CopyQ Documentation

Lukas Holecek

Apr 20, 2020

The Basics

CopyQ is clipboard manager – a desktop application which stores content of the system clipboard whenever it changes and allows to search the history and copy it back to the system clipboard or paste it directly to other applications.

This documentation describes some basic concepts and workflows as well as more advanced topics like scripting and application development process.

CHAPTER 1

Installation

Packages and installation files are available at Releases page. Alternatively you can install the app with one of the following methods.

On **Windows** you can install Chocolatey package.

On **OS X** you can use Homebrew to install the app.

```
brew cask install copyq
```

On **Ubuntu** set up the official repository and install the app from terminal.

```
sudo apt install software-properties-common python-software-properties
sudo add-apt-repository ppa:hluk/copyq
sudo apt update
sudo apt install copyq
```

CHAPTER 2

Basic Usage

This page describes the basic functionality of CopyQ clipboard manager.

2.1 First Start

To start the CopyQ double-click the program icon or run command `copyq`. This starts graphical interface which can be accessed from tray. Click the tray icon to show application window or right-click the tray icon and select "Show/Hide" or run `copyq show` command.

The central element in the application window is **list with clipboard history**. By default the application **stores any new clipboard content** in the list.

If you copy some text it will immediately show at the top of the list. Try copying text or images from various application to see how this works.

See also:

How to disable storing clipboard?

2.2 Basic Item Manipulation

You can **edit selected text items** in the list by pressing `F2`. After editing **save the text** with `F2`.

Create **new item** with `Ctrl+N`, type some text and press `F2`.

Copy the selected items back to clipboard with Enter or `Ctrl+C`.

Move items around with `Ctrl+Down` and `Ctrl+Up`.

You can move important or special items to new tabs (see *Tabs* for more info).

2.3 Search

In the list you can simply **search for text by typing some text**.

For example typing "Example" will hide items that don't contain "Example" text. Press Enter to copy the first found item.

2.4 Tray

To quickly copy item to clipboard you can select the item from tray menu. To display the menu either right-click on tray icon, run command `copyq menu` or use a custom system shortcut.

After selecting an item in tray menu and pressing enter (pressing a number key works as well) the item is copied to the clipboard.

See also:

How to open application window or tray menu using shortcut?

How to paste double-clicked item from application window?

CHAPTER 3

Tabs and Items

3.1 Tabs

Tabs are means to organize texts, images and other data.

Initially there is only one tab which is used for storing clipboard and the tab bar is hidden.

User can create new tabs from "Tabs" menu or using `Ctrl+T`. The tab bar will appear if there is more than one tab. Using mouse, user can reorder tabs and drop items and other data into tabs.

If tab name contains `&`, the following letter is used for quick access to the tab (the letter is underlined in tab bar or tab tree and `&` is hidden). For example, tab named "&Clipboard" can be opened using `Alt+C` shortcut.

Option "Tab Tree" enables user to organize tabs into groups. Tabs with names "Job/Tasks/1" and "Job/Tasks/2" will create following structure in tab tree.

```
> Job
    > Tasks
        > 1
        > 2
```

3.2 Storing Clipboard

If "Store Clipboard" option is enabled (under "General" tab in config dialog) and "Tab for storing clipboard" is set (under "History" tab in config dialog), every time user copies something to clipboard a new item will be created in that particular tab. The item will contain only text and data that are needed by plugins (e.g. plugin "Images" requires `image/svg`, `image/png` or similar).

Any additional data to store can be specified using configuration for "Data" plugin (under "Items" tab in config dialog).

3.3 Organizing Items

Any data or item can be moved or copied to other tab by dragging it using mouse or by pasting it in item list.

Commands can automatically organize items into tabs. For example, following command will put copied images to "Images" tab (to use the command, copy it to the command list in configuration).

```
[Command]
Name=Move Images to Other Tab
Input=image/png
Automatic=true
Remove=true
Icon=\xf03e
Tab=&Images
```

CHAPTER 4

Keyboard

This page lists useful default shortcuts and key mappings for CopyQ and describes how to change them.

CopyQ is keyboard-friendly, i.e. it should be possible to quickly access any functionality with keyboard without using mouse.

4.1 Default Shortcuts

Note: On OS X, use key instead of Ctrl for the shortcuts.

- PgDown/PgUp, Home/End, Up/Down - item list navigation
- Left, Right, Ctrl+Tab, Ctrl+Shift+Tab - tab navigation
- Ctrl+T, Ctrl+W - create and remove tabs
- Ctrl+Up, Ctrl+Down - move selected items
- Ctrl+Left, Ctrl+Right - cycle through item formats
- Esc - cancel search, hide window
- Ctrl+Q - exit
- F2 - edit selected items
- Ctrl+E - edit items in an external editor
- F5 - open action dialog for selected items
- Delete - delete selected items
- Ctrl+A - select all
- Enter - put current item into clipboard and paste item (optional)
- Ctrl+1…Ctrl+9 - focus a tab in given order

- Ctrl+0 - focus last tab

4.2 Search

Start typing a text to search items. This works in main application window and `copyq menu`.

4.3 Change Shortcuts

To change the shortcuts:

- open “File - Preferences”,
- select “Shortcuts” tab,
- click the button next to action you need to change,
- press a shortcut on keyboard,
- click OK to save the dialog.

4.4 Create new Shortcut

If and action with shortcut is missing in the Shortcuts configuration tab, you can use predefined ones:

- open “File - Commands/Global Shortcuts...”,
- click “Add” button,
- select command (e.g. “Show/hide main window”),
- press a shortcut on keyboard,
- click OK to save the dialog.

CHAPTER 5

Images

This page describes how to display and work with images in CopyQ.

5.1 Display Image Items

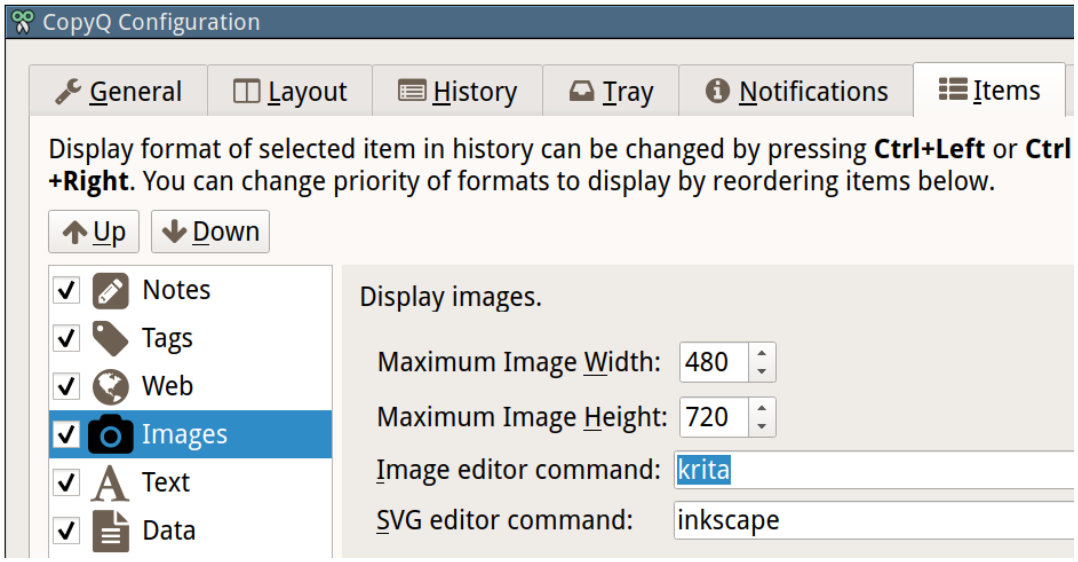
Displaying images can be configured in “Items” configuration tab.

On Windows, “Item Image” plugin needs to be installed.

To disable storing and diplaying image, disable the Image plugin (uncheck the checkbox next to “Image” in configuration).

5.2 Editor

Editors for bitmap and SVG images can be set in the configuration.



Editing an image item (default shortcut is Ctrl+E) should open the image editor.

Unfortunately, sometimes an item looks like an image but is an HTML. You can list available formats in Content dialog F4.

5.3 Preview Image

It's useful to limit size of image item to a maximum width and height in the configuration.

You can still display the whole image in Preview dock (F7) or using Content dialog (F4).

5.4 Take Screenshots

You can use built-in functionality for taking screenshots of whole or part of the desktop.

Paste taken screenshots to CopyQ to store them for later use.

5.5 Save Image to a File

To save an image to a file, either copy it or drag'n'drop it to a file manager (if supported) or save it using command line.

```
copyq read image/png 0 > image.png
```

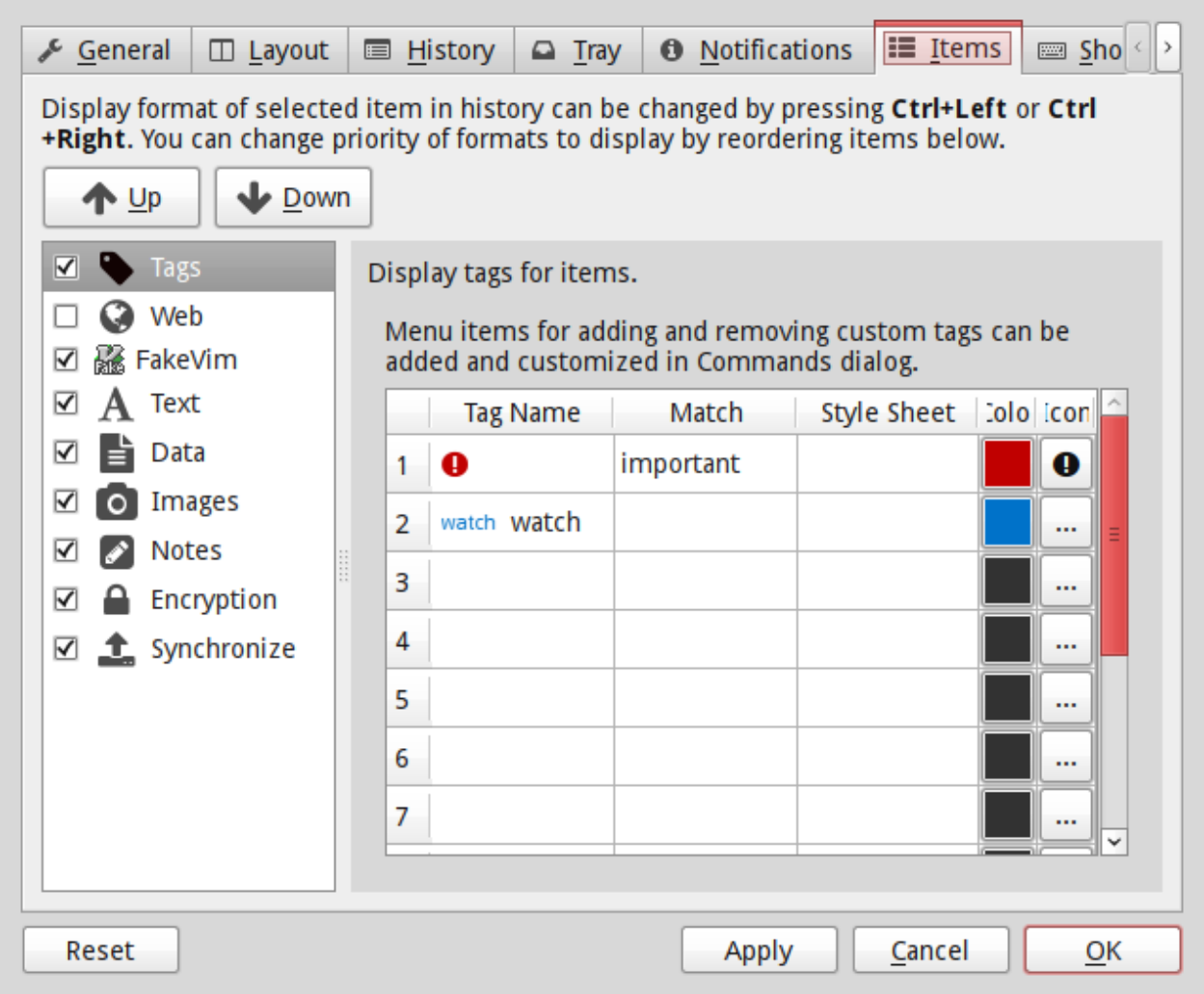
Alternatively use "Save Item/Clipboard To a File" command.

CHAPTER 6

Tags

Tags are small icons or short texts in upper right corner of an item used to mark important or special items.

Tags can be configured in “Items” configuration tab. On Windows, “Item Tags” plugin needs to be installed.



Configuration consists solely of table where each row contains matching and styling rules for tags.

Style from the first row which matches tag text is applied on the tag.

Column in the table are following.

- "Tag Name"

Text for the tag. This is used for matching if "Match" column is empty. Expressions like `\1`, `\2` etc. will be replaced with captured texts from "Match" column.

- "Match"

Regular expression for matching the tags.

E.g. `.*` (any tag), `Important:  .*` (match prefix), `\d\d\d\d-\d\d-\d\d.*` (date time).

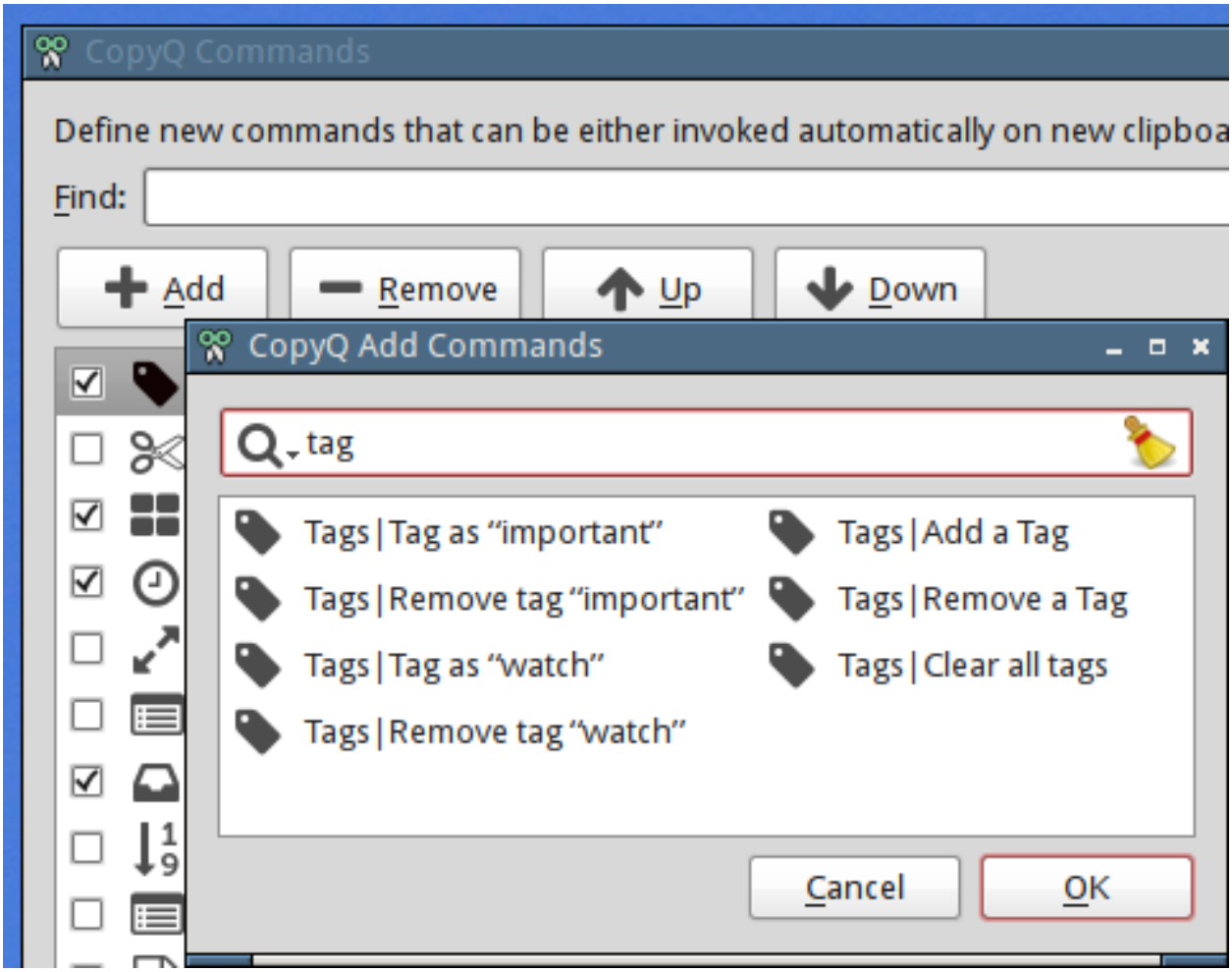
- "Style Sheet"

Simple style sheet (http://doc.qt.io/qt-4.8/stylesheet-reference.html).

E.g. `border:  1px solid white; border-radius:  3px; font-size:  7pt`.

- "Color" - Text color.
- "Icon" - Icon for tag. To show only icon without text you have to set "Match" and keep "Tag Name" field empty.

Tagging items can be accessed from context menu if appropriate commands are added in Command dialog (generated commands are available in the list under "Add" button).



Alternatively, tags are added to an item by setting "application/x-copyq-tags" format. It can contain multiple tags separated by comma. The tag text itself can be written as simple HTML.

Example:

```
copyq write text/plain "Item with tag" application/x-copyq-tags "Some tag text"
```

CHAPTER 7

FAQ - Frequently Asked Questions

7.1 How to open application window or tray menu using shortcut?

Add new command to open window or menu with global shortcut:

1. open "Command" dialog (`F6` shortcut),
2. click "Add" button in the dialog,
3. select "Show/hide main window" or "Show the tray menu" from the list and click "OK" button,
4. click the button next to "Global Shortcut" label and set the shortcut,
5. click "OK" button to save the changes.

For more information about commands see *Writing Commands and Adding Functionality*.

7.2 How to paste double-clicked item from application window?

1. Open "Preferences" (`Ctrl+P` shortcut),
2. go to "History" tab,
3. enable "Paste to current window" option.

Next time you open main window and activate an item it should be pasted.

7.3 How to paste as plain text?

To **paste clipboard as plain text**:

1. open "Command" dialog (`F6` shortcut),
2. click "Add" button in the dialog,

3. select “Paste clipboard as plain text” from the list and click “OK” button,
4. click the button next to “Global Shortcut” label and set the shortcut,
5. click “OK” button to save the changes.

To **paste selected items as plain text** (from application window) follow the steps above but add “Paste as Plain Text” command instead and change “Shortcut”.

You can also disallow rich text storing: go to preferences, “Items” tab and uncheck “Web” checkbox under “Text” uncheck “HTML” checkbox.

7.4 How to disable storing clipboard?

To temporarily disable storing clipboard in item list, select menu item “File - Disable Clipboard Storing” (`Ctrl+Shift+X` shortcut). To re-enable the functionality select “File - Enable Clipboard Storing” (same shortcut).

To permanently disable storing clipboard:

1. Open “Preferences” (`Ctrl+P` shortcut),
2. go to “History” tab,
3. clear “Tab for storing clipboard” field.

7.5 How to back up tabs, configuration and commands?

From menu select “File - Export” and choose what tabs to export and whether to export configuration and commands.

To restore the backup select menu item “File - Import”, select the exported file and choose what to import back.

Note: Importing tabs and commands won’t override existing tabs but create new ones

---

7.6 How to enable or disable displying notification when clipboard changes?

To enable displaying the notifications:

1. open “Preferences” (`Ctrl+P` shortcut),
2. go to “Notifications” tab,
3. set non-zero value for “Interval in seconds to display notifications”,
4. set non-zero value for “Number of lines for clipboard notification”,
5. click “OK” button.

To disable displaying the notifications, set either of the options mentioned above to zero.

7.7 How to load shared commands and share them?

You can stumble upon code that looks like this.

```
[Command]
Name=Show/hide main window
Command=copyq: toggle()
Icon=\xf022
GlobalShortcut=ctrl+shift+1
```

This code represents a command that can used in CopyQ (specifically it opens main window on Ctrl+Shift+1). To use the command in CopyQ:

1. copy the code above,
2. open "Command" dialog (`F6` shortcut),
3. click "Paste Commands" button at the bottom of the dialog,
4. click OK button.

(Now you should be able to open main window with Ctrl+Shift+1.)

To share your commands, you can select the commands from command list in "Command" dialog and press "Copy Selected" button (or just hit Ctrl+C).

7.8 How to omit storing text copied from specific windows like a password manager?

Add and modify automatic command to ignore text copied from the window:

1. open "Command" dialog (`F6` shortcut),
2. click "Add" button in the dialog,
3. select "Ignore *Password* window" from the list and click "OK" button,
4. select "Show Advanced"
5. change "Window" text box to match the title (or part of it) of the window to ignore (e.g. `KeePass`),
6. click "OK" button to save the changes.

Note: This new command should be at top of the command list because automatic commands are executed in order they appear in the list and we don't want to process sensitive data in any way.

7.9 How to enable logging

Set environment variable `COPYQ_LOG_LEVEL` to `DEBUG` for verbose logging and set `COPYQ_LOG_FILE` to a file path for the log.

You can copy current log file path to clipboard from Action dialog (F5 shortcut) by entering command `copyq 'copy(info("log"))'`.

7.10 How to preserve the order of copied items on copy or pasting multiple items?

a. Reverse order of selected items with `Ctrl+Shift+R` and copy them or

b. select items in reverse order and copy.

See #165.

7.11 How does pasting single/multiple items internally work?

`Return` key copies the whole item (with all formats) to the clipboard and – if the "Paste to current window" option is enabled – it sends `Shift+Insert` to previous window. So the target application decides what format to paste on `Shift+Insert`.

If you select more items and press `Return`, just the concatenated text of selected items is put into clipboard. Thought it could do more in future, like join HTML, images or other formats.

See #165.

7.12 How to open the menu or context menu with only the keyboard?

Use `Alt+I` to open the item menu or use the `Menu` key on your keyboard to open the context menu for selected items.

7.13 Is it possible to hide menu bar to have even cleaner main window?

Menu bar can be hidden by modifying style sheet of current theme.

1. Open "Preferences" (`Ctrl+P` shortcut),
2. go to "Appearance" tab,
3. enable checkbox "Set colors for tabs, tool bar and menus",
4. click "Edit Theme" button,
5. find `menu_bar_css` option and add `height: 0`:

```
menu_bar_css="
    ;height: 0
    ;background: ${bg}
    ;color: ${fg}"
```

7.14 How to reuse file paths copied from a file manager?

By default only the text is stored in item list when you copy of cut files from a file manager. Other data are usually needed to be able to copy/paste files from CopyQ.

You have to add new data formats (MIME) to format list in "Data" item under "Item" configuration tab. Commonly used format in many file managers is `text/uri-list`. Other special formats include `x-special/gnome-copied-files` for Nautilus, `application/x-kde-cutselection` for Dolphin. These formats are used to specify type of action (copy or cut).

7.15 Where to find saved items and configuration?

You can find configuration and saved items in:

- Windows folder `%APPDATA%\copyq` for installed version of the app or `config` folder in unzipped portable version,
- Linux directory `~/.config/copyq`.

Run `copyq info config` to get absolute path to the configuration file (parent directory contains saved items).

Note: Main configuration for installed version of the app on Windows is stored in registry.

7.16 Why are items and configuration not saved?

Check access rights to configuration directory and files.

CHAPTER 8

Command Line

Tabs, items, clipboard and configuration can be changed through command line interface. Run command `copyq help` to see complete list of commands and their description.

To add new item to tab with name "notes" run:

```
copyq tab notes add "This is the first note."
```

To print the item:

```
copyq tab notes read 0
```

Add other item:

```
copyq tab notes add "This is second note."
```

and print all items in the tab:

```
copyq eval -- "tab('notes'); for(i=size(); i>0; --i) print(str(read(i-1)) + '\n');"
```

This will print:

```
This is the first note.
This is second note.
```

Among other things that are possible with CopyQ are:

- open video player if text copied in clipboard is URL with multimedia,
- store text copied from a code editor in "code" tab,
- store URLs in different tab,
- save screenshots (print-screen),
- load all files from directory to items (create image gallery),
- replace a text in all matching items,

- run item as a Python script.

CHAPTER 9

Sessions

You can run multiple instances of the application given that they have different session names.

To start new instance with `test1` name, run:

```
copyq --session=test1
```

This instance uses configuration, tabs and items unique to given session name.

You can still start default session (with empty session name) with just:

```
copyq
```

In the same manner you can manipulate the session. E.g. to add an item to first tab in `test1` session, run:

```
copyq --session=test1 add "Some text"
```

Default session has empty name but it can be overridden by setting `COPYQ_SESSION_NAME` environment variable.

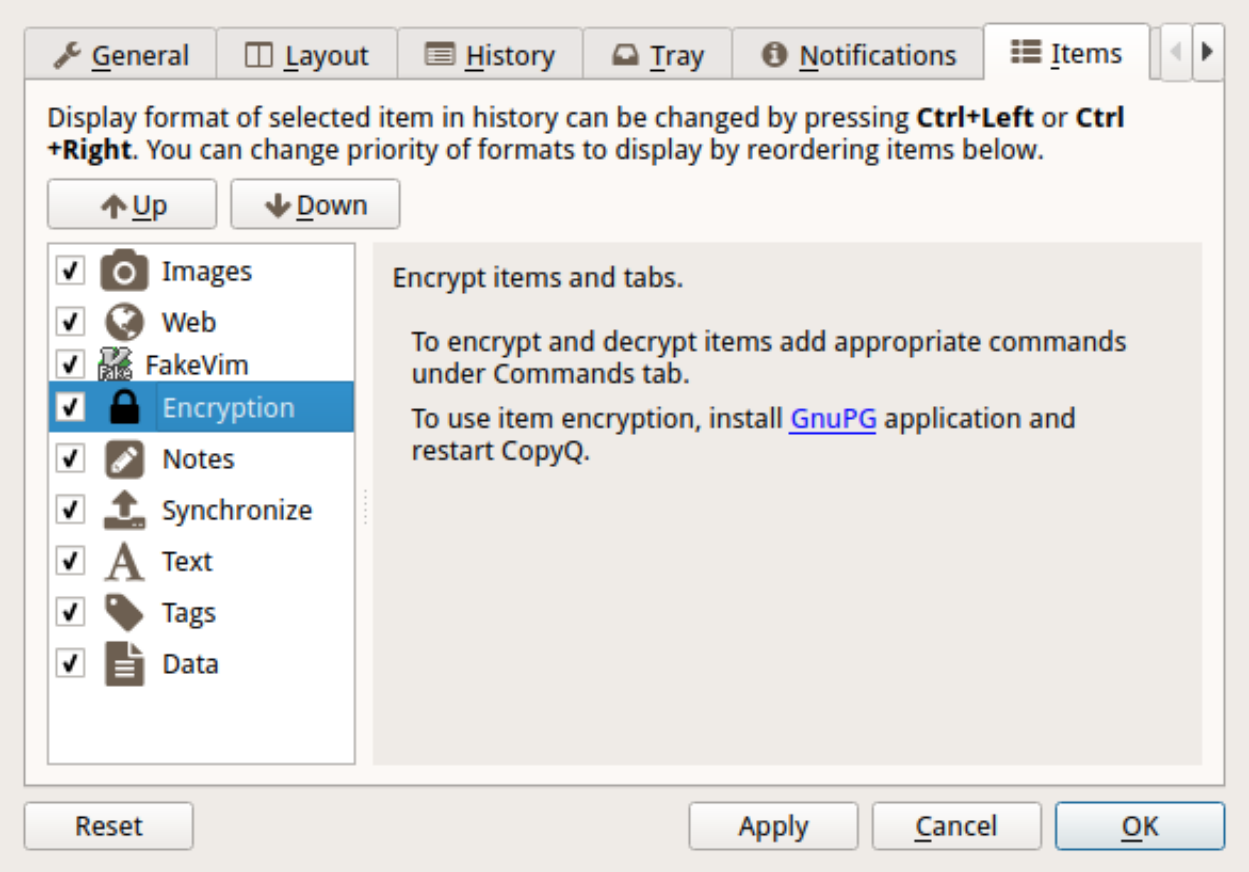
CHAPTER 10

Password Protection

This page describes how to encrypt and protect selected tabs and single items with a password.

10.1 Installation

To enable this feature you need to have “Encryption” item plugin.



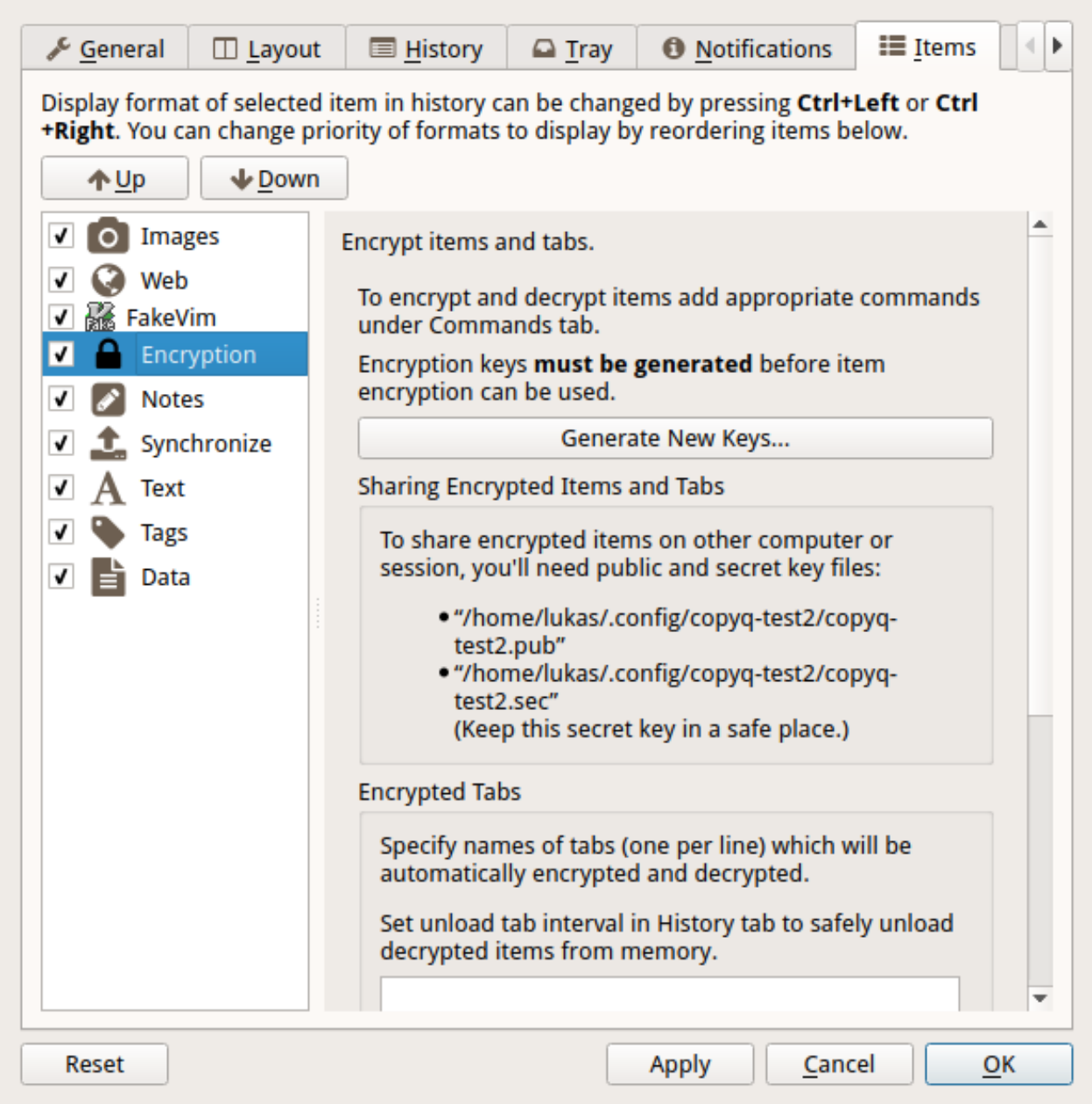
The plugin configuration (under "Items" configuration tab in Configruation dialog) may prompt you to install GnuPG:

- For Windows you can install Gpg4win.
- For Linux install `gpg` command line utility. It's usually provided by `gnupg` package but the package name may differ on some distributions.

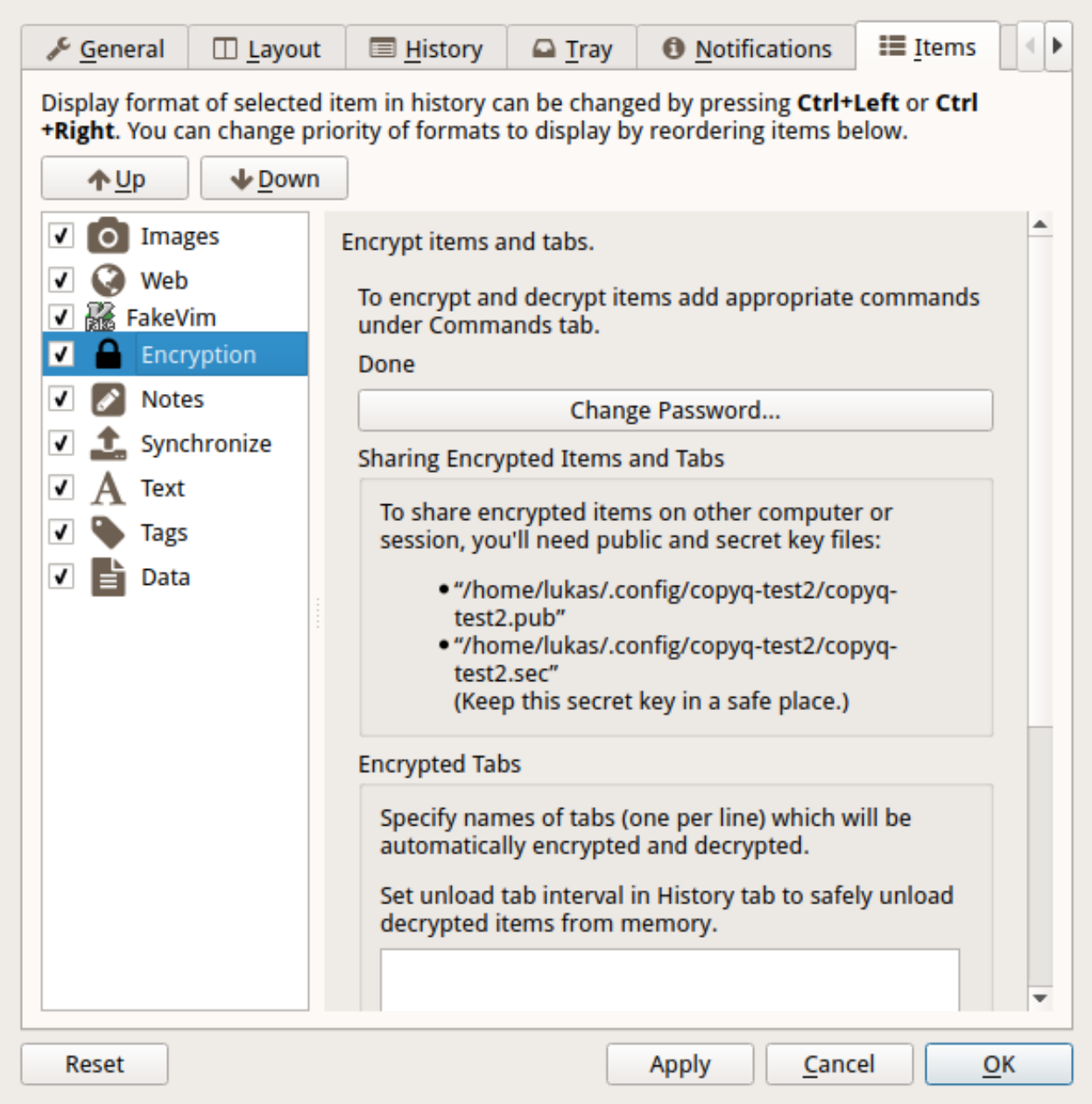
10.2 Generate Keys and Set Password

To be able to encrypt tabs and items you first need to generate private and public key files.

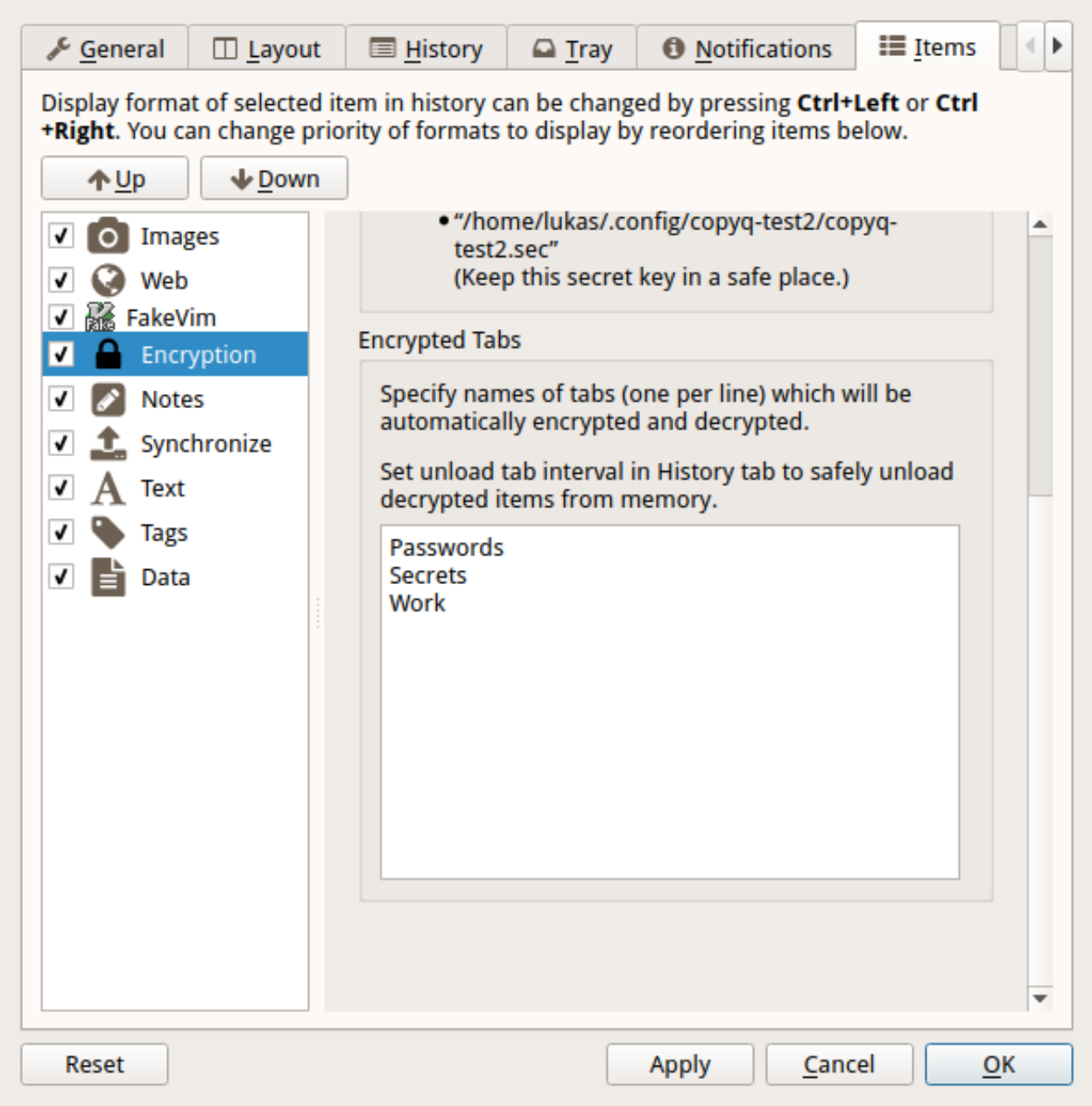
Click on the "Generate Ney Keys…" button and wait.



If didn't set a password in previous step click "Change Password. . . " button and set it.



Last step in configuration is to set tabs to encrypt. You can skip this step if you only need to encrypt single item in each tab (see next section).

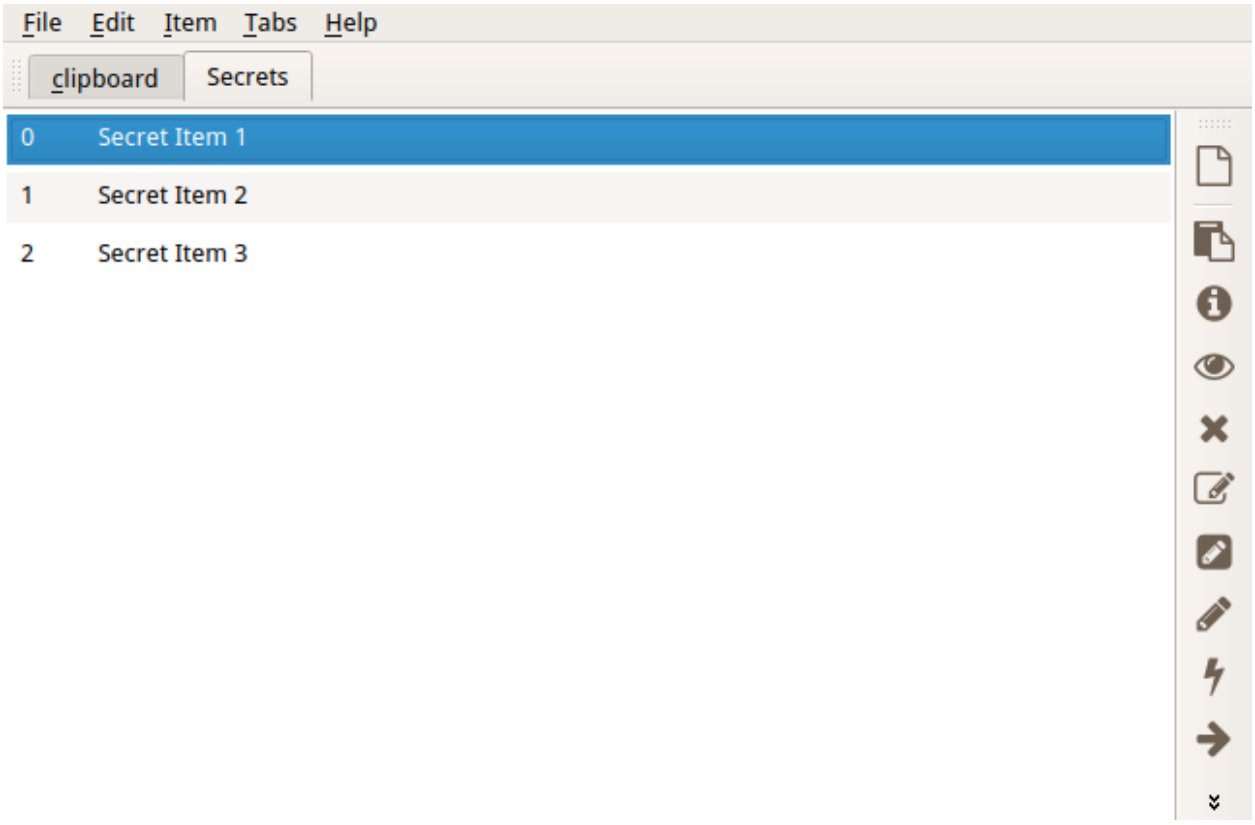


Click "OK" button to confirm Configuration dialog.

10.3 Protect Tabs

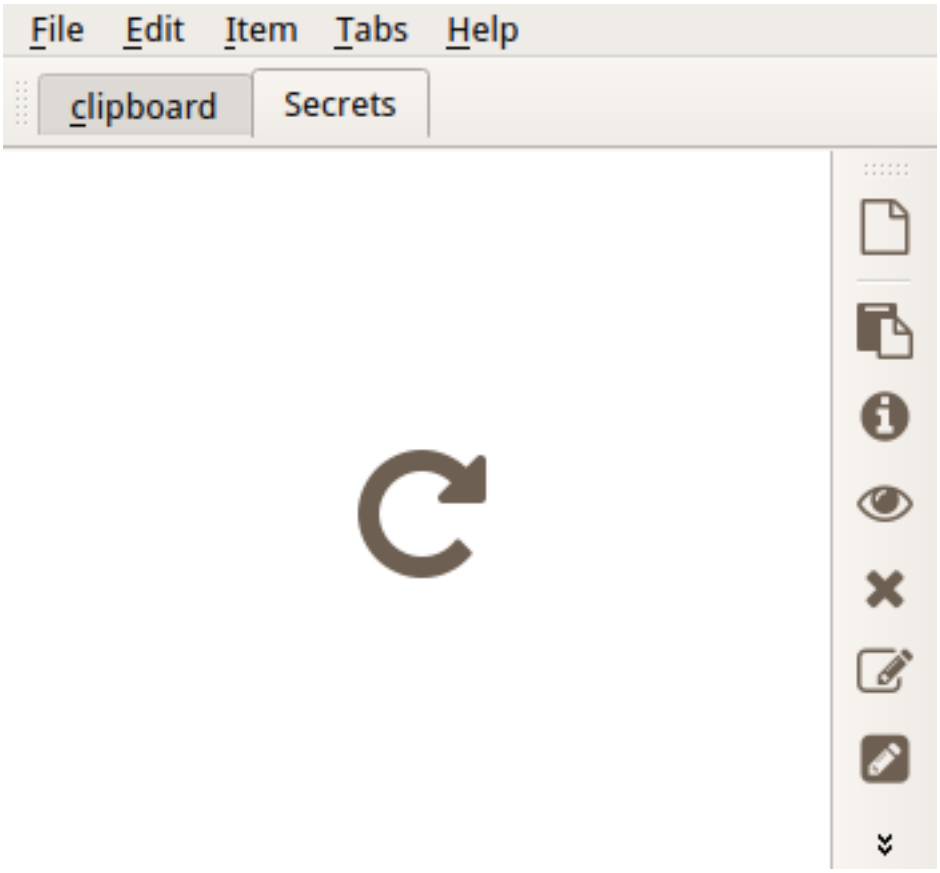
Now you can create the tabs you want to encrypt (Ctrl+T to create new tab).

The tab name should be same as one of the tabs entered in plugin configuration in previous step.



You'll be prompt to enter password in future (you only need to enter it once in a while).

If you enter wrong password or cancel the password prompt you can later click on "Reload" button in tab to enter password again.



10.4 Protect Single Items

To protect items in unprotected tab you can add menu and tool bar actions with keyboard shortcut.

Go to Command dialog F6, click on “Add” button, “Encryption” commands from list and confirm dialogs with “OK” button.

Now you can select items and press Ctrl+L to encrypt (“Items - Encryption - Encrypt” in menu).

To decrypt selected item press Ctrl+L (“Items - Encryption - Decrypt” in menu).

CHAPTER 11

Writing Commands and Adding Functionality

CopyQ allows you to extend its functionality through commands in following ways.

1. Add custom commands to context menu for selected items in history.
2. Run custom commands automatically when clipboard changes.
3. Assign global/system-wide shortcuts to custom commands.

Here are some examples what can be achieved by using commands.

- Automatically store web links or other types of clipboard content in special tabs to keep the history clean.
- Paste current date and time or modified clipboard on a global shortcut.
- Pass selected items or clipboard to external application (e.g. web browser or image editor).
- Keep TODO lists and tag items as "important" or use custom tags.
- See *Command Examples* for some other ideas and useful commands.

11.1 Command Dialog

You can create new commands in Command dialog. To open the dialog either:

- press default shortcut F6 or
- select menu item "Commands/Global Shortcuts…" in "File" menu.

Command dialog contains:

- list of custom commands on the left,
- settings for currently selected command on the right,
- command filter text field at the top,
- buttons to modify the command list (add, remove and move commands) at the top,
- buttons to save, load, copy and paste commands at the bottom.

11.1.1 Create New Command

To create new command click the "Add" button in Command dialog. This opens list with predefined commands.

"New Command" creates new empty command (but it won't do anything without being configured). One of the most frequently used predefined command is "Show/hide main window" which allows you to assign global shortcut for showing and hiding CopyQ window.

If you double click a predefined command (or select one or multiple commands and click OK) it will be added to list of commands. The right part of the Command dialog now shows the configuration for the new command.

For example, for the "Show/hide main window" you'll most likely need to change only the "Global Shortcut" option so click on the button next to it and press the shortcut you want to assign.

Commands can be quickly disabled by clicking the check box next to them in command list.

By clicking on "OK" or "Apply" button in the dialog all commands will be saved permanently.

Command Options

The following options can be set for commands.

If unsure what an option does, hover mouse pointer over it and tool tip with description will appear.

Name

Name of the command. This is used in context menu if "In Menu" check box is enabled. Use `/` in the name to create sub-menus.

Group: Type of Action

This group sets the main type of the command. Usually only one sub-option is set.

a. Automatic

If enabled, the command is triggered whenever clipboard changes.

Automatic items are run in order they appear in the command list. No other automatic commands will be run if a triggered automatic command has "Remove Item" option set or calls `copyq ignore`.

The command is **applied on current clipboard data** - i.e. options below access text or other data in clipboard.

b. In Menu

If enabled, the command can be run from main window either with application shortcut, from context menu or "Item" menu. The command can be also run from tray menu.

Shortcuts can be assigned by clicking on the button next to the option. These shortcuts work only when CopyQ window has focus.

If the command is run from **tray menu**, it is **applied on clipboard data**, otherwise it's **applied on data in selected items**.

c. Global Shortcut

If enabled, the command is triggered whenever assigned shortcut is pressed. These shortcuts work even when CopyQ window doesn't have focus.

This command is **not applied on data** in clipboard nor selected items.

Group: Match Items

This group is visible only for "Automatic" or "In Menu" commands. Sub-options specify when the command can be used.

1. Content

Regular expression to match text of selected items (for "In Menu" command) or clipboard (for "Automatic" command).

For example, `^https?://` will match simple web addresses (text starting with `http://` or `https://`).

2. Window

Regular expression to match window title of active window (only for "Automatic" command).

For example, `– Chromium$` or `Mozilla Firefox$` to match some web browser window titles (`$` in the expression means end of the title).

3. Filter

A command for validating text of selected items (for "In Menu" command) or clipboard (for "Automatic" command).

If the command exits with non-zero exit code it won't be shown in context menu and automatically triggered on clipboard change.

Example, `copyq: if (tab().indexOf("Web") == -1) fail()` triggers the command only if tab "Web" is available.

4. Format

Match format of selected items or clipboard.

The data of this format will be sent to **standard input** of the command - this doesn't apply if the command is triggered with global shortcut.

Command

The command to run.

This can contain either:

- simple command line (e.g. `copyq popup %1` - expression `%1` means text of the selected item or clipboard),
- input for command interpreter (prefixed with `bash:`, `powershell:`, `python:` etc.) or
- CopyQ script (prefixed with `copyq:`).

You can use `COPYQ` environment variable to get path of application binary.

Current CopyQ session name is stored in `COPYQ_SESSION_NAME` environment variable (see *Sessions*).

Example (call CopyQ from Python):

```
python:
import os
from subprocess import call
copyq = os.environ['COPYQ']
call([copyq, 'read', '0'])
```

Example (call CopyQ from PowerShell on Windows):

```
powershell:
$Item1 = (& "$env:COPYQ" read 0 | Out-String)
echo "First item: $Item1"
```

Group: Action

This group is visible only for "Automatic" or "In Menu" commands.

1. Copy to tab

Creates new item in given tab.

2. Remove Item

Removes selected items. If enabled for "Automatic" command, the clipboard will be ignored and no other automatic commands will be executed.

Group: Menu Action

This group is visible only for "In Menu" commands.

1. Hide main window after activation

If enabled, main window will be hidden after the command is executed.

Group: Command options

This group is visible only for "Automatic" or "In Menu" commands.

1. Wait

Show action dialog before applying options below.

2. Transform

Modify selected items - i.e. remove them and replace with **standard output** of the command.

3. Output

Format of **standard output** to save as new item.

4. Separator

Separator for splitting output to multiple items (`\n` to split lines).

5. Output tab

Tab for saving the output of command.

11.1.2 Save and Share Commands

You can back up or share commands by saving them in a file ("Save Selected Commands…" button) or by copying them to clipboard.

The saved commands can be loaded back to command list ("Load Commands…" button) or pasted to the list from clipboard.

You can try some examples by copying commands from *Command Examples*.

CHAPTER 12

Scripting

If you need to process items in some non-trivial way you can take advantage of the scripting interface the application provides. This is accessible on command line as `copyq eval SCRIPT` or `copyq -e SCRIPT` where `SCRIPT` is string containing commands written in Javascript-similar scripting language (Qt Script with is ECMAScript scripting language).

Every command line option is available as function in the scripting interface. Command `copyq help tab` can be written as `copyq eval 'print(help("tab"))'` (note: `print` is needed to print the return value of `help("tab")` function call).

12.1 Searching Items

You can print each item with `copyq read N` where N is item number from 0 to `copyq size` (i.e. number of items in the first tab) and put item to clipboard with `copyq select N`. With these commands it's possible to search items and copy the right one with a script. E.g. having file `script.js` containing

```
var match = "MATCH-THIS";
var i = 0;
while (i < size() && str(read(i)).indexOf(match) === -1)
    ++i;
select(i);
```

and passing it to CopyQ using `cat script.js | copyq eval -` will put first item containing "MATCH-THIS" string to clipboard.

12.2 Working with Tabs

By default commands and functions work with items in the first tab. Calling `read(0, 1, 2)` will read first three items from the first tab. To access items in other tab you need to switch the current tab with `tab("TAB_NAME")` (or `copyq tab TAB_NAME` on command line) where `TAB_NAME` is name of the tab.

For example to search for an item as in the previous script but in all tabs you'll have to run:

```
var match = "MATCH-THIS";
var tabs = tab();
for (var i in tabs) {
    tab(tabs[i]);
    var j = 0;
    while (j < size() && str(read(j)).indexOf(match) === -1)
        ++j;
    if (j < size())
        print("Match in tab \"" + tabs[i] + "\" item number " + j + ".\n");
}
```

12.3 Scripting Functions

As mentioned above, all command line options are also available for scripting e.g.: `show()`, `hide()`, `toggle()`, `copy()`, `paste()`.

Reference for available scripting functions can be found at *Scripting API*.

Other supported functions can be found at ECMAScript Reference.

CHAPTER 13

Command Examples

Here are some useful commands for creating custom menu items, global shortcuts and automatically process new clipboard content in CopyQ.

If you want to use any of the commands below, copy it to clipboard and paste it to the command list in Command dialog (opened with F6 shortcut). For detailed info see *How to load shared commands and share them?*.

All these and more commands are available at CopyQ command repository.

13.1 Join Selected Items

Creates new item containing concatenated text of selected items.

```
[Command]
Name=Join Selected Items
Command=copyq add %1
InMenu=true
Icon=\xf066
Shortcut=Space
```

13.2 Paste Current Date and Time

Copies current date/time text to clipboard and pastes to current window on global shortcut Win+Alt+T.

```
[Command]
Command="
    copyq:
    var time = dateString('yyyy-MM-dd hh:mm:ss')
    copy('Current date/time is ' + time)
    paste()"
GlobalShortcut=meta+alt+t
```

(continues on next page)

(continued from previous page)

```
Icon=\xf017
Name=Paste Current Time
```

13.3 Play Sound when Copying to Clipboard

Following command will play an audio file whenever something is copied clipboard.

On Windows:

```
[Command]
Name=Play Sound on Copy
Command="
     powershell:
     (New-Object Media.SoundPlayer \"C:\\Users\\copy.wav\").PlaySync()"
Automatic=true
Icon=\xf028
```

On Linux (requires VLC multimedia player):

```
[Command]
Name=Play Sound on Copy
Command="
     bash:
     cvlc --play-and-exit ~/audio/example.mp3"
Automatic=true
Icon=\xf028
```

13.4 Edit and Paste

Following command allows to edit current clipboard text before pasting it. If the editing is canceled the text won't be pasted.

```
[Command]
Command="
    copyq:
    var text = dialog('paste', str(clipboard()))
    if (text) {
      copy(text)
      copySelection(text)
      paste()
    }"
GlobalShortcut=ctrl+shift+v
Icon=\xf0ea
Name=Edit and Paste
```

13.5 Ignore Images when Text is Available

This is useful for ignoring cells copied as images from Microsoft Excel and LibreOffice Calc.

```
[Command]
Automatic=true
Command="
    copyq:
    var text = data('text/plain')
    copy(text)
    add(text)"
Icon=\xf031
Input=image/bmp
MatchCommand="copyq: if ( str(data('text/plain')) == '' ) fail()"
Name=Ignore Images when Text Copied
Remove=true
```

13.6 Remove Background and Text Colors

Removes background and text colors from rich text (e.g. text copied from web pages).

Command can be both automatically applied on text copied to clipboard and invoked from menu (or using custom shortcut).

```
[Command]
Automatic=true
Command="
    copyq:
    var html = str(input())
    html = html.replace(/color\\s*:/g, 'xxx:')
    setData('text/html', html)"
Icon=\xf042
InMenu=true
Input=text/html
Name=Remove Background and Text Colors
```

13.7 Linkify

Creates interactive link from plain text.

```
[Command]
Name=Linkify
Match=^(https?|ftps?|file|mailto)://
Command="
    copyq:
    var link = str(input());
    var href = '<a href=\"###\">###</a>';
    write(
      'text/plain', link,
      'text/html', href.replace(/###/g, link)
    );"
Input=text/plain
Automatic=true
Remove=true
Icon=\xf127
```

13.8 Highlight Text

Highlight all occurrences of a text (change `x = "text"` to match something else than `text`).

```
[Command]
Name=Highlight Text
Command="
    copyq:
    x = 'text'
    style = 'background: yellow; text-decoration: underline'

    text = str(input())
    x = x.toLowerCase()
    lowertext = text.toLowerCase()
    html = ''
    a = 0
    esc = function(a, b) {
        return escapeHTML( text.substr(a, b - a) )
    }

    while (1) {
        b = lowertext.indexOf(x, a)
        if (b != -1) {
            html += esc(a, b) + '<span>' + esc(b, b + x.length) + '</span>'
        } else {
            html += esc(a, text.length)
            break
        }
        a = b + x.length;
    }

    tab( selectedtab() )
    write(
        index(),
        'text/plain', text,
        'text/html',
            '<html><head><style>span{'
            + style +
            '}</style></head><body>'
            + html +
            '</body></html>'
    )"
Input=text/plain
Wait=true
InMenu=true
```

13.9 Render HTML

Render HTML code.

```
[Command]
Name=Render HTML
Match=^\\s*<(!|html)
Command="
    copyq:
```

(continues on next page)

(continued from previous page)

```
    tab(selectedtab())
    write(index() + 1, 'text/html', input())"
Input=text/plain
InMenu=true
```

13.10 Translate to English

Pass to text to Google Translate.

```
[Command]
Name=Translate to English
Command="
    copyq:
    text = str(input())
    url = \"https://translate.google.com/#auto/en/???\"

    x = url.replace(\"???\", encodeURIComponent(text))
    html = '<html><head><meta http-equiv=\"refresh\" content=\"0;url=' + x + '\" /></
↪head></html>'

    tab(selectedtab())
    write(index() + 1, \"text/html\", html)"
Input=text/plain
InMenu=true
```

13.11 Paste and Forget

Paste selected items and clear clipboard.

```
[Command]
Name=Paste and Forget
Command="
    copyq:
    tab(selectedtab())
    items = selecteditems()
    if (items.length > 1) {
        text = ''
        for (i in items)
            text += read(items[i]);
        copy(text)
    } else {
        select(items[0])
    }

    hide()
    paste()
    copy('')"
InMenu=true
Icon=\xf0ea
Shortcut=Ctrl+Return
```

13.12 Render Math Equations

Render math equations using MathJax (e.g. `$$x = {-b \pm \sqrt{b^2-4ac} \over 2a}$$`).

```
[Command]
Name=Render Math Equations
Command="
    copyq:
    text = str(input())
    js = 'http://cdn.mathjax.org/mathjax/latest/MathJax.js?config=TeX-AMS-MML_
↪HTMLorMML'

    html = '<html><head><script type=\"text/javascript\" src=\"' + js + '\"></script>
↪</head><body>' + escapeHTML(text) + '</body></html>';

    tab(selectedtab())
    write(index() + 1, 'text/html', html)"
Input=text/plain
InMenu=true
Icon=\xf12b
```

13.13 Move Images to Other Tab

With this command active, images won't be saved in the first tab. This can make application a bit more snappier since big image data won't need to be loaded when main window is displayed or clipboard is stored for the first time.

```
[Command]
Name=Move Images to Other Tab
Input=image/png
Automatic=true
Remove=true
Icon=\xf03e
Tab=&Images
```

13.14 Copy Clipboard to Window Tabs

Following command automatically adds new clipboard to tab with same name as title of the window where copy operation was performed.

```
[Command]
Name=Window Tabs
Command="copyq:
    item = unpack(input())
    window_title = item[\"application/x-copyq-owner-window-title\"]
    if (window_title) {
        // Remove the part of window title before dash
        // (it's usually document name or URL).
        tabname = str(window_title).replace(/.* (-|\x2013) /, \"\")
        tab(\"Windows/\" + tabname)
        write(\"application/x-copyq-item\", input())
    }
    "
```

(continues on next page)

(continued from previous page)

```
Input=application/x-copyq-item
Automatic=true
Icon=\xf009
```

13.15 Quickly Show Current Clipboard Content

Quickly pop up notification with text in clipboard using `Win+Alt+C` system shortcut.

```
[Command]
Name=Show clipboard
Command="
    copyq:
    seconds = 2;
    popup(\"\", clipboard(), seconds * 1000)"
GlobalShortcut=Meta+Alt+C
```

13.16 Replace All Occurrences in Selected Text

```
[Command]
Name=Replace in Selection
Command="
    copyq:
    // Copy without changing X11 selection (on Windows you can use "copy" instead).
    function copy2() {
      try {
        var x = config('copy_clipboard')
        config('copy_clipboard', false)
        try {
          copy.apply(this, arguments)
        } finally {
          config('copy_clipboard', x)
        }
      } catch(e) {
        copy.apply(this, arguments)
      }
    }

    copy2()
    var text = str(clipboard())

    if (text) {
      var r1 = 'Text'
      var r2 = 'Replace with'
      var reply = dialog(r1, '', r2, '')

      if (reply) {
        copy2(text.replace(new RegExp(reply[r1], 'g'), reply[r2]))
        paste()
      }
    }"
Icon=\xf040
GlobalShortcut=Meta+Alt+R
```

13.17 Copy Nth Item

Copy item in row depending on which shortcut was pressed. E.g. Ctrl+2 for item in row "2".

```
[Command]
Name=Copy Nth Item
Command="
    copyq:
    var shortcut = str(data(\"application/x-copyq-shortcut\"))
    var number = shortcut ? shortcut.replace(/^\\D+/g, '') : currentItem();
    selectItems(number)
    copy(\"application/x-copyq-item\", pack(getItem(number)))"
InMenu=true
Icon=\xf0cb
Shortcut=ctrl+1, ctrl+2, ctrl+3, ctrl+4, ctrl+5, ctrl+6, ctrl+7, ctrl+8, ctrl+9,␣
↪ctrl+0
GlobalShortcut=meta+shift+w, meta+shift+e, meta+shift+q, DISABLED
```

13.18 Edit File

Opens file referenced by selected item in external editor (uses "External editor command" from "History" config tab).

Works with following path formats (some editors may not support all of these).

- `C:/...`
- `file://...`
- `~...` (some shells)
- `%...%...` (Windows environment variables)
- `$...` (environment variables)
- `/c/...` (gitbash)

```
[Command]
Name=Edit File
Match=^([a-zA-Z]:[\\\\/]|~|file://|%\\w+%|$\\w+|/)
Command="
    copyq:
    var editor = config('editor')

    var fileName = str(input())
      .replace(/^\\/([a-zA-Z])\\//, '$1:/')
      .replace(/^file:\\/\\//, '')

    hide()
    execute(editor, fileName)"
Input=text/plain
InMenu=true
Icon=\xf040
Shortcut=f4
```

13.19 Change Monitoring State Permanently

Disables clipboard monitoring permanently, i.e. the state is restored when clipboard changes even after application is restarted.

Should be the first automatic command in the list of commands so other commands are not invoked.

```
[Command]
Automatic=true
Command="
    copyq:
    var option = 'disable_monitoring'
    var disabled = str(settings(option)) === 'true'

    if (str(data('application/x-copyq-shortcut'))) {
      disabled = !disabled
      settings(option, disabled)
      popup('', disabled ? 'Monitoring disabled' : 'Monitoring enabled')
    }

    if (disabled) {
      disable()
      ignore()
    } else {
      enable()
    }"
GlobalShortcut=meta+alt+x
Icon=\xf05e
Name=Toggle Monitoring
```

13.20 Show Window Title

Shows source application window title for new items in tag ("Tags" plugin must be enabled in "Items" config tab).

```
[Command]
Automatic=true
Command="
    copyq:
    var window = str(data('application/x-copyq-owner-window-title'))
    var tagsMime = 'application/x-copyq-tags'
    var tags = str(data(tagsMime)) + ', ' + window
    setData(tagsMime, tags)"
Icon=\xf009
Name=Store Window Title
```

13.21 Show Copy Time

Shows copy time of new items in tag ("Tags" plugin must be enabled in "Items" config tab).

```
[Command]
Automatic=true
Command="
```

(continues on next page)

(continued from previous page)

```
    copyq:
    var time = dateString('yyyy-MM-dd hh:mm:ss')
    setData('application/x-copyq-user-copy-time', time)

    var tagsMime = 'application/x-copyq-tags'
    var tags = str(data(tagsMime)) + ', ' + time
    setData(tagsMime, tags)"
Icon=\xf017
Name=Store Copy Time
```

13.22 Mark Selected Items

Toggles highlighting of selected items.

```
[Command]
Command="
    copyq:
    var color = 'rgba(255, 255, 0, 0.5)'
    var mime = 'application/x-copyq-color'

    var firstSelectedItem = selectedItems()[0]
    var currentColor = str(read(mime, firstSelectedItem))
    if (currentColor != color)
      setData(mime, color)
    else
      removeData(mime)"
Icon=\xf1fc
InMenu=true
Name=Mark/Unmark Items
Shortcut=ctrl+m
```

13.23 Change Upper/Lower Case of Selected Text

```
[Command]
Command="
    copyq:
    if (!copy())
      abort()

    var text = str(clipboard())

    var newText = text.toUpperCase()
    if (text == newText)
      newText = text.toLowerCase()

    if (text == newText)
      abort();

    copy(newText)
    paste()"
GlobalShortcut=meta+ctrl+u
```

(continues on next page)

(continued from previous page)

```
Icon=\xf034
Name=Toggle Upper/Lower Case
```

CHAPTER 14

Backup

This page describes how to back up tabs, configuration and commands in CopyQ.

14.1 Back Up Manually

To back up all the data, close the application first and copy configuration directory.

Path to configuration can be retrieved by running following command.

```
copyq info config
```

- Windows: `%APPDATA%\copyq`
- Portable version for Windows: `config` sub-folder in unzipped application directory
- Linux: `~/.config/copyq`

To restore the backup, close the application and replace the configuration directory.

14.2 Export and Import

In CopyQ 3.0.0 you can easily export selected tabs and optionally configuration and commands within the application.

Important: Tabs are always exported **unencrypted** and if a tab is synchronized with directory on disk the files themselves won't be exported.

To export the data click "Export…" in "File" menu and select what to export, confirm with OK button and select file to save the stuff to.

To restore the data click "Import…" in "File" menu, select file to import and select what to import.

Import won't overwrite existing tabs commands but create new ones.

Alternatively you can use command line for export and import everything (selection dialogs won't be opened).

```
copyq exportData {FILE/PATH/TO/EXPORT}
copyq importData {FILE/PATH/TO/IMPORT}
```

CHAPTER 15

Writing Raw Data

Application allows you to save any kind of data using *drag and drop* or scripting interface.

To add an image to `Images` tab you can run:

```
cat image1.png | copyq tab Images write image/png -
```

This works for any other MIME data type (though unknown formats won't be displayed properly).

CHAPTER 16

Scripting API

CopyQ provides scripting capabilities to automatically handle clipboard changes, organize items, change settings and much more.

In addition to features provided by Qt Script there are following *functions*, *types*, *objects* and *MIME types*.

16.1 Execute Script

The scripts can be executed:

- from commands (in Action or Command dialogs – F5, F6 shortcuts) if the first line starts with `copyq:`,
- from command line as `copyq eval '<SCRIPT>'`,
- from command line as `cat script.js | copyq eval -`,
- from command line as `copyq <SCRIPT_FUNCTION> <FUNCTION_ARGUMENT_1> <FUNCTION_ARGUMENT_2> ...`.

When run from command line, result of last expression is printed on stdout.

Command exit values are:

- 0 - script finished without error,
- 1 - `fail()` was called,
- 2 - bad syntax,
- 3 - exception was thrown.

16.2 Command Line

If number of arguments that can be passed to function is limited you can use

```
copyq <FUNCTION1> <FUNCTION1_ARGUMENT_1> <FUNCTION1_ARGUMENT_2> \
          <FUNCTION2> <FUNCTION2_ARGUMENT> \
              <FUNCTION3> <FUNCTION3_ARGUMENTS> ...
```

where `<FUNCTION1>` and `<FUNCTION2>` are scripts where result of last expression is functions that take two and one arguments respectively.

E.g.

```
copyq tab clipboard separator "," read 0 1 2
```

After `eval` no arguments are treated as functions since it can access all arguments.

Arguments recognize escape sequences `\n` (new line), `\t` (tabulator character) and `\\` (backslash).

Argument `-e` is identical to `eval`.

Argument `-` is replaced with data read from stdin.

Argument `--` is skipped and all the remaining arguments are interpreted as they are (escape sequences are ignored and `-e`, `-`, `--` are left unchanged).

16.3 Functions

Argument list parts `...` and `[...]` are optional and can be omitted.

String version()
: Returns version string.

String help()
: Returns help string.

String help(*searchString*, ...)
: Returns help for matched commands.

show()
: Shows main window.

show(*tabName*)
: Shows tab.

showAt()
: Shows main window under mouse cursor.

showAt(*x*, *y*[, *width*, *height*])
: Shows main window with given geometry.

showAt(*x*, *y*, *width*, *height*, *tabName*)
: Shows tab with given geometry.

hide()
: Hides main window.

bool toggle()
: Shows or hides main window.

  Returns true only if main window is being shown.

menu()
: Opens context menu.

menu(*tabName*[, *maxItemCount*[, *x*, *y*]])
Shows context menu for given tab.

This menu doesn't show clipboard and doesn't have any special actions.

Second argument is optional maximum number of items. The default value same as for tray (i.e. value of `config('tray_items')`).

Optional arguments x, y are coordinates in pixels on screen where menu should show up. By default menu shows up under the mouse cursor.

exit()
Exits server.

disable(*), enable(*)
Disables or enables clipboard content storing.

bool monitoring()
Returns true only if clipboard storing is enabled.

bool visible()
Returns true only if main window is visible.

bool focused()
Returns true only if main window has focus.

filter(*filterText*)
Sets text for filtering items in main window.

ignore()
Ignores current clipboard content (used for automatic commands).

This does all of the below.

- Skips any next automatic commands.
- Omits changing window title and tray tool tip.
- Won't store content in clipboard tab.

ByteArray clipboard([*mimeType*])
Returns clipboard data for MIME type (default is text).

Pass argument `"?"` to list available MIME types.

ByteArray selection([*mimeType*])
Same as `clipboard()` for Linux/X11 mouse selection.

bool hasClipboardFormat(*mimeType*)
Returns true only if clipboard contains MIME type.

bool hasSelectionFormat(*mimeType*)
Same as `hasClipboardFormat()` for Linux/X11 mouse selection.

bool copy(*text*)
Sets clipboard plain text.

Same as `copy(mimeText, text)`.

bool copy(*mimeType, data, [mimeType, data]...*)
Sets clipboard data.

This also sets `mimeOwner` format so automatic commands are not run on the new data and it's not store in clipboard tab.

Exception is thrown if clipboard fails to be set.

Example (set both text and rich text):

```
copy(mimeText, 'Hello, World!',
     mimeHtml, '<p>Hello, World!</p>')
```

bool copy()
Sends `Ctrl+C` to current window.

Exception is thrown if clipboard doesn't change (clipboard is reset before sending the shortcut).

ByteArray copySelection(...)
Same as `copy(...)` for Linux/X11 mouse selection.

paste()
Pastes current clipboard.

This is basically only sending `Shift+Insert` shortcut to current window.

Correct functionality depends a lot on target application and window manager.

Array tab()
Returns array of with tab names.

tab(*tabName*)
Sets current tab for the script.

E.g. following script selects third item (index is 2) from tab "Notes".

```
tab('Notes')
select(2)
```

removeTab(*tabName*)
Removes tab.

renameTab(*tabName*, *newTabName*)
Renames tab.

String tabIcon(*tabName*)
Returns path to icon for tab.

tabIcon(*tabName*, *iconPath*)
Sets icon for tab.

count(), *length()*, *size()*
Returns amount of items in current tab.

select(*row*)
Copies item in the row to clipboard.

Additionally, moves selected item to top depending on settings.

next()
Copies next item from current tab to clipboard.

previous()
Copies previous item from current tab to clipboard.

add(*text*, ...)
Adds new text items to current tab.

Throws an exception if space for the items cannot be allocated.

insert(*row*, *text*)
Inserts new text items to current tab.

remove(*row*, ...)
Removes items in current tab.

Throws an exception if some items cannot be removed.

edit(*[row|text]* ...)
Edits items in current tab.

Opens external editor if set, otherwise opens internal editor.

ByteArray read([mimeType]);()
Same as `clipboard()`.

ByteArray read(mimeType, row, ...);()
Returns concatenated data from items.

Pass argument `"?"` to list available MIME types.

write(*row, mimeType, data, [mimeType, data]...*)
Inserts new item to current tab.

Throws an exception if space for the items cannot be allocated.

change(*row, mimeType, data, [mimeType, data]...*)
Changes data in item in current tab.

If data is `undefined` the format is removed from item.

String separator()
Returns item separator (used when concatenating item data).

separator(*separator*)
Sets item separator for concatenating item data.

action()
Opens action dialog.

action(*row*, ..., *command*, *outputItemSeparator*)
Runs command for items in current tab.

popup(*title*, *message*[, *time=8000*])
Shows popup message for given time in milliseconds.

If `time` argument is set to -1, the popup is hidden only after mouse click.

notification(...)
Shows popup message with icon and buttons.

Each button can have script and data.

If button is clicked the notification is hidden and script is executed with the data passed as stdin.

The function returns immediatelly (doesn't wait on user input).

Special arguments:

- '.title' - notification title
- '.message' - notification message (can contain basic HTML)
- '.icon' - notification icon (path to image or font icon)
- '.id' - notification ID - this replaces notification with same ID
- '.time' - duration of notification in milliseconds (default is -1, i.e. waits for mouse click)
- '.button' - adds button (three arguments: name, script and data)

Example:

```
notification(
      '.title', 'Example',
      '.message', 'Notification with button',
      '.button', 'Cancel', '', '',
      '.button', 'OK', 'copyq:popup(input())', 'OK Clicked'
      )
```

exportTab (*fileName*)
Exports current tab into file.

importTab (*fileName*)
Imports items from file to a new tab.

String config ()
Returns help with list of available options.

String config (*optionName*)
Returns value of given option.

Throws an exception if the option is invalid.

String config (*optionName*, *value*)
Sets option and returns new value.

Throws an exception if the option is invalid.

String config (*optionName*, *value*, ...)
Sets multiple options and return list with values in format `optionName=newValue`.

Throws an exception if there is an invalid option in which case it won't set any options.

String info ([*pathName*])
Returns paths and flags used by the application.

E.g. following command prints path to configuration file.

```
copyq info config
```

Value eval (*script*)
Evaluates script and returns result.

Value source (*fileName*)
Evaluates script file and returns result of last expression in the script.

This is useful to move some common code out of commands.

```
// File: c:/copyq/replace_clipboard_text.js
replaceClipboardText = function(replaceWhat, replaceWith)
{
    var text = str(clipboard())
    var newText = text.replace(replaceWhat, replaceWith)
    if (text != newText)
        copy(newText)
}
```

```
source('c:/copyq/replace_clipboard_text.js')
replaceClipboardText('secret', '*****')
```

String currentPath ([*path*])
Get or set current path.

String str(*value*)
Converts a value to string.

If ByteArray object is the argument, it assumes UTF8 encoding. To use different encoding, use `toUnicode()`.

ByteArray input()
Returns standard input passed to the script.

String toUnicode(*ByteArray*, *encodingName*)
Returns string for bytes with given encoding.

String toUnicode(*ByteArray*)
Returns string for bytes with encoding detected by checking Byte Order Mark (BOM).

ByteArray fromUnicode(*String*, *encodingName*)
Returns encoded text.

ByteArray data(*mimeType*)
Returns data for automatic commands or selected items.

If run from menu or using non-global shortcut the data are taken from selected items.

If run for automatic command the data are clipboard content.

ByteArray setData(*mimeType*, *data*)
Modifies data for `data()` and new clipboard item.

Next automatic command will get updated data.

This is also the data used to create new item from clipboard.

E.g. following automatic command will add creation time data and tag to new items.

```
copyq:
var timeFormat = 'yyyy-MM-dd hh:mm:ss'
setData('application/x-copyq-user-copy-time', dateString(timeFormat))
setData(mimeTags, 'copied: ' + time)
```

E.g. following menu command will add tag to selected items.

```
copyq:
setData('application/x-copyq-tags', 'Important')
```

ByteArray removeData(*mimeType*)
Removes data for `data()` and new clipboard item.

Array dataFormats()
Returns formats available for `data()`.

print(*value*)
Prints value to standard output.

abort()
Aborts script evaluation.

fail()
Aborts script evaluation with nonzero exit code.

setCurrentTab(*tabName*)
Focus tab without showing main window.

selectItems(*row*, ...)
Selects items in current tab.

String selectedTab()
Returns tab that was selected when script was executed.

See *Selected Items*.

[row, ...] **selectedItems**()
Returns selected rows in current tab.

See *Selected Items*.

Item selectedItemData(*index*)
Returns data for given selected item.

The data can empty if the item was removed during execution of the script.

See *Selected Items*.

bool setSelectedItemData(*index*, *Item*)
Set data for given selected item.

Returns false only if the data cannot be set, usually if item was removed.

See *Selected Items*.

Item[] selectedItemsData()
Returns data for all selected item.

Some data can empty if the item was removed during execution of the script.

See *Selected Items*.

void setSelectedItemsData(*Item[]*)
Set data to all selected items.

Some data may not be set if the item was removed during execution of the script.

See *Selected Items*.

int currentItem(), *int index()*
Returns current row in current tab.

See *Selected Items*.

String escapeHtml(*text*)
Returns text with special HTML characters escaped.

Item unpack(*data*)
Returns deserialized object from serialized items.

ByteArray pack(*item*)
Returns serialized item.

Item getItem(*row*)
Returns an item in current tab.

setItem(*row*, *item*)
Inserts item to current tab.

String toBase64(*data*)
Returns base64-encoded data.

ByteArray fromBase64(*base64String*)
Returns base64-decoded data.

bool open(*url*, ...)
Tries to open URLs in appropriate applications.

Returns true only if all URLs were successfully opened.

FinishedCommand execute(*argument*, ..., *null*, *stdinData*, ...)
Executes a command.

All arguments after `null` are passed to standard input of the command.

If arguments is function it will be called with array of lines read from stdout whenever available.

E.g. create item for each line on stdout:

```
execute('tail', '-f', 'some_file.log',
        function(lines) { add.apply(this, lines) })
```

Returns object for the finished command or `undefined` on failure.

String currentWindowTitle()
Returns window title of currently focused window.

Value dialog(...)
Shows messages or asks user for input.

Arguments are names and associated values.

Special arguments:

- '.title' - dialog title
- '.icon' - dialog icon (see below for more info)
- '.style' - Qt style sheet for dialog
- '.height', '.width', '.x', '.y' - dialog geometry
- '.label' - dialog message (can contain basic HTML)

```
dialog(
  '.title', 'Command Finished',
  '.label', 'Command <b>successfully</b> finished.'
  )
```

Other arguments are used to get user input.

```
var amount = dialog('.title', 'Amount?', 'Enter Amount', 'n/a')
var filePath = dialog('.title', 'File?', 'Choose File', new File('/home'))
```

If multiple inputs are required, object is returned.

```
var result = dialog(
  'Enter Amount', 'n/a',
  'Choose File', new File(str(currentPath))
  )
print('Amount: ' + result['Enter Amount'] + '\n')
print('File: ' + result['Choose File'] + '\n')
```

Editable combo box can be created by passing array. Current value can be provided using `.defaultChoice` (by default it's the first item).

```
var text = dialog('.defaultChoice', '', 'Select', ['a', 'b', 'c'])
```

List can be created by prefixing name/label with `.list:` and passing array.

```
var items = ['a', 'b', 'c']
var selected_index = dialog('.list:Select', items)
if (selected_index)
    print('Selected item: ' + items[selected_index])
```

Icon for custom dialog can be set from icon font, file path or theme. Icons from icon font can be copied from icon selection dialog in Command dialog or dialog for setting tab icon (in menu 'Tabs/Change Tab Icon').

```
var search = dialog(
  '.title', 'Search',
  '.icon', 'search', // Set icon 'search' from theme.
  'Search', ''
  )
```

Array settings()
Returns array with names of all custom options.

Value settings(*optionName*)
Returns value for an option.

settings(*optionName*)
Sets value for a new option or overrides existing option.

String dateString(*format*)
Returns text representation of current date and time.

See QDateTime::toString() for details on formatting date and time.

Example:

```
var now = dateString('yyyy-MM-dd HH:mm:ss')
```

Command[] commands()
Return list of all commands.

setCommands(*Command[]*)
Clear previous commands and set new ones.

To add new command:

```
var cmds = commands()
cmds.unshift({
        name: 'New Command',
        automatic: true,
        input: 'text/plain',
        cmd: 'copyq: popup("Clipboard", input())'
        })
setCommands(cmds)
```

Command[] importCommands(*String*)
Return list of commands from exported commands text.

String exportCommands(*Command[]*)
Return exported command text.

NetworkReply networkGet(*url*)
Sends HTTP GET request.

Returns reply.

NetworkReply networkPost (*url*, *postData*)
Sends HTTP POST request.

Returns reply.

ByteArray env (*name*)
Returns value of environment variable with given name.

bool setEnv (*name*, *value*)
Sets environment variable with given name to given value.

Returns true only if the variable was set.

sleep (*time*)
Wait for given time in milliseconds.

ByteArray screenshot (*format='png'*[, *screenName*])
Returns image data with screenshot.

Example:

```
copy('image/png', screenshot())
```

ByteArray screenshotSelect (*format='png'*[, *screenName*])
Same as `screenshot()` but allows to select an area on screen.

16.4 Types

class ByteArray()
Wrapper for QByteArray Qt class.

See QByteArray.

`ByteArray` is used to store all item data (image data, HTML and even plain text).

Use `str()` to convert it to string. Strings are usually more versatile. For example to concatenate two items, the data need to be converted to strings first.

```
var text = str(read(0)) + str(read(1))
```

class File()
Wrapper for QFile Qt class.

See QFile.

Following code reads contents of "README.md" file from current directory.

```
var f = new File("README.md")
f.open()
var bytes = f.readAll()
```

class Dir()
Wrapper for QDir Qt class.

See QDir.

class TemporaryFile()
Wrapper for QTemporaryFile Qt class.

See QTemporaryFile.

```
var f = new TemporaryFile()
f.open()
f.setAutoRemove(false)
popup('New temporary file', f.fileName())
```

class Item(*Object*)
Type is `Object` and each property is MIME type with data.

Example:

```
var item = {}
item[mimeText] = 'Hello, World!'
item[mimeHtml] = '<p>Hello, World!</p>'
write(mimeItems, pack(item))
```

class FinishedCommand(*Object*)
Type is `Object` and properties are:

- `stdout` - standard output
- `stderr` - standard error output
- `exit_code` - exit code

class NetworkReply(*Object*)
Type is `Object` and properties are:

- `data` - reply data
- `error` - error string (set only if an error occurred)
- `redirect` - URL for redirection (set only if redirection is needed)
- `headers` - reply headers (array of pairs with header name and header content)

class Command(*Object*)
Wrapper for a command (from Command dialog).

Properties are same as members of Command struct.

16.5 Objects

arguments
Array for accessing arguments passed to current function or the script (`arguments[0]` is the script itself).

16.6 MIME Types

Item and clipboard can provide multiple formats for their data. Type of the data is determined by MIME type.

Here is list of some common and builtin (start with `application/x-copyq-`) MIME types.

These MIME types values are assigned to global variables prefixed with `mime`.

Note: Content for following types is UTF-8 encoded.

mimeText
Data contains plain text content.

mimeHtml
Data contains HTML content.

mimeUriList
Data contains list of links to files, web pages etc.

mimeWindowTitle
Current window title for copied clipboard.

mimeItems
Serialized items.

mimeItemNotes
Data contains notes for item.

mimeOwner
If available, the clipboard was set from CopyQ (from script or copied items).

Such clipboard is ignored in CopyQ, i.e. it won't be stored in clipboard tab and automatic commands won't be executed on it.

mimeClipboardMode
Contains `selection` if data is from X11 mouse selection.

mimeCurrentTab
Current tab name when invoking command from main window.

Following command print the tab name when invoked from main window.

```
copyq data application/x-copyq-current-tab
copyq selectedTab
```

mimeSelectedItems
Selected items when invoking command from main window.

mimeCurrentItem
Current item when invoking command from main window.

mimeHidden
If set to `1`, the clipboard or item content will be hidden in GUI.

This won't hide notes and tags.

E.g. if you run following, window title and tool tip will be cleared.

```
copyq copy application/x-copyq-hidden 1 plain/text "This is secret"
```

mimeShortcut
Application or global shortcut which activated the command.

```
copyq:
var shortcut = data(mimeShortcut)
popup("Shortcut Pressed", shortcut)
```

mimeColor
Item color (same as the one used by themes).

Examples: #ffff00 rgba(255,255,0,0.5) bg - #000099

mimeOutputTab
Name of the tab where to store new item.

The clipboard data will be stored in tab with this name after all automatic commands are run.

Clear or remove the format to omit storing the data.

E.g. to omit storing the clipboard data use following in an automatic command.

```
removeData(mimeOutputTab)
```

Valid only in automatic commands.

mimeSyncToClipboard
If exists the X11 selection data will be copied to clipboard.

The synchronization will happend after all automatic commands are run.

```
removeData(mimeSyncToClipboard)
```

Valid only in Linux/X11 in automatic commands.

mimeSyncToSelection
If exists the clipboard data will be copied to X11 selection.

The synchronization will happend after all automatic commands are run.

```
removeData(mimeSyncToSelection)
```

Valid only in Linux/X11 in automatic commands.

16.7 Selected Items

Functions that get and set data for selected items and current tab are only available if called from Action dialog or from a command which is in menu.

Selected items are indexed from top to bottom as they appeared in the current tab at the time the command is executed.

CHAPTER 17

Build from Source Code

This page describes how to build the application from source code.

17.1 Get the Source Code

Download the source code from git repository

```
git clone https://github.com/hluk/CopyQ.git
```

or download the latest source code archive from:

- latest release
- master branch in zip
- master branch in tar.gz

17.2 Install Dependencies

The build requires:

- CMake
- Qt

On **Ubuntu** you can install all build dependencies with:

```
sudo apt install cmake qtbase5-private-dev qtscript5-dev qttools5-dev qttools5-dev-
↪tools libqt5svg5-dev
```

Following is optional but recommended:

```
sudo apt install libxfixes-dev libxtst-dev
```

17.3 Build and Install

Build the source code with CMake and make or using an IDE of your choice (see next sections).

```
cd CopyQ    # set root source code directory as current
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local .
make
make install
```

17.4 Qt Creator

Qt Creator is IDE focused on developing C++ and Qt applications.

Install Qt Creator from your package manager or by selecting it from Qt installation utility.

Set up Qt library, C++ compiler and CMake.

See also:

Adding Kits

Open file `CMakeLists.txt` in repository clone to create new project.

17.5 Visual Studio

You need to install Qt for given version Visual Studio.

In Visual Studio 2017 open folder containing repository clone using "File - Open - Folder".

In older versions, create solution manually by running `cmake -G "Visual Studio 14 2015 Win64" .` (select appropriate generator name) in repository clone folder.

See also:

CMake - Visual Studio Generators

CHAPTER 18

Fixing Bugs and Adding Features

This page describes how to build, fix and improve the source code.

18.1 Making Changes

Pull requests are welcome at github project page.

For more info see Creating a pull request from a fork.

Try to keep the code style consistent with the existing code.

18.2 Build the Debug Version

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Debug -DWITH_TESTS=ON ..
make
```

18.3 Run Tests

You can run automated tests if the application is built either in debug mode, with CMake flag `-DWITH_TESTS=ON` or QMake flag `CONFIG+=tests` (releases are usually build with tests).

Run the tests with following command.

```
copyq tests
```

This command will execute all test cases in new special CopyQ session so that user configuration, tabs and items are not modified. It's better to close any other CopyQ session before running tests since they can affect test results.

While running tests there must be **no keyboard and mouse interaction**. Preferably you can execute the tests in separate virtual environment. On Linux you can run the tests on virtual X11 server with `xvfb-run`.

```
xvfb-run sh -c 'openbox & sleep 1; copyq tests'
```

Test invocation examples:

- Print help for tests: `copyq tests --help`
- Run specific tests: `copyq tests commandHelp commandVersion`
- Run specific tests for a plugin: `copyq tests 'PLUGINS:pinned' isPinned`
- Run tests only for specific plugins: `copyq tests 'PLUGINS:pinned|tags'`
- List tests: `copyq tests -functions`
- List tests for a plugin: `copyq tests PLUGINS:tags -functions`
- Less verbose tests: `copyq tests -silent`
- Slower GUI tests: `COPYQ_TESTS_KEYS_WAIT=1000 COPYQ_TESTS_KEY_DELAY=50 copyq tests editItems`

CHAPTER 19

Source Code Overview

This page describes application processes and source code.

19.1 Applications, Frameworks and Libraries

The application is written in C++11 and uses Qt framework.

Source code can be build either with CMake (preferred) or QMake.

Most icons in the application are taken from theme by default (which currently works only on Linux) with fallback to built-in icons provided by FontAwesome.

Application logo was created in Blender (scene source is here).

The logo is used for bigger application icon. Smaller icons were created in Inkscape (icon source is here).

19.2 Application Processes

There are these system processes:

- main GUI application,
- clipboard monitor (started from main application),
- multiple clients (run scripts in main application).

19.2.1 Main GUI Application

The main GUI application (or server) can be executed by running `copyq` binary without attributes (session name can be optionally specified on command line).

It creates local server allowing communication with clipboard monitor process and other client processes.

Each user can run multiple main application processes each with unique session name (default name is empty).

19.2.2 Clipboard Monitor

Clipboard monitoring happens in separate process because otherwise it would block GUI (in Qt clipboard needs to be accessed in main GUI thread). The process is allowed to crash or loop indefinitely due to bugs on some platforms.

Setting and retrieving clipboard can still happen in GUI thread (copying and pasting in various GUI widgets) but it's preferred to send and receive clipboard data using monitor process.

The monitor process is launched as soon as GUI application starts and is restarted whenever it doesn't respond to keep-alive requests.

19.2.3 Clients and Scripting

Scripting language is Qt Script (mostly same syntax and functions as JavaScript).

API is described in *Scripting API*.

A script can be started by passing arguments to `copyq`. This tells the server (main GUI application) to run the script.

After script finishes, the server sends back output of last command and exit code (non-zero if script crashes).

```
copyq eval 'read(0,1,2)' # prints first three items in list
copyq eval 'fail()' # exit code will be non-zero
```

While script is running, it can send print requests to client.

```
copyq eval 'print("Hello, "); print("World!\n")'
```

Scripts can ask for stdin from client.

```
copyq eval 'var client_stdin = input()'
```

The script run in current directory of client process.

```
copyq eval 'Dir().absolutePath()'
copyq eval 'execute("ls", "-l").stdout'
```

Single function call where all arguments are numbers or strings can be executed by passing function name and function arguments on command line. Following commands are equal.

```
copyq eval 'copy("Hello, World!")'
copyq copy "Hello, World!"
```

Getting application version or help mustn't require the server to be running.

```
copyq help
copyq version
```

Scripts run in separate thread and communicate with main thread by calling methods on an object of `ScriptableProxy` class. If called from non-main thread, these methods invoke a slot on an `QObject` in main thread and pass it a function object which simply calls the method again.

```
bool ScriptableProxy::loadTab(const QString &tabName)
{
    // This section is wrapped in an macro so to remove duplicate code.
    if (!m_inMainThread) {
        // Callable object just wraps the lambda so it's possible to send it to a␣
↪slot.
```

(continues on next page)

(continued from previous page)

```
        auto callable = createCallable([&]{ return loadTab(tabName); });

        m_inMainThread = true;
        QMetaObject::invokeMethod(m_wnd, "invoke", Qt::BlockingQueuedConnection, Q_
↪ARG(Callable*, &callable));
        m_inMainThread = false;

        return callable.result();
    }

    // Now it's possible to call method on an object in main thread.
    return m_wnd->loadTab(tabName);
}
```

19.3 Platform-dependent Code

Code for various platforms is stored in src/platform.

This leverages amount of `#ifs` and similar preprocessor directives in common code.

Each supported platform implements PlatformNativeInterface and `createPlatformNativeInterface()`.

The implementations can contain:

- creating Qt application objects,
- clipboard handling (for clipboard monitor),
- focusing window and getting window titles,
- getting system paths,
- setting "autostart" option,
- handling global shortcuts (**note:** this part is in qxt/).

For unsupported platforms there is simple implementation to get started.

19.4 Plugins

Plugins are built as dynamic libraries which are loaded from runtime plugin directory (platform-dependent) after application start.

Code is stored in plugins.

Plugins implement interfaces from src/item/itemwidget.h.

To create new plugin just duplicate and rewrite an existing plugin. You can build the plugin with `make {PLUGIN_NAME}`.

19.5 Continuous Integration (CI)

The application binaries and packages are built and tested on multiple CI servers.

- Travis CI

- Builds packages for OS X.
- Builds and runs tests for Linux binaries with Qt 4.
- GitLab CI
- Builds and runs tests for Ubuntu 16.04 binaries with Qt 5.
- Screenshots are taken while GUI tests are running. These are available if a test fails.
- AppVeyor
- Builds installers and portable packages for Windows with Qt 5.
- Provides downloads for recent commits.
- Release build are based on gcc-compiled binaries (Visual Studio builds are also available).
- OBS Linux Packages
- Builds release packages for various Linux distributions.
- Beta OBS Linux Packages
- Builds beta and unstable packages for various Linux distributions.
- Coveralls
- Contains coverage report from tests run with Travis CI.

CHAPTER 20

Translations

Translations can be done either via Weblate (preferred) or by using Qt utilities.

All GUI strings should be translatable. This is indicated in code with `tr("Some GUI text", "Hints for translators")`.

20.1 Adding New Language

To add new language for the application follow these steps.

1. Edit `copyq.pro` and add file name for new language (`translations/copyq_<LANGUAGE>.ts`) to `TRANSLATIONS` variable.
2. Create new language file with `lupdate copyq.pro`.
3. Add new language file to Git repository.
4. Translate with Weblate service or locally with `linguist translations/copyq_<LANGUAGE>.ts`.

CHAPTER 21

Text Encoding

This page serves as concept for adding additional CopyQ command line switch to print and read texts in UTF-8 (i.e. without using system encoding).

Every time the bytes are read from a command (standard output or arguments from client) the input is expected to be either just series of bytes or text in system encoding (possibly Latin1 on Windows). But texts/strings in CopyQ and in clipboard are UTF-8 formatted (except some MIME types with specified encoding).

When reading system-encoded text (MIME starts with "text/") CopyQ re-encodes the data from system encoding to UTF-8. That's not a problem if the received data is really in system encoding. But if you send data from Perl with the UTF-8 switch, CopyQ must also know that UTF-8 is used instead of system encoding.

The same goes for other way. CopyQ sends texts back to client or to a command in system encoding so it needs to convert these texts from UTF-8.

As for the re-encoding part, Qt 5 does nice job transforming characters from UTF-8 but of course for lot of characters in UTF-8 there is no alternative in Latin1 and other encodings.

CHAPTER 22

Customize and Build the Windows Installer

22.1 Translations

Most of the translations for the installer are taken directly from the installer generator Inno Setup (http://www.jrsoftware.org/isinfo.php).

You can add translations for CopyQ-specific messages in `shared/copyq.iss`. Just copy lines starting with `en.` from `[Custom Messages]` section and change prefix to `de.` (for german translation).

22.2 Modify and Test Installation

Normally the installation file is generated automatically by Appveyor which executes appveyor-after-build.bat to generate portable app folder from build files and runs Inno Setup (the last line).

So you basically don't have to build the app, you just need: - the unzipped portable version of the app, - clone of this repository and - Inno Setup.

Open shared/copyq.iss in Inno Setup and add few lines at the beginning of the file.

```
#define AppVersion 2.8.1-beta
#define Source C:\path\to\CopyQ-repository-clone
#define Destination C:\path\to\CopyQ-portable
```

You should now be able to modify the file in Inno Setup and run it easily.

Index

Symbols

A

B

C

D

E

F

H

I

M

N

P

R

S

T

V

W