context |
| flags | Server Bits:<br>• 0 Connected<br>• 1 Connecting<br>• 2 Away<br>• 3 EndOfMotd(Login complete)<br>• 4 Has WHOX<br>• 5 Has IDMSG (FreeNode)<br>The following correspond to the /chanopt command<br>• 6 Hide Join/Part Message (text_hidejoinpart)<br>• 7 unused (was for color paste)<br>• 8 Beep on message (alert_beep)<br>• 9 Blink Tray (alert_tray)<br>• 10 Blink Task Bar (alert_taskbar)<br>Example of checking if the current context has Hide Join/Part messages set:<br>• 1<br>• 2<br>• 3<br>`if( HexChat::context_info->{flags} & (1 << 6) ) {`<br>`    HexChat::print( "Hide Join/Part messages is ena`<br>`}` |
| id | Unique server ID |
| lag | lag in milliseconds |
| maxmodes | Maximum modes per line |
| network | network name to which this channel belongs |
| nickmodes | Nickname mode chars e.g. "vo" |
| nickprefixes | Nickname prefixes e.g. "+@" |
| queue | number of bytes in the send queue |
| server | server name to which this channel belongs |
| type | the type of this context - 1 - server - 2 - channel - 3 - dialog - 4 - notices - 5 - server notices |
| users | Number of users in a channel |

"dcc" - list of DCC file transfers

| Key | Value |
|---|---|
| address32 | address of the remote user(ipv4 address) |
| cps | bytes per second(speed) |
| destfile | destination full pathname |
| file | file name |
| nick | nick of the person this DCC connection is connected to |
| port | TCP port number |
| pos | bytes sent/received |
| poshigh | bytes sent/received, high order 32 bits |
| resume | point at which this file was resumed (zero if it was not resumed) |
| resumehigh | point at which this file was resumed, high order 32 bits |
| size | file size in bytes low order 32 bits |
| sizehigh | file size in bytes, high order 32 bits (when the files is > 4GB) |
| status | DCC Status:<br>• 0 - queued<br>• 2 - failed<br>• 3 - done<br>• 4 - connecting<br>• 5 - aborted |
| type | DCC Type:<br>• 0 - send<br>• 1 - receive<br>• 2 - chatrecv<br>• 3 - chatsend |

“ignore” - current ignore list

| Key | Value |
|---|---|
| mask | ignore mask. e.g: *!*@*.aol.com |
| flags | Bit field of flags.<br>• 0 - private<br>• 1 - notice<br>• 2 - channel<br>• 3 - ctcp<br>• 4 - invite<br>• 5 - unignore<br>• 6 - nosave<br>• 7 - dcc |

“notify” - list of people on notify

| Key | Value |
|---|---|
| networks | comma separated list of networks where you will be notfified about this user’s online/offline status or `undef` if you will be notificed on every network you are connected to |
| nick | nickname |
| flags | 0 = is online |
| on | time when user came online |
| off | time when user went offline |
| seen | time when user was last verified still online |

The values indexed by on, off and seen can be passed to localtime and gmtime, see perldoc -f localtime and perldoc -f gmtime for more details.

"users" - list of users in the current channel

| Key | Value |
|---|---|
| account | account name or `undef` (2.9.6+) |
| away | away status (boolean) |
| host | host name in the form: user@host or `undef` if not known |
| lasttalk | last time a user was seen talking, this is the an epoch time |
| nick | nick name |
| prefix | prefix character, .e.g: @ or + |
| realname | Real name or `undef` |
| selected | selected status in the user list, only works when retrieving the user list of the focused tab. You can use the /USELECT command to select the nicks |

"networks" - list of networks and the associated settings from network list

| Key | Value |
|---|---|
| autojoins | An object with the following methods:<br>• `channels()` - returns a list of this networks' autojoin channels in list context, a count of the number autojoin channels in scalar context<br>• `keys()` - returns a list of the keys to go with the channels, the order is the same as the channels, if a channel doesn't have a key, '' will be returned in its place<br>• `pairs()` - a combination of channels() and keys(), returns a list of (channels, keys) pairs. This can be assigned to a hash for a mapping from channel to key.<br>• `as_hash()` - return the pairs as a hash reference<br>• `as_string()` - the original string that was used to construct this autojoin object, this can be used with the JOIN command to join all the channels in the autojoin list<br>• `as_array()` - return an array reference of hash references consisting of the keys "channel" and "key"<br>• `as_bool()` - returns true if the network has autojoins and false otherwise |
| connect_commands | An array reference containing the connect commands for a network. An empty array if there aren't any |
| encoding | the encoding for the network |
| flags | a hash reference corresponding to the checkboxes in the network edit window<br>• allow_invalid - true if "Accept invalid SSL certificate" is checked<br>• autoconnect - true if "Auto connect to this network at startup" is checked<br>• cycle - true if "Connect to selected server only" is NOT checked<br>• use_global - true if "Use global user information" is checked<br>• use_proxy - true if "Bypass proxy server" is NOT checked<br>• use_ssl - true if "Use SSL for all the servers on this network" is checked |
| irc_nick1 | Corresponds with the "Nick name" field in the network edit window |
| irc_nick2 | Corresponds with the "Second choice" field in the network edit window |
| irc_real_name | Corresponds with the "Real name" field in the network edit window |
| irc_user_name | Corresponds with the "User name" field in the network edit window |
| network | Name of the network |
| nickserv_password | Corresponds with the "Nickserv password" field in the network edit window |
| selected | Index into the list of servers in the "servers" key, this is used if the "cycle" flag is false |
| server_password | Corresponds with the "Server password" field in the network edit window |
| servers | An array reference of hash references with a "host" and "port" key. If a port is not specified then 6667 will be used. |

**HexChat::user_info( [$nick] )**

- **$nick** - the nick to look for, if this is not given your own nick will be used as default

This function is mainly intended to be used as a shortcut for when you need to retrieve some information about only one user in a channel. Otherwise it is better to use `get_list`. If `$nick` is found a hash reference containing the same keys as those in the "users" list of `get_list` is returned otherwise `undef` is returned. Since it relies on `get_list` this function can only be used in a channel context.

**HexChat::context_info( [$context] )**

- **$context** - context returned from `get_context`, `find_context` and `get_list`, this is the context that you want infomation about. If this is omitted, it will default to current context.

This function will return the information normally retrieved with `get_info`, except this is for the context that is passed in. The information will be returned in a hashref. The keys of the hash are the `$id` you would normally supply to `get_info` as well as all the keys that are valid for the items in the "channels" list from `get_list`. Use of this function is more efficient than calling `get_list( "channels" )` and searching through the result.

Example:

```
use strict;
use warnings;
use HexChat qw(:all); # imports all the functions documented on this page

register( "User Count", "0.1",
   "Print out the number of users on the current channel" );
hook_command( "UCOUNT", \&display_count );
sub display_count {
    prnt "There are " . context_info()->{users} . " users in this channel.";
    return EAT_HEXCHAT;
}
```

**HexChat::plugin_pref_set( $setting, $value )**

- **$setting** - name of the setting you want to store
- **$value** - value of that setting

This function allows you to store settings in addon_perl.conf via HexChat. Returns 1 on success, 0 on failure.

**HexChat::plugin_pref_get( $setting )**

- **$setting** - name of the setting you want to retrieve

Returns the value of the specified setting or undef it is not found.

**HexChat::plugin_pref_del( $setting )**

- **$setting** - name of the setting you want to delete

Returns 1 on success (including deletion of not existing settings), 0 on failure.

**HexChat::plugin_pref_list( )** Returns a hashref of all stored settings or an empty hashref on failure.

**HexChat::strip_code( $string )**

- **$string** - string to remove codes from

This function will remove bold, color, beep, reset, reverse and underline codes from `$string`. It will also remove ANSI escape codes which might get used by certain terminal based clients. If it is called in void context `$string` will be modified otherwise a modified copy of `$string` is returned.

## Examples

### Asynchronous DNS resolution with hook_fd

```perl
use strict;
use warnings;
use HexChat qw(:all);
use Net::DNS;

hook_command( "BGDNS", sub {
    my $host = $_[0][1];
    my $resolver = Net::DNS::Resolver->new;
    my $sock = $resolver->bgsend( $host );

    hook_fd( $sock, sub {
        my $ready_sock = $_[0];
        my $packet = $resolver->bgread( $ready_sock );

        if( $packet->authority && (my @answers = $packet->answer ) ) {

            if( @answers ) {
                prnt "$host:";
                my $padding = " " x (length( $host ) + 2);
                for my $answer ( @answers ) {
                    prnt $padding . $answer->rdatastr . ' ' . $answer->type;
                }
            }
        } else {
            prnt "Unable to resolve $host";
        }

        return REMOVE;
    },
    {
        flags => FD_READ,
    });

    return EAT_HEXCHAT;
});
```

## Contact Information

Contact Lian Wan Situ at <atmcmnky [at] yahoo.com> for questions, comments and corrections about this page or the Perl plugin itself. You can also find me in #xchat on freenode under the nick Khisanth.

## HexChat Python Interface

### Features

Here are some of the features of the python plugin interface:

- Comprehensive, consistent and straightforward API
- Load, unload, reload, and autoload support
- Per plugin independent interpreter state
- Python interactive console
- Python interactive command execution
- Python 2 and 3 support (2.9.6+)
- Full thread support (except for Python2 on Windows)
- Stdout and stderr redirected to HexChat console
- Dynamic list management
- Nice context treatment
- Plugin preferences

**Python 2 or Python 3** As of HexChat 2.9.6 the plugin supports both so which should you pick:

As a user most older scripts will not be updated for Python 3 so 2 is your best bet.

As a developer I would just recommend you make your scripts compatible for both but do note that the Python 2 plugin on Windows does not support threads while the Python 3 plugin does.

### Commands

The Python plugin comes with a **`py`** command that takes these arguments.

**`load`** `<file>`
    Load a script with given filename. **`/load`** will also work.

**`unload`** `<filename|module name>`
    Unload module with given filename, or module name. **`/unload`** will also work.

**`reload`** `<filename|module name>`
    Reload module with given filename, or module name. **`/reload`** will also work.

**`list`**
    List Python scripts loaded

**`exec`** `<command>`
    Execute given Python command interactively. For example:

```
/py exec import hexchat; print(hexchat.get_info('channel'))
```

**`console`**
    Open the Python interactive console in a query `>>python<<`. Every message sent will be intercepted by the Python plugin interface, and interpreted interactively. Notice that the console and /py exec commands live in the same interpreter state.

**about**
Show some information about the Python plugin interface.

### Autoloading modules

If you want some module to be autoloaded together with the Python plugin interface (which usually loads at startup time), just make sure it has a `.py` extension and put it in the `addons` subdir of HexChat's config directory.

### Context theory

Before starting to explain what the API offers, I'll do a short introduction about the HexChat context concept. Not because it's something hard to understand, but because you'll understand better the API explanations if you know what I'm talking about.

You can think about a context as an HexChat channel, server, or query tab. Each of these tabs, has its own context, and is related to a given server and channel (queries are a special kind of channel).

The *current* context is the one where HexChat passes control to the module. For example, when HexChat receives a command in a specific channel, and you have asked HexChat to tell you about this event, the current context will be set to this channel before your module is called.

### Text Formatting

- Bold: `'\002'`
- Color: `'\003'`
- Hidden: `'\010'`
- Underline: `'\037'`
- Original Attributes: `'\017'`
- Reverse Color: `'\026'`
- Beep: `'\007'`
- Italics: `'\035'` (2.10.0+)

For example this will print underlined red text:

```
print('\037\00304Text!')
```

### Bit fields

Some lists return bit fields which many Python scripters may not be familair with so here is an example of how to use one:

```
...

# We know its the 15th bit field we want, this will get that value
text_strip = 1 << 15
text_strip_unset = 1 << 16 # If this is set use the global option

def get_chanopt (channel, option):
    for chan in hexchat.get_list('channels'):
        if chan.channel == channel:
            return bool(chan.flags & option)

if get_chanopt('#hexchat', text_strip_unset):
    stripped = bool(hexchat.get_prefs('text_stripcolor_msg'))
else:
    stripped = get_chanopt ('#hexchat', text_strip)

print('Color stripping in #hexchat is: {}'.format(stripped))
```

### Hello world

Here is the traditional *hello world* example.

```
__module_name__ = "helloworld"
__module_version__ = "1.0"
__module_description__ = "Python module example"

print("Hello world!")
```

This module will print "Hello world!" in the HexChat console, and sleep forever until it's unloaded. It's a simple module, but already introduces some concepts. Notice how the module information is set. This information is obligatory, and will be shown when listing the loaded HexChat modules.

### hexchat module

The hexchat module is your passport to every HexChat functionality offered by the Python plugin interface. Here's a simple example:

```
import hexchat
hexchat.prnt("Hi everyone!")
```

The following functions are available in the hexchat module.

#### Constants and Attributes

hexchat.**PRI_HIGHEST**
hexchat.**PRI_HIGH**
hexchat.**PRI_NORM**
hexchat.**PRI_LOW**
hexchat.**PRI_LOWEST**
Priority given to hooks.

hexchat.**EAT_PLUGIN**
hexchat.**EAT_HEXCHAT**
hexchat.**EAT_ALL**
hexchat.**EAT_NONE**
Used as return values for callbacks.

hexchat.**__version__**
Tuple of (MAJOR_VERSION, MINOR_VERSION)

#### Generic functions

hexchat.**prnt**(*string*)
This function will print string in the current context. It's mainly useful as a parameter to pass to some other function, since the usual print statement will have the same results. You have a usage example above.

This function is badly named because `"print"` is a reserved keyword of the Python language until Python 3.

`hexchat.`**`emit_print`**(*event_name*, **args*)

This function will generate a *print event* with the given arguments. To check which events are available, and the number and meaning of arguments, have a look at the *Settings* → *Text Events* window. Here is one example:

```
hexchat.emit_print("Channel Message", "John", "Hi there", "@")
```

With plugin version 1.0+ this function takes keywords for certain attributes such as *time*.

`hexchat.`**`command`**(*string*)

Execute the given command in the current *context*. This has the same results as executing a command in the HexChat window, but notice that the / prefix is not used. Here is an example:

```
hexchat.command("server irc.openprojects.net")
```

A list of commands is provided here: *List of Commands*.

`hexchat.`**`nickcmp`**(*s1*, *s2*)

This function will do an RFC1459 compliant string comparison and is useful to compare channels and nicknames.

**Returns** Returns 0 if they match and less than or greater than 0 if s1 is less than or greather than s2

```
if hexchat.nickcmp(nick, "mynick") == 0:
    print("They are the same!")
```

`hexchat.`**`strip`**(*text*[, *length=-1*, *flags=3* ])

This function can strip colors and attributes from text.

**Parameters**

- **`length`** – -1 for entire string
- **`flags`** – 1: Strip Colors 2: Strip Attributes 3: Strip All

**Returns** Stripped String

```
text = '\00304\002test' # Bold red text
print(text)
print(hexchat.strip(text, len(text), 1)) # Bold uncolored text
```

### Information retrieving functions

`hexchat.`**`get_info`**(*type*)

Retrieve the information specified by the `type` string in the current context. At the moment of this writing, the following information types are available to be queried:

- **away:** Away reason or None if you are not away.
- **channel:** Channel name of the current context.
- **charset:** Charset in current context.
- **configdir:** HexChat config directory e.g.: “~/.config/hexchat”.
- **event_text NAME:** Returns text event string for requested event.
- **gtkwin_ptr:** Returns hex representation of the pointer to the current Gtk window.
- **host:** Real hostname of the server you connected to.
- **inputbox:** Contents of inputbox.
- **network:** Current network name or None.

- **nick:** Your current nick name.
- **nickserv:** Current networks password or None (**password** is the same with clearer name).
- **modes:** Current channel modes or None.
- **password:** Current networks password or None.
- **server:** Current server name (what the server claims to be) or None if you are not connected.
- **topic:** Current channel topic.
- **version:** HexChat version number.
- **win_status:** Returns status of window: ‘active’, ‘hidden’, or ‘normal’.

Example:

```
if hexchat.get_info("network") == 'freenode':
    hexchat.prnt('connected!')
```

You can also get the format of Text Events by using *event_text* and the event:

```
print(hexchat.get_info("event_text Channel Message"))
```

`hexchat.`**`get_prefs`**`(`*name*`)`
Retrieve the HexChat setting information specified by the `name` string, as available by the `/set` command.

```
print("Current preferred nick: " + hexchat.get_prefs("irc_nick1"))
```

A list of settings is provided here: *List of Settings*.

On top of that there are a few special preferences:

- **id**: unique server id
- **state_cursor**: location of cursor in input box

`hexchat.`**`get_list`**`(`*type*`)`
With this function you may retrieve a list containing the selected information from the current context, like a DCC list, a channel list, a user list, etc. Each list item will have its attributes set dynamically depending on the information provided by the list type.

The example below is a rewrite of the example provided with HexChat’s plugin API documentation. It prints a list of every DCC transfer happening at the moment. Notice how similar the interface is to the C API provided by HexChat.

```
list = hexchat.get_list("dcc")
if list:
    print("--- DCC LIST ------------------")
    print("File  To/From   KB/s   Position")
    for i in list:
        print("%6s %10s %.2f  %d" % (i.file, i.nick, i.cps/1024, i.pos))
```

Below you will find what each list type has to offer.

### List Types

**channels** The channels list type gives you access to the channels, queries and their servers. The following attributes are available in each list item:

- **channel:** Channel or query name.

- **channelkey:** Channel key. (2.9.6+)
- **chanmodes:** Channel modes e.g. beI,k,l. (2.12.2+)
- **chantypes:** Channel types e.g. #!&.
- **context:** A context object, giving access to that channel/server.
- **id:** Unique server id.
- **lag:** Latency in milliseconds.
- **maxmodes:** Max modes per line.
- **network:** Network name to which this channel belongs.
- **nickprefixes:** Nickname prefixes e.g. @%+.
- **nickmodes:** Nickname mode chars e.g. ov.
- **queue:** Number of bytes in the send-queue.
- **server:** Server name to which this channel belongs.
- **users:** Number of users in the channel.
- **type:** Type of context.
  - 1: Server
  - 2: Channel
  - 3: Dialog
  - 4: Notices
  - 5: SNotices
- **flags:** Bit field of flags:
  - 0: Connected
  - 1: Connecting
  - 2: Away
  - 3: End of MOTD (Login Complete)
  - 4: Has WHOX
  - 5: Has IDMSG
  - 6: Join/Parts hidden
  - 7: Join/Parts hidden unset
  - 8: Beep on Message
  - 9: Blink Tray
  - 10: Blink Task Bar
  - 11: Logging (This and the following are 2.10.0+)
  - 12: Logging unset
  - 13: Scrollback
  - 14: Scrollback unset
  - 15: Strip Colors

- 16: Strip Colors unset

**dcc** The dcc list type gives you access to a list of DCC file transfers. The following attributes are available in each list item:

- **address32:** Address of the remote user (ipv4 address, as an int).
- **cps:** Bytes per second (speed).
- **destfile:** Destination full pathname.
- **file:** Filename.
- **nick:** Nickname of person who the file is from/to.
- **port:** TCP port number.
- **pos:** Bytes sent/received.
- **resume:** Point at which this file was resumed (or zero if it was not resumed).
- **size:** File size in bytes.
- **status:** DCC status:
  - 0: queued
  - 1: active
  - 2: failed
  - 3: done
  - 4: connecting
  - 5: aborted
- **type:** DCC type:
  - 0: send
  - 1: receive
  - 2: chatrecv
  - 3: chatsend

**users** The users list type gives you access to a list of users in the current channel. The following attributes are available in each list item:

- **account:** Account name or None (2.9.6+)
- **away:** Away status.
- **host:** Host name in the form user@host (or None, if not known).
- **lasttalk:** Time they last talked (2.9.6+)
- **nick:** Nick name.
- **prefix:** Prefix character, .e.g: @ or +. Points to a single char.
- **realname:** Real name.
- **selected:** Selected status in the userlist.

**ignore** The ignore list type gives you access to the current ignored list. The following attributes are available in each list item:

- **mask:** Ignore mask (for example, "*!*@*.aol.com").
- **flags:** Bit field of flags:
  - 0: private
  - 1: notice
  - 2: channel
  - 3: ctcp
  - 4: invite
  - 5: unignore
  - 6: nosave
  - 7: dcc

**notify** The notify list shows users on your friends list and their status:

- **nick:** Users nickname
- **networks:** Networks they are setup to notify on (None for all)
- **flags:** 0 is offline, 1 is online
- **on:** Time when user last came on (2.9.6+)
- **off:** Time when user last logged off (2.9.6+)
- **seen:** Time when user was last seen (2.9.6+)

**Hook functions** These functions allow one to hook into HexChat events.

**Parameters**

**callback** A callback is the function that will be called when the event happens.

The callback supposed to return one of the EAT_* constants, it is able control how HexChat will proceed after the callback returns. These are the available constants, and their meanings:

- *EAT_PLUGIN*: Don't let any other plugin receive this event.
- *EAT_HEXCHAT*: Don't let HexChat treat this event as usual.
- *EAT_ALL*: Eat the event completely.
- *EAT_NONE*: Let everything happen as usual.

**Note:** Returning `None` is the same as returning *EAT_NONE*.

**userdata** The parameter userdata, if given, allows you to pass a custom object to your callback.

**attributes** If you create a hook with *hook_server_attrs()* or *hook_print_attrs()* the last argument in the callback will be an *Attribute* object.

**Attribute**

Attribute.**time**
The time the event occurred (from server-time) or 0

**priority** When a priority keyword parameter is accepted, it means that this callback may be hooked with five different priorities which are constants will define the order in which your plugin will be called. Most of the time, you won't want to change its default value (*PRI_NORM*).

**word and word_eol** These parameters, when available in a command or server callback, are lists of strings which contain the parameters the user entered for the particular command. For example, if you executed:

```
/command NICK Hi there!
```

- **word[0]** is `command`
- **word[1]** is `NICK`
- **word[2]** is `Hi`
- **word[3]** is `there!`
- **word_eol[0]** is `command NICK Hi there!`
- **word_eol[1]** is `NICK Hi there!`
- **word_eol[2]** is `Hi there!`
- **word_eol[3]** is `there!`

These parameters are also used in print events. When created by these events they have a completely different meaning though. Text events (*Settings* → *Text events*) have numbered arguments associated with them, these apply to the item in the **word** list. For example on a "Channel Message" event:

[23:29:26] <@Nick> hello everyone

- **word[0]** is `Nick`
- **word[1]** is `hello everyone`
- **word[2]** is `@`
- **word_eol[0]** is `Nick hello everyone @`
- **word_eol[1]** is `hello everyone @`
- **word_eol[2]** is `@`

hexchat.**hook_command**(*name*, *callback*[, *userdata=None*, *priority=PRI_NORM*, *help=None* ])

This function allows you to hook into the name HexChat command. It means that everytime you type `/name ...`, `callback` will be called. Parameters `userdata` and `priority` have their meanings explained above, and the parameter help, if given, allows you to pass a help text which will be shown when `/help name` is executed. If the command starts with a period it will not show up in `/help` though.

You may also hook an empty string to capture every message a user sends, either when they hit enter or use `/say`.

**Returns** New Hook Handler

```
def onotice_cb(word, word_eol, userdata):
    if len(word) < 2:
        print("Second arg must be the message!")
    else:
        hexchat.command("NOTICE @{} {}".format(hexchat.get_info("channel"), word_eol[1]))
    return hexchat.EAT_ALL

hexchat.hook_command("ONOTICE", onotice_cb, help="/ONOTICE <message> Sends a notice to all ops")
```

You may return one of `EAT_*` constants in the callback, to control HexChat's behavior, as explained above.

`hexchat.`**`hook_print`**(*name*, *callback*[, *userdata=None*, *priority=PRI_NORM* ])

This function allows you to register a callback to trap any print events. The event names are available in the *Settings → Text Events* window. Parameters `userdata` and `priority` have their meanings explained above.

**Parameters** **`name`** – event name (see *Settings → Text Events*)

**Returns** New Hook Handler

```
def youpart_cb(word, word_eol, userdata):
    print("You have left channel " + word[2])
    return hexchat.EAT_HEXCHAT # Don't let HexChat do its normal printing

hexchat.hook_print("You Part", youpart_cb)
```

Along with Text Events there are a handfull of *special* events you can hook with this:

- **Open Context**: Called when a new context is created.
- **Close Context**: Called when a context is closed.
- **Focus Tab**: Called when a tab is brought to front.
- **Focus Window**: Called a toplevel window is focused, or the main tab-window is focused by the window manager.
- **DCC Chat Text**: Called when some text from a DCC Chat arrives. It provides these elements in the word list:
  - Address
  - Port
  - Nick
  - Message
- **Key Press**: Called when some keys are pressed in the input box. It provides these elements in the word list:
  - Key Value
  - State Bitfield (shift, capslock, alt)
  - String version of the key
  - Length of the string (may be 0 for unprintable keys)

`hexchat.`**`hook_print_attrs`**(*name*, *callback*[, *userdata=None*, *priority=PRI_NORM* ])

This function is the same as *`hook_print()`* except its callback will have a new *Attribute* argument.

**Returns** New Hook Handler

New in version 1.0.

```
def youpart_cb(word, word_eol, userdata, attributes):
    if attributes.time: # Time may be 0 if server-time is not enabled.
        print("You have left channel {} at {}".format(word[2], attributes.time))
        return hexchat.EAT_HEXCHAT

hexchat.hook_print_attrs("You Part", youpart_cb)
```

`hexchat.`**`hook_server`**(*name*, *callback*[, *userdata=None*, *priority=PRI_NORM* ])

This function allows you to register a callback to be called when a certain server event occurs. You can use this to trap `PRIVMSG`, `NOTICE`, `PART`, a server numeric, etc. Parameters `userdata` and `priority` have their meanings explained above.

You can hook the special event "RAW LINE" to capture all server events.

**Returns** New Hook Handler

```
 def kick_cb(word, word_eol, userdata):
     print('{} was kicked from {} ({})'.format(word[3], word[2], word_eol[4]))
     # Don't eat this event, let other plugins and HexChat see it too
     return hexchat.EAT_NONE

hexchat.hook_server("KICK", kick_cb)
```

`hexchat.`**`hook_server_attrs`**(*name*, *callback*[, *userdata=None*, *priority=PRI_NORM* ])

This function is the same as *`hook_server()`* Except its callback will have a new *Attribute* argument.

**Returns** New Hook Handler

New in version 1.0.

```
def kick_cb(word, word_eol, userdata, attributes):
    if attributes.time: # Time may be 0 if server-time is not enabled.
        print('He was kicked at {}'.format(attributes.time))
        return hexchat.EAT_NONE

hexchat.hook_server_attrs("KICK", kick_cb)
```

`hexchat.`**`hook_timer`**(*timeout*, *callback*[, *userdata=None* ])

This function allows you to register a callback to be called every timeout milliseconds. Parameters userdata and priority have their meanings explained above.

**Returns** New Hook Handler

```
myhook = None

def stop_cb(word, word_eol, userdata):
    global myhook
    if myhook is not None:
        hexchat.unhook(myhook)
        myhook = None
        print("Timeout removed!")

def timeout_cb(userdata):
    print("Annoying message every 5 seconds! Type /STOP to stop it.")
    return 1 # Keep the timeout going

myhook = hexchat.hook_timer(5000, timeout_cb)
hexchat.hook_command("STOP", stop_cb)
```

If you return a true value from the callback, the timer will be kept, otherwise it is removed.

`hexchat.`**`hook_unload`**`(`*callback*[, *userdata=None* ]`)`

This function allows you to register a callback to be called when the plugin is going to be unloaded. Parameters `userdata` and `priority` have their meanings explained above.

**Returns** New Hook Handler

```
def unload_cb(userdata):
    print("We're being unloaded!")

hexchat.hook_unload(unload_cb)
```

`hexchat.`**`unhook`**`(`*handler*`)`

Unhooks any hook registered with the hook functions above.

**Parameters** **`handler`** – Handler returned from *`hook_print()`*, *`hook_command()`*, *`hook_server()`* or *`hook_timer()`*

As of version 1.0 of the plugin hooks from *`hook_print()`* and *`hook_command()`* can be unhooked by their names.

**Plugin preferences** You can use pluginpref to easily store and retrieve settings.

`hexchat.`**`set_pluginpref`**`(`*name*, *value*`)`

Stores settings in addon_python.conf in the config dir.

**Returns**

- False: Failure
- True: Success

New in version 0.9.

**Note:** Until the plugin uses different a config file per script it's recommened to use 'scriptname_settingname' to avoid conflicts.

`hexchat.`**`get_pluginpref`**`(`*name*`)`

This will return the value of the variable of that name. If there is none by this name it will return `None`.

**Returns** String or Integer of stored setting or None if it does not exist.

**Note:** Strings of numbers and booleans are always returned as Integers.

New in version 0.9.

`hexchat.`**`del_pluginpref`**`(`*name*`)`

Deletes the specified variable.

**Returns**

- False: Failure
- True: Success (or never existing),

New in version 0.9.

`hexchat.`**`list_pluginpref`**`()`

Returns a list of all currently set preferences.

**Return type** List of Strings

New in version 0.9.

**Context handling** Below you will find information about how to work with contexts.

**Context objects** As explained in the Context theory session above, contexts give access to a specific channel/query/server tab of HexChat. Every function available in the xchat module will be evaluated in the current context, which will be specified by HexChat itself before passing control to the module. Sometimes you may want to work in a specific context, and that's where *context* objects come into play.

You may create a context object using `get_context()` or `find_context()` functions as explained below, or trough the `get_list()` function, as explained above.

`hexchat.`**`get_context`**`()`

**Return type** *context*

`hexchat.`**`find_context`**`(`*server=None*, *channel=None*`)`

Finds a context based on a channel and servername or if no parameters are given returns the current (front) context.

**Parameters**

- **`server`** – if None only looks for channel name
- **`channel`** – if None looks for front context of given server

**Return type** *context*

```
cnc = hexchat.find_context(channel='#conectiva')
cnc.command('whois niemeyer')
```

**`context`**

The context object returned by the functions listed above has these methods:

`context.`**`set`**`()`

Changes the current context to be the one represented by this context object.

`context.`**`prnt`**`(`*string*`)`

Does the same as the `prnt()` function but in the given context.

`context.`**`emit_print`**`(`*event_name*, **args*`)`

Does the same as the `emit_print()` function but in the given context.

`context.`**`command`**`(`*string*`)`

Does the same as the `command()` function but in the given context

`context.`**`get_info`**`(`*type*`)`

Does the same as the `get_info()` function but in the given context.

`context.`**`get_list`**`(`*type*`)`

Does the same as the `get_list()` function but in the given context.

---

Maintained by: TingPing

Original Author: Gustavo Niemeyer gustavo@niemeyer.net

### 7.2.4 Building Perl modules on Windows

#### Building Perl modules on Windows

##### CPAN

###### Software

- Visual Studio 2012 Express for Windows Desktop + Visual Studio 2012 Update 3
- Perl x86 or x64
- MSYS (I'm linking to this version but you can use ANY MSYS, it's possible to use MSYS from MozillaBuild but it's really ancient)
- This script

###### Setup

You must paste my script to directory above your Perl installation. Script was written for Perl installed in MozillaBuild directory: C:\mozilla-build\perl-5.18\x64 or \x86, so script must be in perl-5.18 dir, if your Perl installation dir is in another place you must edit PATHs in lines 13 & 16.

MSYS can be extracted to any directory you want, you just need to edit PATH in *SET MSYS=I:\MSYS\bin*, fg. *SET MSYS=C:\MSYS\bin*

###### Usage

To use script you must open cmd (Win+R, type cmd), navigate to directory where you pasted cpan.bat. Type *cpan.bat x86* if you're using 32bit Hexchat & Perl or *cpan.bat x64* for 64bit version. After that you can use all cpan commands.

##### Old depreciated method

###### Software

To start building Perl modules you need to download and install this software (in their default install paths):

- MozillaBuild or direct link
- Visual Studio 2012 Express for Windows Desktop + Visual Studio 2012 Update 3
- Perl x86 or x64 (Perl **MUST** be installed to C:\Perl)

###### Downloading and Extracting

Start MozillaBuild console (it's in C:\mozilla-build) *start-msvc11.bat* (if you're using Hexchat x86) or *start-msvc11-x64.bat* (if you're using Hexchat x64).

Type:

```
mkdir perl; cd perl
```

(we will work in *perl* directory).

All Perl modules can be downloaded from CPAN site.

Now we can download the module which can be done using *wget http://link* command. Later we extract it using *tar -zxvf module.tar.gz*.

In this guide we will build Net::Telnet module. So we are downloading it and extracting:

```
wget http://search.cpan.org/CPAN/authors/id/J/JR/JROGERS/Net-Telnet-3.03.tar.gz
tar -zxvf Net-Telnet-3.03.tar.gz
```

Type *cd Net-Telnet-3.03* and we can start building.

### Note on Perl module dependencies

This building method isn't perfect so if you're building modue which depends on other module, you need to build it before repeating all steps in this guide.

### Building

Fist thing we must do is to configure module and point it to our Perl installation using:

```
c:/perl/bin/perl.exe Makefile.pl
```

Note: It really depends if it is Makefile.pl or Build.pl, just check in folder and read README.

You should get something like this:

```
Checking if your kit is complete...
Looks good
Writing Makefile for Net::Telnet
Writing MYMETA.yml and MYMETA.json
```

It means that module was properly configured. Now we can move to compilation which can be done using:

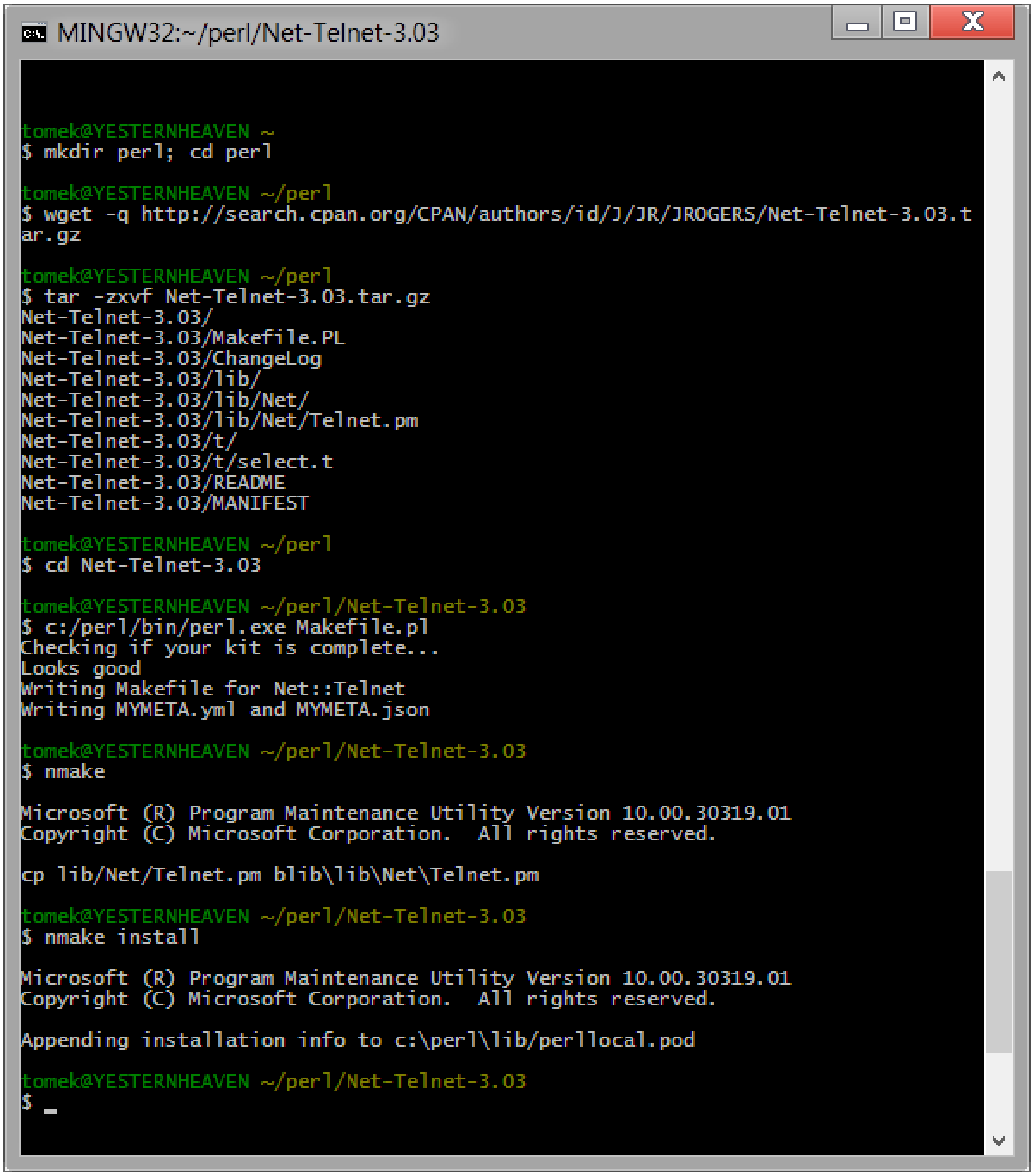
```
nmake
```

After it we can install module using:

```
nmake install
```

Now if you didn't get any error you should have Perl module installed and it can be used with Perl script in Hexchat.

Whole operation should look like this:



MINGW32:~/perl/Net-Telnet-3.03

```
tomek@YESTERNHEAVEN ~
$ mkdir perl; cd perl

tomek@YESTERNHEAVEN ~/perl
$ wget -q http://search.cpan.org/CPAN/authors/id/J/JR/JROGERS/Net-Telnet-3.03.t
ar.gz

tomek@YESTERNHEAVEN ~/perl
$ tar -zxvf Net-Telnet-3.03.tar.gz
Net-Telnet-3.03/
Net-Telnet-3.03/Makefile.PL
Net-Telnet-3.03/ChangeLog
Net-Telnet-3.03/lib/
Net-Telnet-3.03/lib/Net/
Net-Telnet-3.03/lib/Net/Telnet.pm
Net-Telnet-3.03/t/
Net-Telnet-3.03/t/select.t
Net-Telnet-3.03/README
Net-Telnet-3.03/MANIFEST

tomek@YESTERNHEAVEN ~/perl
$ cd Net-Telnet-3.03

tomek@YESTERNHEAVEN ~/perl/Net-Telnet-3.03
$ c:/perl/bin/perl.exe Makefile.pl
Checking if your kit is complete...
Looks good
Writing Makefile for Net::Telnet
Writing MYMETA.yml and MYMETA.json

tomek@YESTERNHEAVEN ~/perl/Net-Telnet-3.03
$ nmake

Microsoft (R) Program Maintenance Utility Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.

cp lib/Net/Telnet.pm blib\lib\Net\Telnet.pm

tomek@YESTERNHEAVEN ~/perl/Net-Telnet-3.03
$ nmake install

Microsoft (R) Program Maintenance Utility Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.

Appending installation info to c:\perl\lib/perllocal.pod

tomek@YESTERNHEAVEN ~/perl/Net-Telnet-3.03
$ _
```

# Index