Open Source IRC Client for Windows and Linux

User Manual

Author	Daniel Hetrick
License	GPL 3 (explained)
Source Code	https://github.com/nutjob-laboratories/merk
Last modified	Thursday, July 10, 2025

Summary

IRC (**Internet Relay Chat**) is a text-based chat system for instant messaging. IRC is designed for group communication in discussion forums, called *channels*,[1] but also allows one-on-one communication via private messages[2]… .

Internet Relay Chat is implemented as an application layer protocol to facilitate communication in the form of text. The chat process works on a client–server networking model. Users connect, using a client—which may be a web app, a standalone desktop program, or embedded into part of a larger program—to an IRC server, which may be part of a larger IRC network. Examples of ways used to connect include the programs Mibbit, KiwiIRC, mIRC and the paid service IRCCloud.

From the Wikipedia entry on IRC, at https://en.wikipedia.org/wiki/IRC

MERK is a free and open source Internet Relay Chat client for Windows and Linux. It uses a "multiple document interface", in which the application works as a parent window that contains other windows for servers, channels, and private chats. The popular Windows shareware client mIRC is an example of another IRC client that uses a multiple document interface.

MERK is written in the Python programming language, using the PyQt library for the graphical interface and the Twisted library for networking. MERK also comes bundled with three other open source libraries:

- **qt5reactor**, for getting PyQt and Twisted to work together
- **pyspellchecker**, which provides the spellchecking mechanism
- **emoji**, providing support for emoji shortcodes

MERK has been designed with three goals: to have a modern look (in comparison to other open source and free client, such as XChat/HexChat), to take advantage of technological and cultural advances in text-based messaging, and to make as much of the user interface and operation of the application as configurable as possible.

MERK has been in development since 2019, and is being developed by Dan Hetrick, who also wrote this document.

Directories and Configuration Files

MERK stores all its settings in a directory it creates in the user's home directory, named **`.merk`**. Inside this directory, MERK creates three more directories:

- **`logs`**. This is where MERK stores channel and private chat logs.
- **`styles`**. This is where MERK stores text style files, and the palette used for dark mode.
- **`scripts`**. This is where MERK stores, and first looks for, scripts. This is the default directory chosen when running a script via the server window toolbar, input menu, or right click menus, or when saving a script in the editor.

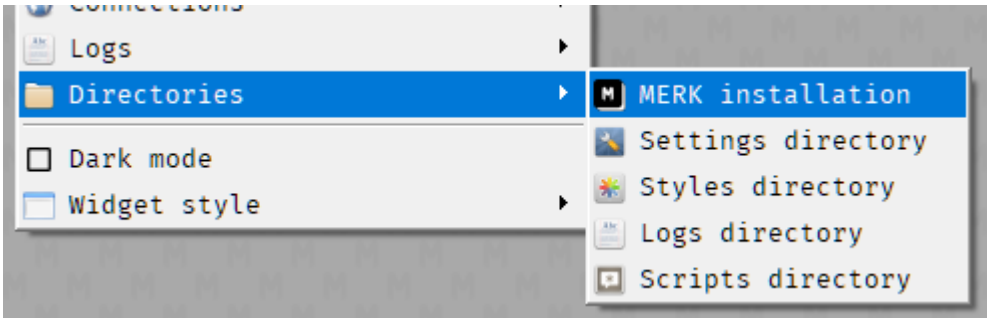
MERK also creates two other files in this directory, **`settings.json`** and **`user.json`**:

- **`settings.json`**. Where MERK stores and loads application settings.
- **`user.json`**. Where MERK stores user information, such as the chosen nickname, username, and the like, as well as the application's connection history and any connection scripts.

When using the **`/script`** command, if a full filename is not provided, MERK will look for the script in several locations, in order:

1. The **`scripts`** directory.
2. The settings directory (by default, **`.merk`** in the user's home directory).
3. The application's installation directory.

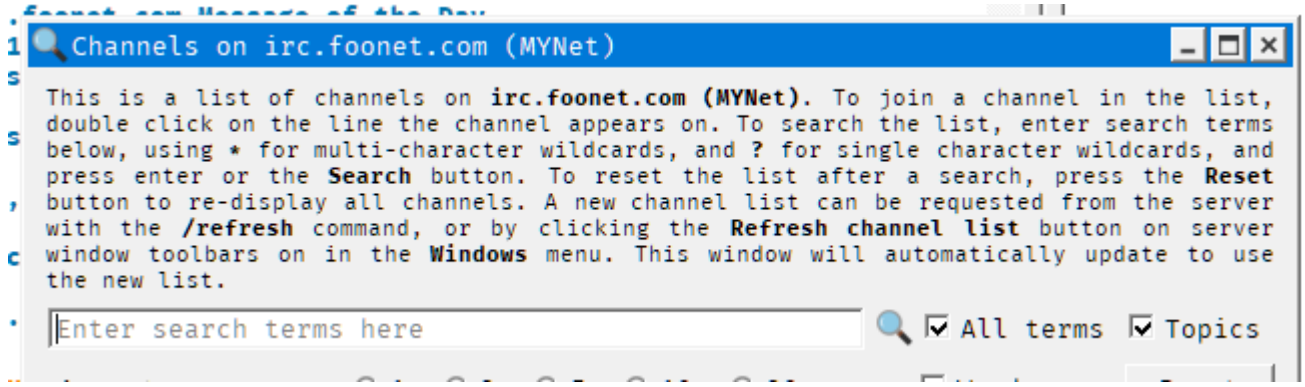
First, MERK will attempt to find the script using the provided filename, and if the script is still not found, it will append the default file extension (which is **`.merk`**) to the filename and search again.



These folders can be opened in your default file manager from the client by clicking on the appropriate entry in the "Directories" submenu, near the bottom of the "Settings" menu.

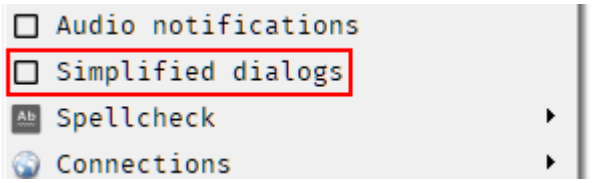
New User Help

Most dialogs feature text explaining how the dialog or the settings in it work.

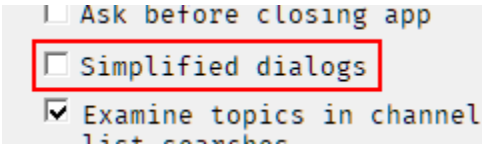


The explanation text from the channel list dialog.

While new users of MERK may find these helpful, experienced users may not want to see them. To hide the help text on dialogs, turn on "Simplified dialogs" in the settings menu or the settings dialog:



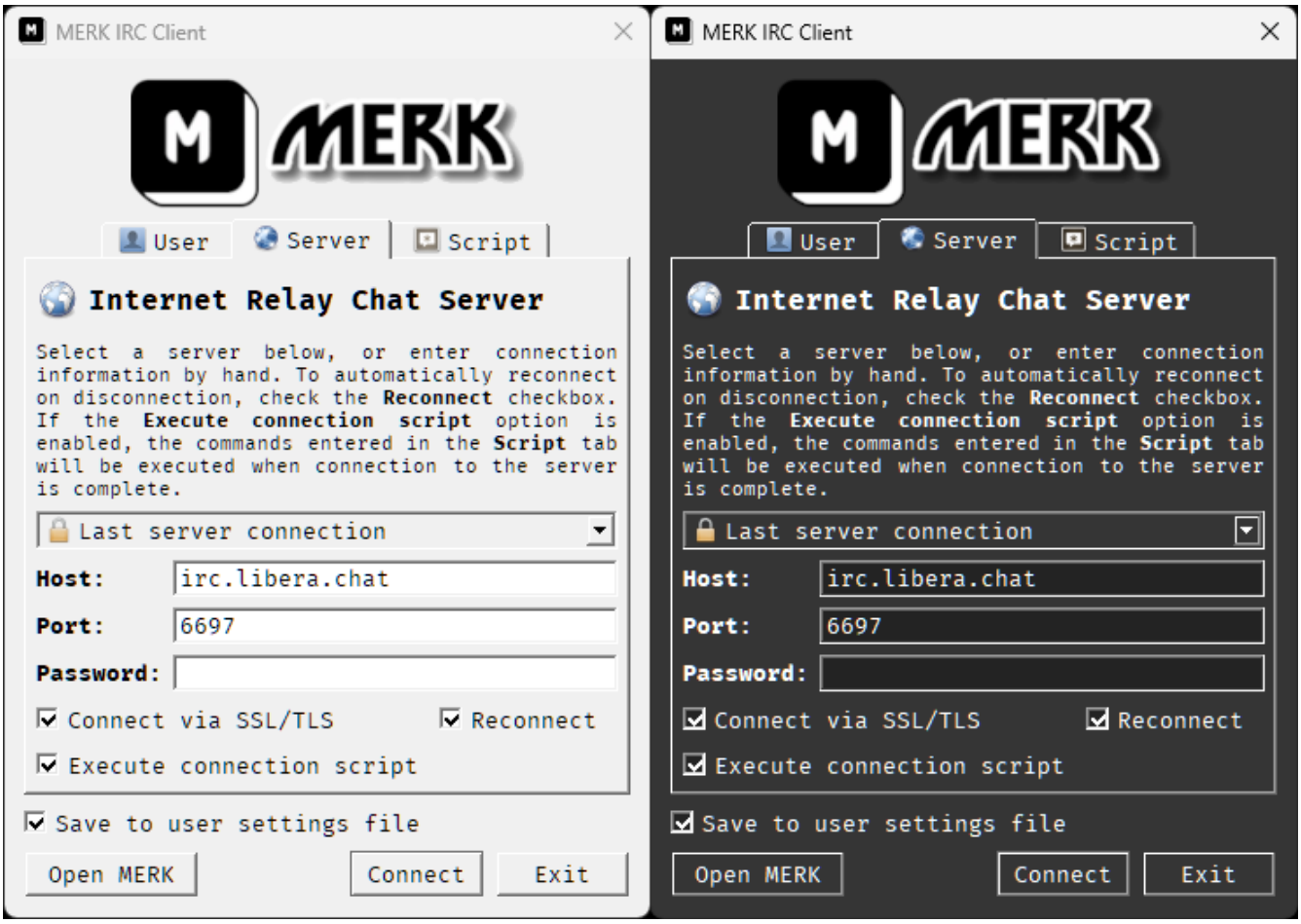
The "Simplified dialogs" option in the "Settings" menu. Click this entry to simplified dialogs on and hide the help text.



The "Simplified dialogs" option on the first page of the "Settings" dialog.

"Simplified dialogs" is turned off by default, showing the help text on dialogs every time a dialog is opened.

Regular and Dark Mode



Normal Mode *Dark Mode*

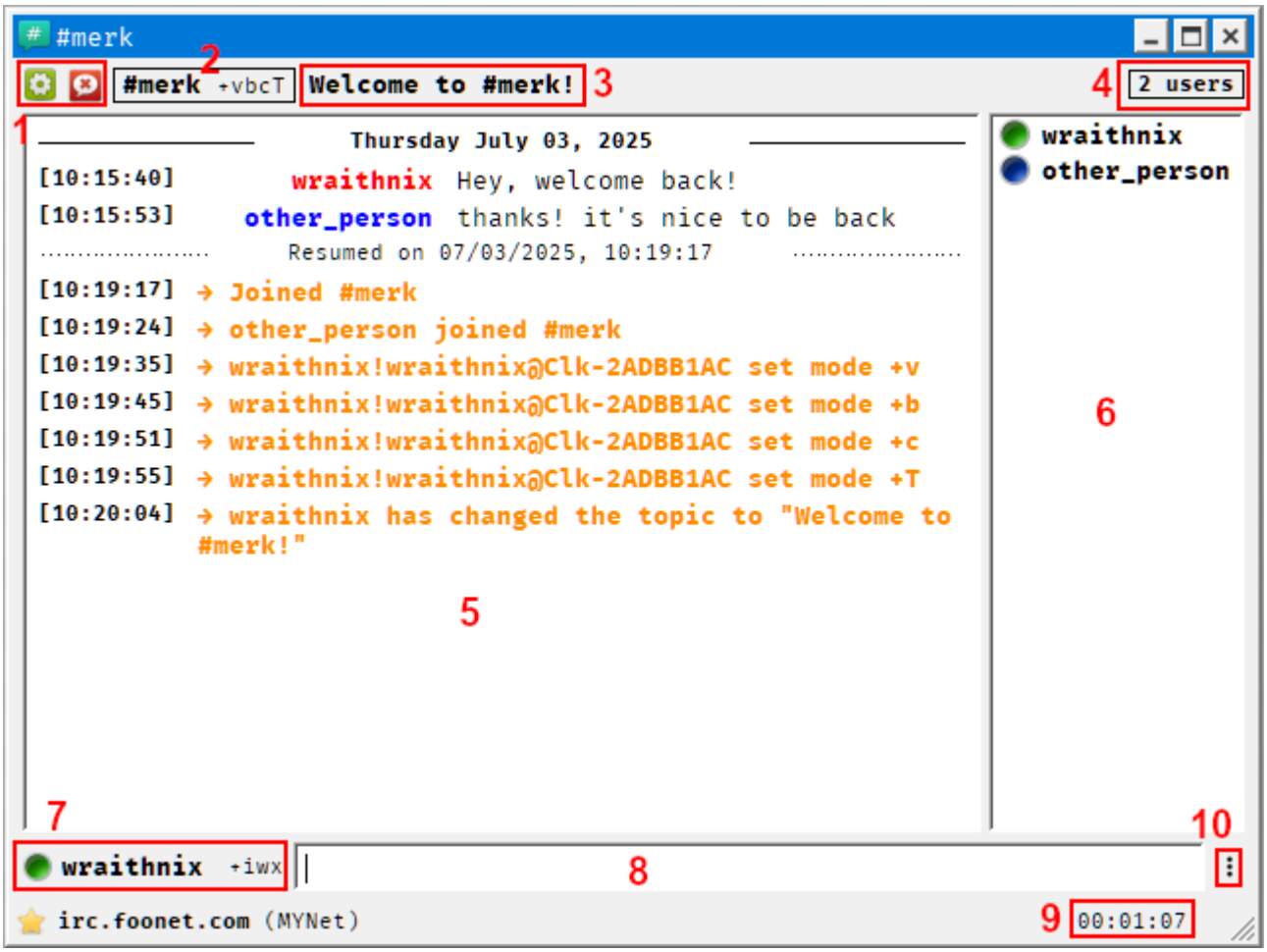
MERK can be operated in "normal" mode, seen above to the left, or in "dark" mode, on the right. Other than using a darker palette for backgrounds and a lighter color for text, both modes operate identically.

To switch to "dark" mode, select it in the "Settings" menu or in the "Settings" dialog. MERK will have to be restarted for it to take effect, and you'll be prompted to restart MERK automatically.

For more advanced users, if you want to edit the palette that "dark" mode uses, all of the colors used by the application are stored in a file in the **`styles`** directory named **`dark.palette`**. The file format is specific to MERK, but you can edit it with a text editor. All colors are stored in the hexadecimal format used by HTML. To re-create the file and reset the "dark" mode values back to the default, delete **`dark.palette`** and restart MERK; the application will regenerate the file with default values.

Channel Windows

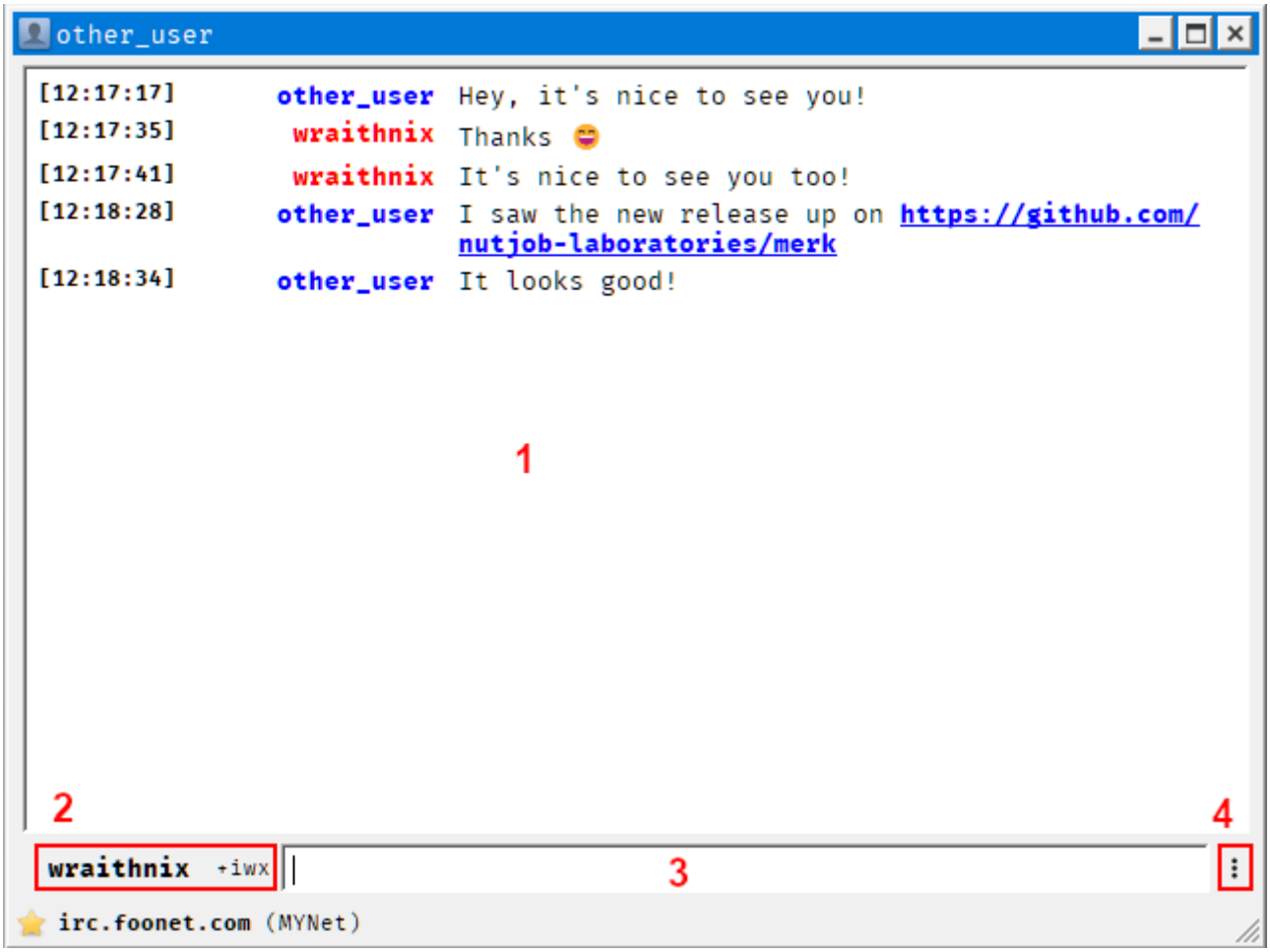
Closing a channel window leaves the channel.



1. **Mode Editor** and **Banlist**. The mode editor button displays a menu that allows the user to set or remove popular channel modes, if their status allows it; if they are not a privileged enough user, the button is hidden. The banlist displays a list of users that have been banned from the channel; if the banlist is empty, the button is hidden.
2. **Name and mode display**. Here, the channel name and any channel modes are displayed.
3. **Topic**. The channels topic is displayed here. Click on the topic to edit it, and press enter to send any changes to the server.
4. **User count**. How many users are currently in the channel
5. **Chat display**. Channel chat, as well as system messages, are displayed here.
6. **User list**. A list of users in the channel is displayed here. Privileged users have special icons next to their name (green for channel operators, blue for voiced users, etc.), and normal users do not. Nicknames are displayed in bold if the users are present, and in normal weight if they are away. Double click a user's name to open a private chat window.
7. **Nickname**. This displays the currently used nickname, and any user modes set.
8. **Text input widget**. Type your chat or commands here, and press "enter" to send them to the server or client.
9. **Uptime**. This displays how long the client has been connected to the channel.
10. **Input menu**. Clicking on this brings up a menu that allows you to do various tasks, like changing the spellchecker's language.

Private Chat Windows

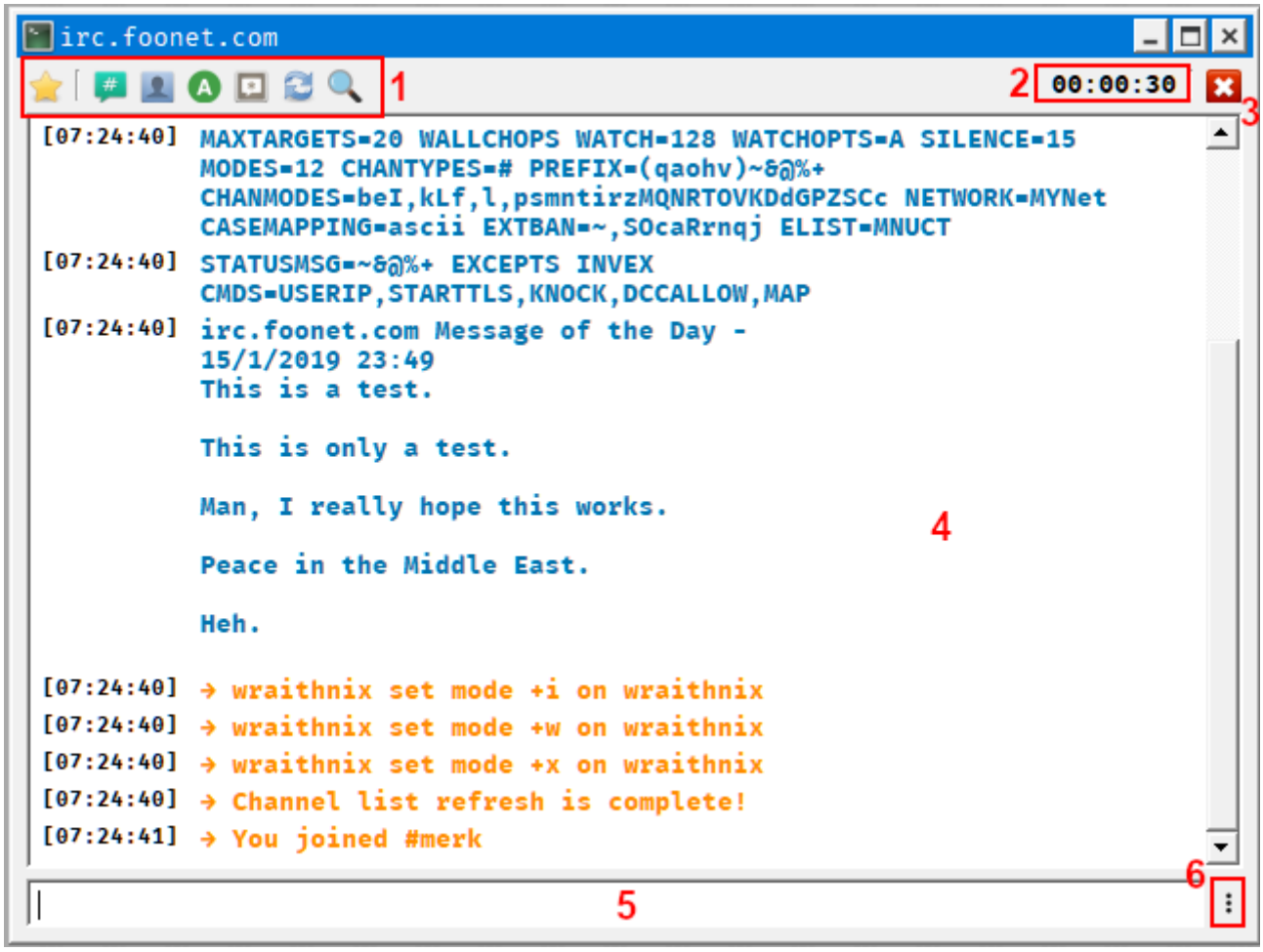
Closing a private chat window does not leave the chat, or block the sender; it only closes the window.



1. **Chat display**. Private chat is displayed here, as well as system messages.
2. **Nickname**. This displays the currently used nickname, and any user modes set.
3. **Text input widget**. Type your chat or commands here, and press "enter" to send them to the server or the client.
4. **Input menu**. Clicking on this brings up a menu that allows you to do various tasks, like changing the spellchecker's language.

Server Windows

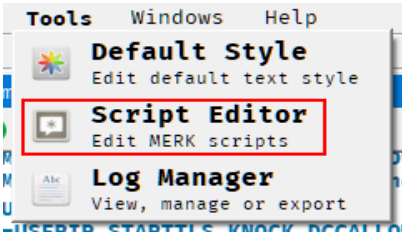
Server windows behave differently from channels or private chat windows: closing a server window does not disconnect from the server, it only hides the window. To view a hidden server window, click on its entry in the "Windows" menu or the system tray menu.



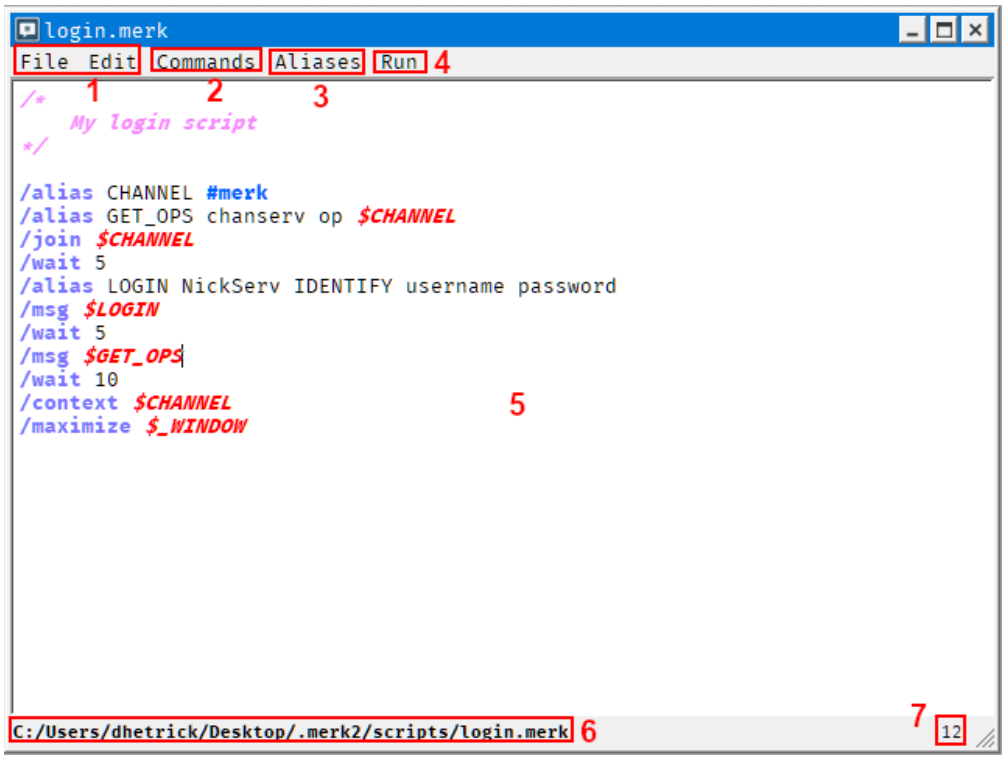
1. **Toolbar**. Buttons that perform basic actions; some on the IRC server, such as joining a channel, changing your nickname, and setting your away status, and others on the client, like selecting a script to run, refreshing the channel list from the server, and opening the channel list dialog.
2. **Connection uptime**. This displays how long MERK has been connected to the server.
3. **Disconnect**. Pressing this button issues a **`QUIT`** command and quickly disconnects from the IRC server.
4. **Display**. Displays any messages from the server, as well as notices, outgoing private messages, and the like.
5. **Text input widget**. Type commands in here, and press "enter" to execute them.
6. **Input menu**. Clicking on this brings up a menu that allows you to do various tasks, like changing the spellchecker's language. This button is present on channel or private message windows, too.

Script Editor

To launch the script editor, use the /edit command, or select **Script Editor** from the "Tools" menu:



The script editor is a basic text editor with a few special features:

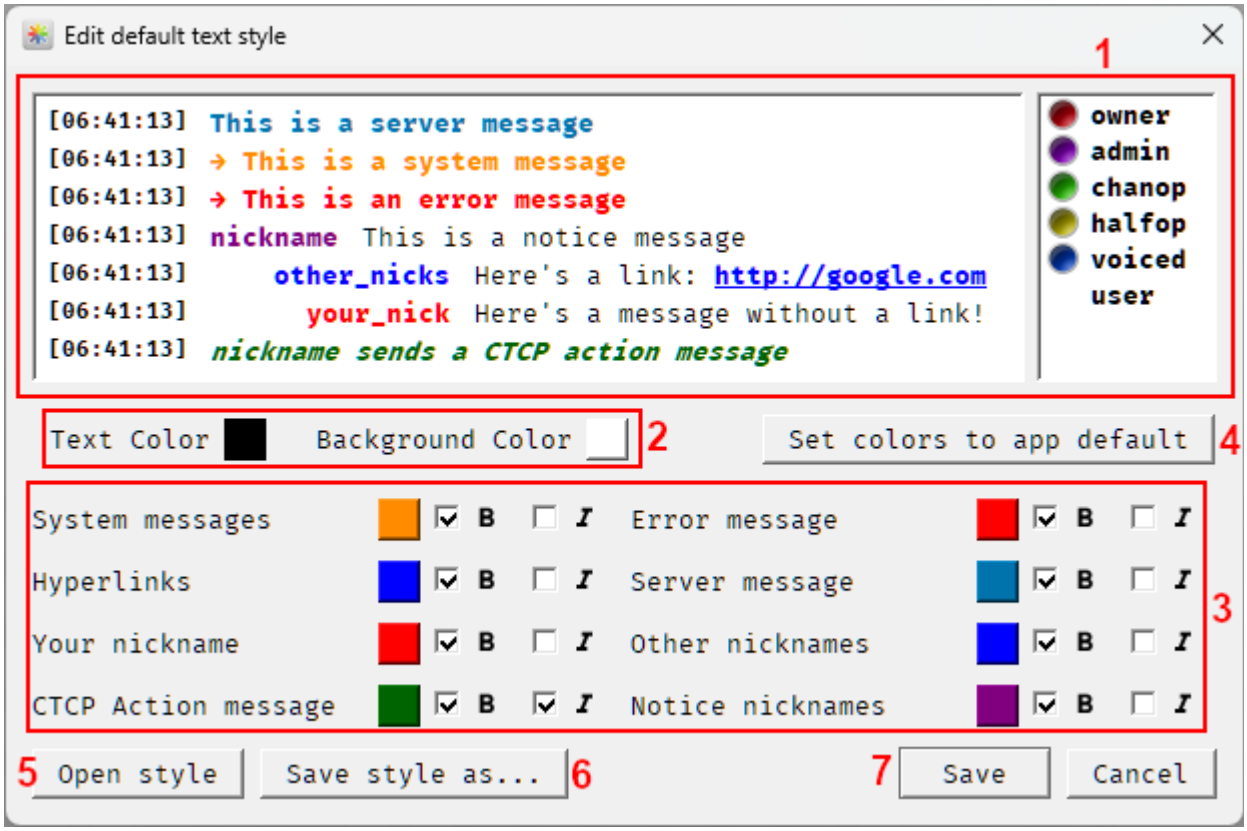


An example script open in the script editor.

1. **File** and **Edit Menus**. All the normal selections of a text editor, like opening and saving files, cut and paste, find and replace, etc. Connection scripts can also be opened for editing, as well as created.
2. **Commands**. Each entry in this menu allows the user to insert a command into the open script. Click the desired command, fill out the entries in the dialog that pops up, if needed, and the command is inserted into the script.
3. **Aliases**. Insert built-in aliases into the script.
4. **Run**. Run the currently open script in any context/window available. The user also can run the script in all contexts/windows simultaneously.
5. **Script display**. Features syntax highlighting. Colors used for the display can be set in the settings dialog.
6. **Filename**. The currently open script's filename is displayed here.
7. **Line number**. The line number the cursor is currently on.

Text Styles

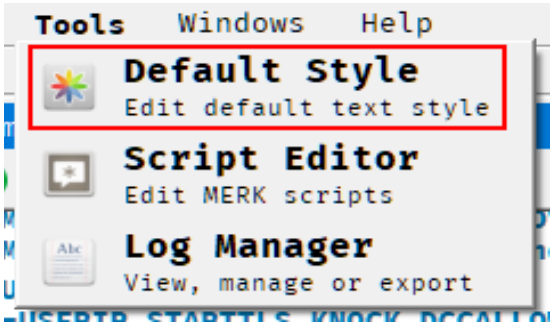
MERK has a text style engine that colors and styles all chat text, and can be edited by end users with the style editor:



1. **Display**. This is what the text style will look like in the client. Any changes in color or style will be displayed here instantly.
2. **Background and foreground color**. Set the color of the text and the background color here.
3. **Message styles**. Change the color and style of individual message types here.
4. **Default style**. Set all colors to the default style that ships with MERK. This is different from the "default" style that is applied to server windows and any windows that do not have a style.
5. **Open style**. Here, you can open any existing MERK style file for editing. Colors and styles will be loaded and displayed.
6. **Save as…**. Save this style to a file. It will not be applied, only saved to a file.
7. **Save** and **Cancel**. Saving this file also automatically applies it. Pressing the "cancel" button closes the dialog, and all changes are discarded.

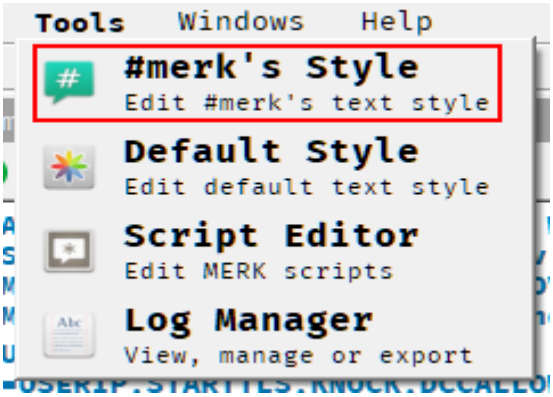
How Styles Are Applied

All chat windows start by using the default style, which can be edited by selecting "Edit default text style" in the "Tools" menu.



All chat windows can have their own styles which can be edited by selecting the "Edit text style" option from the "Tools" menu, the input options menu, or the chat display right click menu. Styles for channel and private chat windows are saved with the IRC network of the channel or private chat in mind, so they will load no matter which server the client is connected to. For example, if the user has set a text style for the **#merk** channel on the EFnet network, it will load and be applied to the **#merk** channel window if the user is connected to **irc.underworld.no** on port 6667, **irc.choopa.net** on port 9999, or **irc.prison.net** on port 6667, as all of these servers are on the EFnet IRC network.

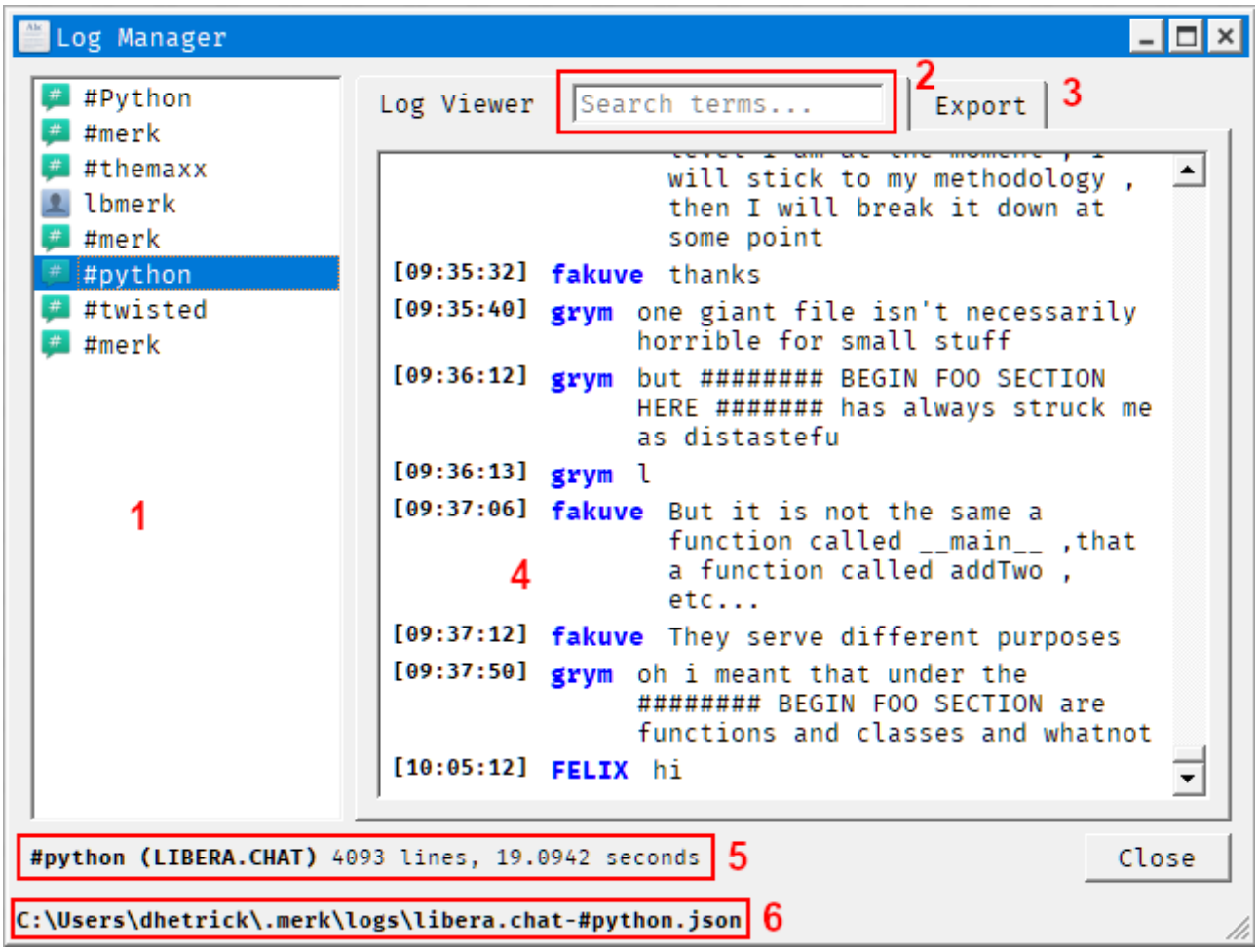
Server window text styles are specific to the server being connected to, regardless of what network the server is on. So, if the user has set a style for **irc.prison.net** on port 6667, the server window text style for that connection will be loaded. If the user connects to **irc.choopa.net** on port 9999, the default style will be loaded; even though both servers on the EFnet network, they are still different servers.



The top entry in the "Tools" menu will be the option to open the text style editor for the current window, if that window's style can be edited. Here, we see the option to edit the style for the **#merk** IRC channel.

Log Manager

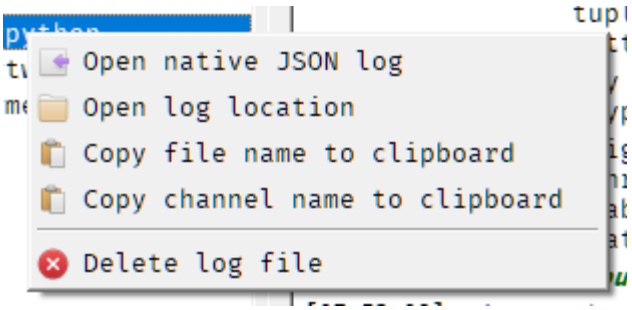
The log manager allows users to view, delete, and export MERK logs.



1. **Logs**. A full list of all logs in MERK. Hover the mouse over the log name to see what IRC network the log is from. Click a log name to view that log in full, as well as use export options.
2. **Search**. Search the log for specific words. Typing in the terms and pressing enter will find the first instance of the term in the log; hitting enter again will find the next, and so on.
3. **Log Viewer** and **Export Tabs**. Click tabs to switch functions. The **Export** tab has settings and functionality to export MERK logs to JSON or a custom delimited format.
4. **Log display**. Logs are loaded in full for viewing. For longer logs, this may take some time. Logs use the whatever default text style the user has set.
5. **Log information**. Contains the log name, what IRC network the log is from, how many lines of chat the log contains, and how long it took to render the entire log.
6. **Log filename**. The full filename of the current log.

Click the "Close" button to close the log manager.

Additional log options can be seen by right-clicking on the log's name:



If the "Open native JSON log" option is selected, the JSON file that MERK uses will be opened in the default application that the user's operating system to open JSON files. There is additional information in the native log format that tells MERK how to render the log for viewing; to use MERK logs with other applications, the log should be exported to strip this information out.

Commands and Scripting Guide

Special Commands

- **`/wait`** – ***Can only*** *be called from* ***scripts***
- **`/context`** – ***Can only*** *be called from* ***scripts***
- **`/end`** – ***Can only*** *be called from* ***scripts***
- **`/usage`** – ***Can only*** *be called from* ***scripts***
- **`/style`** – ***Cannot*** *be called from* ***scripts***
- **`/log`** – ***Cannot*** *be called from* ***scripts***
- **`/settings`** – ***Cannot*** *be called from* ***scripts***
- **`/edit`** – ***Cannot*** *be called from* ***scripts***
- **`/focus`** – ***Cannot*** *be called from* ***scripts****; use* **`/context`** *instead*

This is a list of commands that can be issued in either the text input widget in the client, or in scripts. There are four commands, however, that can only be issued in scripts: **`/wait`**, **`/context`**, **`/usage`**, and **`/end`**. These four commands *cannot* be used in the text input widget. Several other commands cannot be called by a script, and can only be used in the text input widget: **`/edit`**, **`/style`**, **`/log`**, **`/focus`**, and **`/settings`** will display an error and prevent execution if called by a script. To "move" to another window in a script, use **`/context`** instead of **`/focus`**.

The /**`ignore`** command hides chat from a given nickname or user. However, messages from an /**`ignore`**d user are still received and logged, they are just not displayed. That user's chat is hidden from *all* chat displays, no matter what server they are on. You can pass a nickname (which will hide all chat from any user with that nickname) or a hostmask (which will hide chat from only users with that hostmask) to the /**`ignore`** command. The /**`ignore`** list is saved to the configuration file, and will be applied universally until the user is /**`unignore`**d.

The **`/print`** command can be used to print text to the current or another window; use the name of the window context as the first argument to print to another window. If this window cannot be found, the text will print to whatever the current window context is.

```
/* This will print to the current window */
/print Hello world!

/* This will print to the #merk channel window.
   If the client is not in #merk, it will print to the current window. */
/print #merk Hello world!
```

Command List

All Commands

Commands in the list with a gray background are IRC or IRC server related commands, while those with a white background are MERK-specific commands.

Command	Description
/away [MESSAGE]	Sets status as "away"
/back	Sets status as "back"
/ctcp USER REQUEST	Sends a CTCP request to a user. Valid requests are TIME, VERSION, or FINGER
/invite NICKNAME CHANNEL	Sends a channel invitation
/join CHANNEL [KEY]	Joins a channel
/kick CHANNEL NICKNAME [MESSAGE]	Kicks a user from a channel
/knock CHANNEL [MESSAGE]	Requests an invitation to a channel
/list [TERMS]	Lists or searches channels on the server; use "*" for multi-character wildcard and "?" for single character
/me MESSAGE...	Sends a CTCP action message to the current chat
/mode TARGET MODE...	Sets a mode on a channel or user
/msg TARGET MESSAGE...	Sends a message
/nick NEW_NICKNAME	Changes your nickname
/notice TARGET MESSAGE...	Sends a notice
/oper USERNAME PASSWORD	Logs into an operator account
/part CHANNEL [MESSAGE]	Leaves a channel
/ping USER [TEXT]	Sends a CTCP ping to a user
/quit [MESSAGE]	Disconnects from the current IRC server
/raw TEXT...	Sends unprocessed data to the server
/refresh	Requests a new list of channels from the server
/time	Requests server time
/topic CHANNEL NEW_TOPIC	Sets a channel topic
/version [SERVER]	Requests server version
/who NICKNAME [o]	Requests user information from the server
/whois NICKNAME [SERVER]	Requests user information from the server
/whowas NICKNAME [COUNT] [SERVER]	Requests information about previously connected users
/alias	Prints a list of all current aliases
/alias TOKEN TEXT...	Creates an alias that can be referenced by **$TOKEN**
/cascade	Cascades all subwindows
/clear [WINDOW]	Clears a window's chat display
/config [SETTING] [VALUE...]	Changes a setting, or searches and displays one or all settings in the configuration file

Command	Description
/connect SERVER [PORT] [PASSWORD]	Connects to an IRC server
/connectssl SERVER [PORT] [PASSWORD]	Connects to an IRC server via SSL
/context WINDOW_NAME	Moves execution of the script to **WINDOW_NAME**; can only be called from scripts, and should be used in scripts rather than **/focus**
/edit [FILENAME]	Opens a script in the editor; if called without an argument, opens an editor window
/end	Immediately ends a script. Can only be called from scripts.
/exit [SECONDS]	Exits the client, with an optional pause of **SECONDS** before exit
/find [TERMS]	Finds filenames that can be found by other commands, like **/script** or **/edit**. If called without any arguments, **/find** will list all files visible to commands. Can use ***** for multi-character wildcards and **?** for single character wildcards.
/focus [SERVER] WINDOW	Switches focus to another window. Cannot be called from scripts (use **/context** instead)
/help [COMMAND]	Displays command usage information
/ignore USER	Hides a **USER**'s chat in all chat windows. This can be set to a nickname or hostmask. Capitalization is ignored.
/log	Opens the log manger. Cannot be called from a script
/maximize [SERVER] WINDOW	Maximizes a window
/minimize [SERVER] WINDOW	Minimizes a window
/play FILENAME	Plays a WAV file
/print [WINDOW] TEXT...	Prints text to a window
/restore [SERVER] WINDOW	Restores a window
/script FILENAME [ARGUMENTS]	Executes a list of commands in a file
/settings	Opens the settings dialog. Cannot be called from a script
/shell ALIAS COMMAND...	Executes an external program, and stores the output in an alias
/style	Edits the current window's style. Cannot be called from a script
/tile	Tiles all subwindows
/unalias TOKEN	Deletes the alias referenced by **$TOKEN**
/unignore USER	Un-hides a **USER**'s chat in all chat windows. This can be set to a nickname or hostmask. Capitalization is ignored. To un-hide all users, use ***** as the argument.
/usage NUMBER [MESSAGE...]	Prevents a script from running unless **NUMBER** arguments are passed to it, and displays **MESSAGE**. Can only be called from scripts

Command	Description
/wait SECONDS	Pauses script execution for SECONDS; can only be called from scripts
/xconnect SERVER [PORT] [PASSWORD]	Connects to an IRC server & executes connection script
/xconnectssl SERVER [PORT] [PASSWORD]	Connects to an IRC server via SSL & executes connection script

Script-Only Commands

These commands can only be called from scripts. Attempts to use them in the text input widget will fail and show an error.

Command	Description
/context WINDOW_NAME	Moves execution of the script to **WINDOW_NAME**; can only be called from scripts, and should be used in scripts rather than **/focus**
/end	Immediately ends a script. Can only be called from scripts.
/usage NUMBER [MESSAGE...]	Prevents a script from running unless **NUMBER** arguments are passed to it, displaying **MESSAGE**. Can only be called from scripts.
/wait SECONDS	Pauses script execution for SECONDS; can only be called from scripts

Non-Script Commands

These commands cannot be called from scripts. Scripts that use these commands will prevent the script from being executed, and show an error.

Command	Description
/edit [FILENAME]	Opens a script in the editor; if called without an argument, opens an editor window. Cannot be called from a script
/focus [SERVER] WINDOW	Switches focus to another window. Cannot be called from a script (use **/context** instead)
/log	Opens the log manger. Cannot be called from a script
/settings	Opens the settings dialog. Cannot be called from a script
/style	Edits the current window's style. Cannot be called from a script

Context-less Commands

These commands can be called without specifying the channel, chat, or window they are for. They will run in the current context. Commands with a gray background are IRC specific commands.

Command	Description
`/clear`	Clears the current window's chat display
`/invite NICKNAME`	Sends a channel invitation to the current channel
`/kick NICKNAME [MESSAGE]`	Kicks a user from the channel
`/me MESSAGE...`	Sends a CTCP action message to the current chat
`/mode MODE...`	Sets a mode on the current channel
`/part [MESSAGE]`	Leaves the channel
`/topic NEW_TOPIC`	Sets the current channel's topic

Scripting MERK

There are two types of scripts in MERK: connection scripts, and all other scripts.

Connection Scripts

Connection scripts are the scripts entered into the connection dialog, and are intended to be executed as soon as the client connects to the server. Unlike other scripts, they are stored in the user configuration file, and, outside of connection, can only be executed with the script editor. Connection scripts do *not* have a context (see ***Context*** and ***Writing Connection Scripts*** below).

All Other Scripts

All other scripts are, well, *scripts*: a list of commands, one per line, issued in order. Scripts have a context, which is the window that they are called from or executed in. They can be executed in several ways:

- From the "**Run**" button on a server window's toolbar. The script will be executed in the server window's context.
- From the "**Run**" entry in a window's input menu. The script will be executed in that window's context.
- From the "**Run**" entry in a window's chat display right-click menu. The script will be executed in that window's context.
- By issuing the **`/script`** command. The script will be executed in the window that the command was called from's context.
- From the "**Run**" menu in a script editor window. The user can select which context to run the script in, or optionally select to run the script on *all* windows simultaneously (with each window running that script in the window's context).

Scripts can have comments. Comments must begin with **`/*`** and end with **`*/`**, and can span multiple lines. Commands issued within comment blocks will be ignored, as will any text inside the comment block:

```
/*
/msg $_WINDOW This command WILL NOT be executed
*/

/msg $_WINDOW This command WILL be executed
```

Commands in scripts should be issued at the *start* of a line, with no spaces or tabs in front of them. This is not enforced, however, and scripts with spaces or tabs at the beginning of a line will execute normally.

Errors

For the most part, MERK ignores errors in scripts, with four exceptions: calling **`/wait`** with a non-number argument, trying to **`/context`** to a window/context that does not exist, calling **`/usage`** in a script with the wrong number of arguments to it or a non-number first argument, and calling **`/end`** with too many arguments. Any of these will halt script execution immediately. Every other error will display an error message (if script error messages are turned on in settings), and continue execution of the script. Error messages will be displayed for:

- Lines that do not contain a command
- Lines that start with `/` and are not followed by a valid command
- Calls to commands with an incorrect number of arguments
- Calls to commands with invalid arguments

If an error is encountered, an error message with be displayed. The error message will contain:

- The line number the error occurred on
- A description of the error

After the error is displayed, ***the script will continue to execute***. Scripts run in an external process, and cannot be halted (with the two exceptions described above). Please write, read, and test your scripts carefully.

Aliases

Aliases are tokens that can be created to insert specific strings into your input in the client (if the **Interpolate aliases into input** setting is turned on, which is the default) or into your scripts. For example, let's create an alias named 'GREETING', and set it to the value 'Hello world!':

```
/alias GREETING Hello world!
```

Now, if you want to insert the string "Hello world!" into a command or any output, you can use the alias interpolation symbol, which is **$** by default, followed by the aliases name, to insert your alias into the command or output:

```
/msg #mychannel $GREETING
```

This sends a message to `#mychannel` that says "Hello world!" to everyone in the channel!

Alias names *must* start with a letter, and not a number or other symbol. This is to prevent overwriting built-in aliases created for each window's context (see ***Built-In Aliases***).

To create an alias, use the **`/alias`** command. To see a list of all aliases set for the current window, issue the **`/alias`** command with no arguments. To delete an alias, issue the **`/unalias`** command.

Aliases can also be used as macros, and can contain an entire command. For example, let's say that you like to issue a greeting to everyone that enters a channel, but typing **`/msg #mychannel Hello, and welcome!`** is a pain to type every time someone joins, you could create this alias:

```
/alias GREETING /msg $_WINDOW Hello, and welcome!
```

Now, whenever someone joins your channel, just type **`$GREETING`** into the text input widget to send your message! The above example uses a built-in alias, which is explained in ***Built-In Aliases***.

All aliases are *global*; that is, they are available to all and every script executed on the client after they are created. They can also be changed by any script or command. Aliases created by connection scripts will be available and visible to any scripts executed after the connection script (including other connection scripts). Any script can also delete an alias with the **`/unalias`** command.

Built-in aliases cannot be deleted with the **`/unalias`** command. The client will display an error that says the alias doesn't exist if attempted.

Built-In Aliases

Each window has a number of aliases for use that are built-in to the window's context, and do not require the user to create them. Built-in alias names start with an underscore (_) and are all uppercase.

Alias	Value
_HOST	The reported hostname of the server the window is connected to; if that is not know, then this will be set to the server's address, a colon, and the server's port.
_MODE	Any modes set on the user associated with the window.
_NICKNAME	The user's nickname.
_PORT	The port on the server the window is connected to
_PRESENT	If the window the alias is being used in is a channel window, this will contain a list of users in that channel, separated by commas.
_REALNAME	The user's realname, as set in user settings.
_SERVER	The server the window is connected to; this will be the address used to connect to the server
_STATUS	If the window is associated with a channel, this will contain the window's channel status (**`operator`**, **`voiced`**, etc.); otherwise, this will be set to **`normal`**.
_TOPIC	If the window the alias is being used in is a channel window, this will contain the channel's topic, if there is one.
_UPTIME	How long the window the alias is used in has been connected or has been in use, in seconds.
_USERNAME	The user's username, as set in user settings.
_WINDOW	The name of the window the alias is being used in
_WINDOW_TYPE	The type of window the alias is being used in; either `server` for server windows, `channel` for channel windows, or `private` for private chat windows

Built-in aliases can be very useful in scripts, where the script may not "know" what context it is running in:

```
/* This sends a message to the current channel */
/msg $_WINDOW Hello, everybody! My name is $_NICKNAME

/* This sets the current channel's topic */
/topic $_WINDOW We've been around for $_UPTIME seconds!

/* This leaves the current channel */
/part $_WINDOW Goodbye, $_PRESENT
```

Context

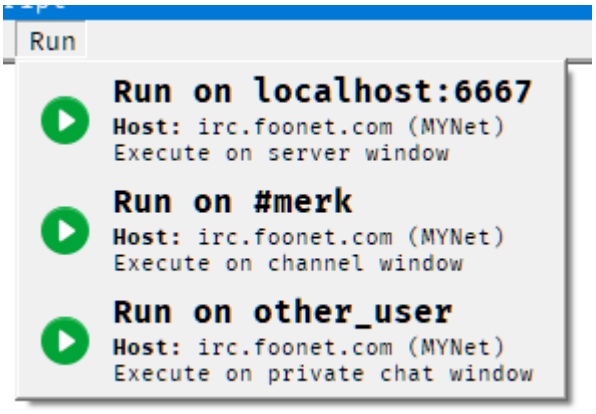
A script or command's *context* is a reference to the window the script or command is being executed in. Context is, for the most part, only necessary for scripts; the context for any commands issued by a window's text input widget is the window the command is being issued in.

Some commands can ignore an argument if they are for the current context; for example, when issuing the **`/part`** command, you can ignore the **`CHANNEL`** argument if the command is intended to be executed in the current window's context:

```
/* This leaves the current channel */
/part

/* This invites a user to the current channel */
/invite my_friend

/* This kicks a user from the current channel */
/kick my_enemy

/* This gives a user operator status in the current channel */
/mode +o my_friend

/* This sets the topic in the current channel */
/topic Welcome to my channel!
```

Context-less commands should *not* be issued by scripts, as it can get confusing if you run the script in the wrong context. However, if the script is being ran in a channel's or private chat window's context, context-less commands are available to the script.

When running a script from a window, either through the server window's toolbar, or the input menu, the script is always ran in that window's context. The script editor "Run" menu allows you to choose which context to run the script in.



*An example "Run" menu from the script editor. The client is connected to a server on **`localhost:6667`**, and is in the channel **`#merk`** while having a private chat with **`other_user`**. Each selection will run the script in the specified context.*

A window's context is "connected" to any other window contexts that share the same network stream. Commands issued from the text input widget in server or chat windows can only effect other windows that share the same server connection. This is called a "shared context".

For example, let's assume that MERK is connected to two servers, **`irc.example.com`** and **`irc.other.net`**. On **`irc.example.com`**, MERK is connected to two channels, **`#merk`** and **`#python`**. On **`irc.other.net`**, MERK is connected to the channel **`#qt`**. In this example, **`#merk`** and **`#python`** have a shared context, while **`#qt`** doesn't have a shared context with the other two window. Commands issued in **`#qt`** will not be able to have an effect on **`#merk`** or **`#python`**, and vice versa.

Scripts do not have this limitation, because they can use the **`/context`** command. The **`/context`** command will search for all windows, no matter what context. The order **`/context`** looks for windows is:

1. Windows that have a shared context with the context the script was executed in.
2. Windows from all contexts.
3. Server windows.

`/context` will move to the *first* window it finds with the name passed to it. If you are in multiple channels with the same name, this may be problematic.

Script Arguments

Arguments can be passed to a script with the **/script** command; just pass them as arguments to the command following the script file name.

Script arguments are tokenized differently than arguments for commands. In arguments for most commands, arguments are considered to consist of a single word, with no spaces. Arguments to scripts can contain spaces, and the number of them is important. To use an argument with spaces in it, contain the argument with quotation marks.

```
/* This calls a script with a single argument */
/script myscript.merk "Hello, world!"

/* Here are multiple arguments, with spaces in each */
/script test.merk "First argument!" "Second argument!" "And third!"
```

To access these arguments, a built-in alias is created for each one, and another is created for all arguments. Each built-in alias is named with the number of the argument: **_1** for the first argument, **_2** for the second, **_3** for the third, and so on. The built-in alias **_0** contains all arguments passed to the script, joined by single spaces.

To make sure your script is called with the right number or arguments, use the **/usage** command. As the first argument to **/usage**, pass the number of arguments your script requires. All arguments after this first will be displayed in the error message if your script it called with an improper number of arguments.

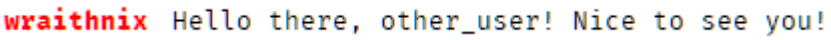
As an example, let's write a script that sends a greeting to someone in the current chat. Our script will require a single argument, a name. When executed with the right number of arguments, it will send the greeting to chat, and if executed with too few arguments, will tell the user how to use the script. Open the script editor, and paste the following code into it, saving the file as **greet.merk**.

```
/*
     Greeting Script
     By Dan Hetrick
     From the MERK User Guide
*/

/*
     This script requires a single argument.
     If none or more than one argument is passed,
     display script usage information.
*/
/usage 1 Usage: /script greet NAME

/msg $_WINDOW Hello there, $_1! Nice to see you!
```

Let's execute our script! In the text input widget, type the following and hit enter:

```
/script greet other_user
```

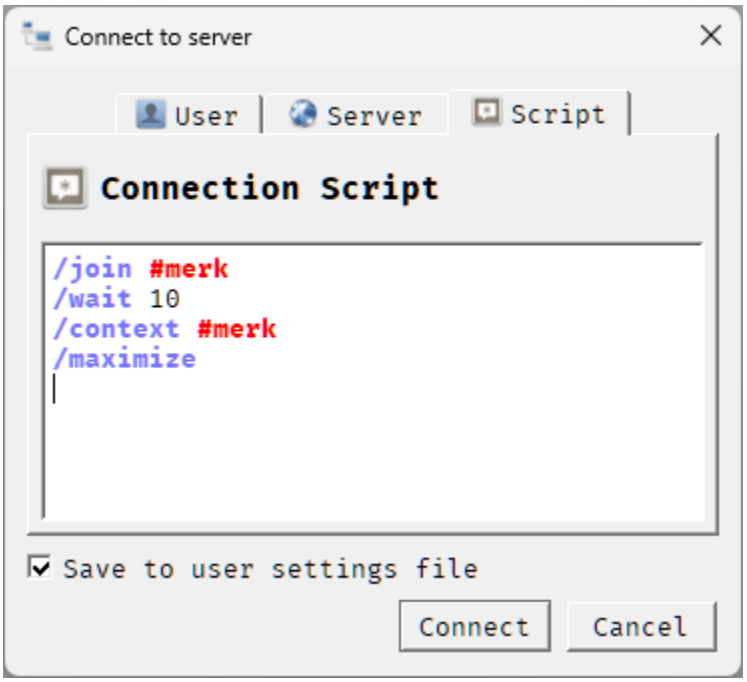
Our greeting is sent to the current chat:



If we executed our script with no arguments, or more than one argument, an error message is displayed:

Connection scripts are *never* called with any arguments.

Writing Connection Scripts



Connection scripts are the scripts that can be entered in the connection dialog, and are executed as soon as the client completes connecting to a server. They are also the only kind of script that *does not have a context*. They can behave, in certain circumstances, as if they have the context of the server's window, but that should not be relied on. To issue commands that will have an effect on another window, use the **`/context`** command to move the script to that window's context.

Before **`/context`**ing to another context, be aware that that window (and the context) may "not exist" yet. The channel join may not have completed, the private chat that you intended to start has not started yet, etc. The **`/wait`** command will help you in these situations, so you can make sure that all the contexts for your script have been created before you issue commands.

For example, let's say that when you connect to your favorite server, automatically join your favorite channel, **`#merk`**, say hello, and maximize the channel window. Your connection script might look like:

```
/alias FAVORITE #merk
/join $FAVORITE
/msg $FAVORITE Hello, everybody!
/wait 10
/context $FAVORITE
/print $_WINDOW Maximizing $FAVORITE!
/maximize $_WINDOW
```

How long to **`/wait`** after connection will take some trial and error, due to many factors: the speed of your Internet connection, the speed of your computer, how busy the server is, how big of a log the client is loading for display, among other things. When in doubt, a longer **`/wait`** is preferable to a shorter one, to make sure that your script executes properly. When first writing a connection script, try **`/wait 30`** to pause the script for 30 seconds, and tweak from there.

Advanced Settings

The last section in the "Settings" dialog shows a number of settings that can be used to fundamentally change how MERK works. ***WARNING:*** Changing these settings may break your installation of MERK, break any existing scripts, or fill up your hard drive. If this occurs, you can reset MERK back to default settings with the following command-line option:

```
python merk.py --reset
```

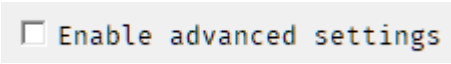
If you are running MERK with the PyInstaller executable, use:

```
merk.exe --reset
```

Changing these settings is not recommended, but may be desired.

Options

To change any of these settings, advanced settings must be enabled by clicking this checkbox:



Unchecking this checkbox will reset all the advanced settings to the value stored in the configuration file, and no settings will be saved. This *must* be enabled when clicking the "Apply" or "Apply & Restart" buttons for any changes to be saved. When changing any any advanced setting, restarting MERK is recommended.

- **Enable aliases**. Unchecking this box will disable aliases in scripts, in user input, and any alias-related commands. This will also disable a number of alias-related settings elsewhere in the "Settings" dialog.
- **Interpolate aliases into input from the text input widget**. If unchecked, aliases will still work in scripts, but cannot be used in the text input widget.
- **Alias interpolation symbol**. Here, you can set the symbol used my MERK to interpolate aliases into input or scripts (by default, **$**). This can be a single or multiple-character symbol.
- **Enable text style editor**. Unchecking this option will disable the ability to edit window text styles. Any edited text styles will still be loaded and used.
- **Show error messages when executing scripts**. Unchecking this option will hide any error messages from scripts. Scripts will still be executed normally.
- **Enable `/shell` command**. Unchecking this option will disable the **`/shell`** command.

- **Show server pings in server windows**. MERK sends "pings" to the IRC server to keep the network connection active. Checking this box will display whenever the client receives a "ping" reply from the server.
- **Save all system messages to log**. Checking this option will save nearly all messages displayed if logging is turned on. This will drastically increase the size of saved logs.
- **Write all network input and output to STDOUT**. This will show all IRC network traffic in STDOUT, which is normally printed to the console MERK is being ran from. This will not display anything if MERK is being executed with the PyInstaller executable (without using additional software to view STDOUT).
- **Write all network input and output to a file in the user's settings directory**. This will write all network input and output to a file or files in the **`settings`** directory. Each IRC connection will have its input and output written to a file named **`SERVER-PORT.txt`**. So, a connection to **`irc.libera.chat`** on port **`6697`** would have its input and output written to **`irc.libera.chat-6697.txt`**. **WARNING!** This will write a *lot* data to your hard drive.