Getting Started

1 Requirements 3
   1.1 Speaking English .................................................. 3
   1.2 Knowing how to use Git ........................................... 3
   1.3 Knowing how to use Github ..................................... 3
   1.4 Running Linux ...................................................... 4

2 Set Up 5
   2.1 Create a Sandbox .................................................. 5
   2.2 Install mint-dev-tools ............................................ 5

3 Technology 7
   3.1 Computer Languages ............................................. 7
   3.2 GNOME Toolkit and libraries .................................... 8
   3.3 Tools ................................................................. 8

4 Mint tools 11
   4.1 mint-common ...................................................... 11
   4.2 mintbackup ........................................................ 11
   4.3 mintdesktop ...................................................... 11
   4.4 mintdrivers ........................................................ 12
   4.5 mininstall ........................................................ 12
   4.6 mintlocale ......................................................... 12
   4.7 mintmenu ........................................................... 12
   4.8 mintnanny .......................................................... 17
   4.9 mintreport .......................................................... 17
   4.10 mintsources ...................................................... 18
   4.11 mintstick ........................................................... 18
   4.12 mintsystem ....................................................... 18
   4.13 mintupdate ....................................................... 18
   4.14 mintupload ........................................................ 21
   4.15 mintwelcome ..................................................... 21

5 Cinnamon 23
   5.1 Processes .......................................................... 23
   5.2 Libraries ........................................................... 24
   5.3 Core components .................................................. 25
   5.4 Visible desktop layer ............................................. 26
6 XApps
6.1 libxapp .................................................. 30
6.2 python-xapp ............................................. 30
6.3 xed ......................................................... 30
6.4 xviewer .................................................... 30
6.5 xplayer ..................................................... 32
6.6 xreader .................................................... 32
6.7 pix ........................................................ 32
6.8 blueberry .................................................. 32
6.9 slick-greeter ............................................. 35
6.10 lightdm-settings ........................................ 35

7 Building .................................................................. 37
7.1 Downloading the source code ................................ 37
7.2 Building a project for the first time ....................... 37
7.3 Building a project ............................................... 38
7.4 Respecting the build order .................................... 38

8 Daily Builds .......................................................... 39
8.1 Adding the PPA ............................................... 39
8.2 Upgrading software to unstable versions ................... 39
8.3 Reporting bugs .................................................. 39
8.4 Downgrading back to stable .................................. 40

9 Coding Guidelines .................................................. 41
9.1 Coding Style .................................................... 41

10 Cinnamon Javascript Optimization Techniques ............... 43
10.1 Notes ........................................................ 43
10.2 Reducing the overall number of gsettings signal listeners 43
10.3 Using declared functions in signal listeners ............... 43
10.4 Factorizing callbacks ........................................ 44
10.5 Grouping properties in smaller objects .................... 44
10.6 Using hashmaps vs properties ............................... 44
If you want to help us develop Linux Mint, you’ve come to the right place!
Welcome to the Linux Mint Development Guide.
You don’t need much to get started. If you know more than us you’ll teach us, if we know more than you, we’ll teach you :) 
That said, there are a few things you’ll require before going further. Let’s go through them.

### 1.1 Speaking English

Developers come from all around the World, if you speak English you’ll be able to work with anybody. 
You’re probably OK if you’re here reading this guide. You don’t need to be fluent or to have great English, but you need to understand English enough to communicate.

### 1.2 Knowing how to use Git

Git is the version control system we’re using to keep track of changes. We’re using it all the time and everywhere.
If you don’t know about Git, stop right there, you need to learn it.
To learn Git, visit Github.io.
Make sure you’re familiar with the concepts of commits, branches, remotes, reverts and rebases.

**Hint:** If you’re new to Git, enjoy! It’s both easy and really fun.
If you ask developers around the team, most will tell you Git is by far their favorite toy.

### 1.3 Knowing how to use Github

We’re using Github to host our Git repositories and to work together on the code.
You’ll need to have a Github account set up. To open a Github account, visit Github.com. You’ll also need to know how to use Github to browse code changes, to fork a project, to make pull requests, etc. To set up your Github account properly and learn how to use Github, visit the Github Help.

1.4 Running Linux

For most projects, you’ll need a computer running the latest version of Linux Mint or LMDE. You can run an earlier version, or a different distribution, but if you run the latest Linux Mint or LMDE release you’re guaranteed everything will work.
This chapter explains how to get your computer set up.

## 2.1 Create a Sandbox

When you build projects it produces .deb packages in their parent directory, so it’s a good idea to create a directory for all your development needs, in which you’ll have subdirectories for each project, or each group of projects. This keeps things tidy as well and well organized in your computer so it becomes easier to search for code across different projects.

We commonly call our main development directory “Sandbox” and place it in our home folder.

```
mkdir ~/Sandbox
```

Of course, you can call your “Sandbox” whatever you want and place it anywhere you want as well.

## 2.2 Install mint-dev-tools

Install the `mint-dev-tools` package from the Linux Mint repositories.

```
apt update
apt install mint-dev-tools --install-recommends
```

It contains useful tools to help you compile and develop Linux Mint projects.
This chapter gives you an overview of the technology we’re using.

### 3.1 Computer Languages

We use a variety of computer languages in Linux Mint.

You don’t need to know them all and you don’t need to know them well. It really depends on which project you want to work and what you want to achieve.

Here are the languages we use the most.

#### 3.1.1 Python

Scripts which run in terminals or in the backgrounds are usually either written in Bash or in Python.

Some software applications and most configuration tools are also written in Python.

The advantage of Python is that it is easy to learn and fast to develop with.

#### 3.1.2 C

Many software applications and most libraries are written in C.

The C language is low-level, hard to master and tedious to develop with, but it gives fast performance and it’s most supported language in Linux (everything is accessible from C).

#### 3.1.3 Javascript

The graphical elements of Cinnamon, as well as Cinnamon applets, desklets and extensions are written in Javascript.
3.1.4 Vala

Vala is used in Slick Greeter (the login screen).

3.2 GNOME Toolkit and libraries

All our user interfaces use GTK3 toolkit.
Our development relies heavily on the GNOME libraries, in particular we use Gio, GLib, GObject and dconf a lot.
In C we access these libraries directly.
In Python and Javascript we access them via GObject Introspection.

3.3 Tools

3.3.1 Development environment

To write and edit code, you can use anything you want. Some people prefer lightweight editors while others prefer full-fledge IDEs. It’s a matter of taste. Development is all about fun, so what matters the most is that you love the tools you use.

If you’re not sure what to use, have a look around and try a few editors/IDE until you find your favorite one.
Many developers within the team use Sublime.

```
apt update
apt install sublime-text
```

If you install Sublime, also install its Package Control and then use it to install the GitGutter and TrailingSpaces extensions.

Visual Studio Code is also very popular within the team.
You can also check Atom, Brackets and Geany.

And if you want a complete IDE, there’s also Eclipse and Netbeans.

3.3.2 Version control

There’s less choice when it comes to version control because we’re all using git and nothing else. All our code is version-controlled with it.

That said, you don’t necessarily have to use the git command line.

Here are a few tools you can use to make using git easier.

```
gitk
```

is ugly and looks dated (it was developed in Tcl/Tk) but it’s very useful to quickly look at the commit history, to create branches and to cherry pick.

You can install it from the repositories:
From a project directory, simply type `gitk` to see the history of commits. You can also specify a branch name to see that branch instead, or a subdirectory to only see the history of a particular directory.

`gitg` is very similar and it looks better (it’s using Gtk), but its feature set is slightly different.

From the repository you can also look at `git-cola` and `git-gui`.

If you’re looking for a more complete solution, have a look at Gitkraken.

And last but not least, check the plugins and features available in your IDE/editor. Visual Studio Code, Atom and Sublime in particular come with a lot of support for Git and Github.

### 3.3.3 Glade

We can write our user interfaces in programming language, or we can use Glade and draw them with the mouse.

**Glade** is a tool to design and edit GTK user interfaces and save them in XML (in a .glade or .ui file).

Once a user interface is saved, we simply tell our program to open that file and we can access the widgets from it programmatically.

Many of our projects separate the code from the user interface.

### 3.3.4 devhelp

Devhelp shows the reference manuals for the development libraries installed on your computer. For most libraries, the documentation is included in their `-dev` or `-doc` package (for instance, if you’re working with GTK3, make sure to install `libgtk-3-dev` and `libgtk-3-doc`).

You can launch DevHelp from the applications menu and use it to browse or search the libraries reference manuals. You’ll often need to check the syntax or the arguments of a particular function. It’s nice to be able to get the information locally without having to search online.

### 3.3.5 d-feet

Some programs use DBus to communicate with others. We use d-feet to browse and troubleshoot DBus.
With d-feet you can quickly find a service on DBus, browse its interface and even call some of its functions manually.

### 3.3.6 meld

Meld is a visual diff tool. It shows the differences between two files and it’s great at it.

```bash
apt update
apt install meld
```

### 3.3.7 Other cool tools

Most of our configuration is stored in dconf and we use gsettings (from the command line) to look at it or modify it. If you want to do graphically, you can install dconf-editor.

awf is useful to test widgets when working on GTK themes.

```bash
apt update
apt install awf dconf-editor
```
Mint tools

The first Mint tools were developed around 2006 when the Linux Mint project was born. Throughout the years, new tools were added to Linux Mint to implement functions that it was missing, or to make the user experience easier and more comfortable.

Some tools, which were very useful in the past, also disappeared when they were no longer needed. Here’s a list of the currently active tools projects developed by Linux Mint.

4.1 mint-common

Common utility functions and libraries used by the Mint tools are placed in the mint-common project.

This project is developed in Python and its source code is available on Github.

4.2 mintbackup

The Backup Tool, mintbackup, makes it easy to save and restore backups of files within the home directory.

It also supports the ability to save the list of installed packages, so they can be reinstalled later.

This project is developed in Python and its source code is available on Github.

4.3 mintdesktop

This is a tool which provides some additional settings for the MATE desktop environment and the ability to switch window managers.

This project is developed in Python and its source code is available on Github.
4.4 mintdrivers

The *Driver Manager*, mintdrivers, makes it easy to install proprietary drivers when applicable. It relies on the *ubuntu-drivers* backend and isn’t available in LMDE.

This project is developed in Python and its source code is available on [Github](https://github.com).

4.5 mintinstall

The *Software Manager*, mintinstall, is an App store for Free Software. It provides access to popular applications from within the repository.

It’s also compatible with Flatpak and able to list flatpaks from multiple flatpak repositories.

This project is developed in Python and its source code is available on [Github](https://github.com).

4.6 mintlocale

The mintlocale project provides two configuration tools.

The first one is dedicated to locale selection and installation.

The second one is dedicated to input methods:

This project is developed in Python and its source code is available on [Github](https://github.com).

4.7 mintmenu

This is the main application menu for the MATE edition of Linux Mint.
Fig. 2: Desktop Settings
Fig. 3: Driver Manager
Fig. 4: Software Manager

Fig. 5: Language Settings

4.7. mintmenu
To write in Hiragana, Katakana and Kanji follow the steps below:

- Install the language support packages: Already installed
- Change the input method framework to Fiatx.
- Log out and log back in.
- Right-click the Fiatx applet in the system tray and choose "Configure".
- In the Fiatx configuration screen, add the Mozc input method (uncheck "Only Show Current Language" if necessary).
- Switch between input methods with Ctrl+Space.

Note: Mozc can also be used with the IBus input method.

Fig. 6: Input Methods

Fig. 7: MintMenu
This project is developed in Python and its source code is available on Github.

### 4.8 mintnanny

The *Domain Blocker*, mintnanny, blocks outgoing traffic towards chosen domain names using `/etc/hosts`.

![Domain Blocker](image)

**Fig. 8: Domain Blocker**

This project is developed in Python and its source code is available on Github.

### 4.9 mintreport

The *System Reports*, mintreport, provides system information and helps the user collect information about application crashes.

This project is developed in Python and its source code is available on Github.
4.10 mintsources

The Software Sources configuration tool, mintsources, helps the user configure software repositories, choose a mirror, add PPAs and perform maintenance tasks related to package management.

This project is developed in Python and its source code is available on Github.

4.11 mintstick

The mintstick project provides two utilities.
The first one is dedicated to formatting USB sticks.
The second one is used to make live USB sticks from ISO images:

This project is developed in Python and its source code is available on Github.

4.12 mintsystem

This project provides small utilities, as well as files, scripts and resources used by the OS.

4.13 mintupdate

The Update Manager, mintupdate, provides users with software and security updates.
Fig. 10: Software Sources Configuration Tool

Fig. 11: USB Stick Formatter
Fig. 12: USB Image Writer

Fig. 13: Update Manager
This project is developed in Python and its source code is available on Github.

4.14 mintupload

The Upload Manager, mintupload, allows the user to upload files to a particular location, without browsing it, just by dropping the files with the mouse.

![Fig. 14: Welcome Screen](image)

This project is developed in Python and its source code is available on Github.

4.15 mintwelcome

The Welcome Screen, mintwelcome, welcomes new users into Linux Mint and guides them through their first steps. This project is developed in Python and its source code is available on Github.
Fig. 15: Welcome Screen
The Cinnamon desktop environment is a very large development project.

Between 2006 and 2010 the main desktop environment for Linux Mint was GNOME 2. It was very stable and very popular.

In 2011, Linux Mint 12 was unable to ship with GNOME 2. The upstream GNOME team had released a brand new desktop (GNOME 3 aka “Gnome Shell”) which was using new technologies (Clutter, GTK3), which had a completely different design and implemented a radically different paradigm than its predecessor but which used the same namespaces and thus it couldn’t be installed alongside GNOME 2. Following the decision from Debian to upgrade GNOME to version 3, GNOME 2 was no longer available in Linux Mint.

To tackle this issue two new projects were started:

- A project called “MATE” was started by a developer called Perberos. Its goal was to rename and repackage GNOME 2 so that it could be just as it was before.
- A project called “MGSE” was started by Linux Mint. Its goal was to develop extensions for GNOME 3 to give it back some of the functionality it had lost and which was available in GNOME 2 (a panel, a systray, an application menu, a window-centric alt-tab selector, a window-list, etc).

Linux Mint 12 shipped with both MATE and GNOME3+MGSE.

6 months later and after a huge amount of work, MATE was becoming stable, and from a set of extensions MGSE became a fork of GNOME 3 called Cinnamon.

Linux Mint 13 was the first Linux release to ship with the Cinnamon desktop. Since then Linux Mint has a MATE and a Cinnamon edition, both providing users with a conservative desktop paradigm, one forked from GNOME 2 and the other forked and derived from GNOME 3.

### 5.1 Processes

![Fig. 1: Binary view of the various processes within a Cinnamon session](image-url)
The figure above shows the various processes at play within a Cinnamon session.

After you log in, the following processes are automatically started:

- cinnamon-session (the session manager which starts all the other processes)
- cinnamon (which is the visual part of the cinnamon desktop)
- nemo-desktop (which handles the desktop icons and desktop context menu)
- cinnamon-screensaver (the screensaver)
- various csd-* processes (which are settings daemon plugins and run in the background)

The nemo process starts when you browse files and directories. It remains open as long as at least one file manager window is open.

The cinnamon-settings process starts when you launch the System Settings and remains open as long as at least one configuration module is open.

5.2 Libraries

5.2.1 cinnamon-menus

The cinnamon-menus library provides utility functions to read and monitor the set of desktop applications installed on the computer. Thanks to cinnamon-menus, Cinnamon can quickly list installed applications within the application menu, fetch application icons for the menu, the alt-tab selector and the window-list and keep this data in sync whenever applications are installed or removed from the computer.

The cinnamon-menus library is developed in C and the source code is available on Github.

5.2.2 cinnamon-desktop

Cinnamon-desktop is a set of utility libraries and settings used by other Cinnamon components.

Whenever multiple desktop components need to access the same resource (whether this is a setting or a utility function), we place this resource in cinnamon-desktop.

Here’s an overview of some of the resources currently in cinnamon-desktop:

<table>
<thead>
<tr>
<th>cinnamon.desktop</th>
<th>dconf settings schemas used by several Cinnamon components</th>
</tr>
</thead>
<tbody>
<tr>
<td>libcvc</td>
<td>A PulseAudio utility library used to control sound volume and devices</td>
</tr>
<tr>
<td>gnomerr</td>
<td>An Xrandr utility library to detect, load and save monitor configurations</td>
</tr>
<tr>
<td>gnome-xkb</td>
<td>A keyboard layout utility library</td>
</tr>
<tr>
<td>gnome-bg</td>
<td>A wallpaper utility library</td>
</tr>
<tr>
<td>gnome-installer</td>
<td>A cross-distribution library used to install software applications</td>
</tr>
</tbody>
</table>

The cinnamon-desktop library is developed in C and the source code is available on Github.

5.2.3 muffin

Muffin, or libmuffin to be more precise is a window management library.

Within the Cinnamon desktop environment, the Window Manager isn’t running in a separate process. The main cinnamon process implements the libmuffin library and therefore runs both the visible components (panel, applets..etc) and the window manager.
The muffin package also provides a muffin binary. This binary is a small program which implements libmuffin and provides a minimal window manager, sometimes used by the developers as a troubleshooting tool. Note that whether or not muffin is installed by default in Linux Mint, it doesn’t run by default in a Cinnamon session. The cinnamon process, which also implements libmuffin, is the default window manager.

The clutter and cogl libraries are also part of the muffin package now. Clutter is a library for creating and displaying both 2d and 3d graphical elements. It is used both by muffin itself (eg. for compositing and setting up the stage), and also by St in cinnamon (all St widgets are clutter actors). Cogl is a library that clutter uses for 3d rendering.

Muffin is developed in C and the source code is available on Github.

### 5.2.4 cjs

CJS is Cinnamon’s Javascript interpreter. It uses MozJS (Mozilla’s SpiderMonkey) and makes it possible to work with GObject and interact with GIR, GNOME and Cinnamon libraries using that language.

CJS is run by and within the main cinnamon process and the parts of the desktop written in Javascript are contained in the main Cinnamon component.

CJS is developed in C++ and Javascript and the source code is available on Github.

### 5.3 Core components

#### 5.3.1 cinnamon-session

The Cinnamon session manager is responsible for launching all the components needed by the session after you log in, and closing the session properly when you want to log out.

Among other things, the session manager launches the core components required by the session (such as the desktop itself and its components), as well as applications which are configured to start automatically.

Cinnamon-session also provides a DBus interface called the Presence interface, which makes it easy for applications such as media players to set the sessions as busy and inhibit power management (suspend, hibernate, etc...) and the screensaver during video playback.

Last but not least, the session management lets applications register so they can be closed cleanly. The text editor for instance is registered to the session when launched and interacts with it on logout. If a document isn’t saved, the session is aware of it and lets you save your work before proceeding to log out.

#### 5.3.2 cinnamon-settings-daemon

Cinnamon-settings-daemon is a collection of processes which run in the background during your Cinnamon session.

Here’s a description of some of these processes.
Developer Guide

<table>
<thead>
<tr>
<th>Cinnamon Settings Daemon Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csd-automount</td>
<td>Automatically mounts hardware devices when they are plugged in</td>
</tr>
<tr>
<td>csd-clipboard</td>
<td>Manages the additional copy-paste buffer available via Ctrl+C/Ctrl+V</td>
</tr>
<tr>
<td>csd-housekeeping</td>
<td>Handles the thumbnail cache and keeps an eye on the space available on the disk</td>
</tr>
<tr>
<td>csd-keyboard</td>
<td>Handles keyboard layouts and configuration</td>
</tr>
<tr>
<td>csd-media-keys</td>
<td>Handles media keys</td>
</tr>
<tr>
<td>csd-mouse</td>
<td>Handles mice and touch devices</td>
</tr>
<tr>
<td>csd-orientation</td>
<td>Handles accelerometers and screen orientation</td>
</tr>
<tr>
<td>csd-power</td>
<td>Handles battery and power management</td>
</tr>
<tr>
<td>csd-print-notifications</td>
<td>Handles printer notifications</td>
</tr>
<tr>
<td>csd-wacom</td>
<td>Handles wacom devices</td>
</tr>
<tr>
<td>csd-xrandr</td>
<td>Handles screen resolution and monitors configuration</td>
</tr>
<tr>
<td>csd-xsettings</td>
<td>Handles X11 and GTK configuration</td>
</tr>
</tbody>
</table>

Cinnamon-settings-daemon is developed in C and the source code is available on Github.

5.4 Visible desktop layer

5.4.1 cinnamon-screensaver

The Cinnamon screensaver is responsible for locking the screen and to a lesser extent for handling some power management functions (although most of these are handled by csd-power within the Cinnamon Settings Daemon).

Cinnamon-screensaver is developed in Python and the source code is available on Github.

5.4.2 cinnamon

The Cinnamon github project is the biggest and most active project within the overall project.

It contains various subcomponents written in C:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>St</td>
<td>Cinnamon’s widget toolkit written on top of Clutter</td>
</tr>
<tr>
<td>Appsys</td>
<td>An abstraction of Gio.AppInfo and cinnamon-menus, providing metadata on installed applications</td>
</tr>
<tr>
<td>DocInfo</td>
<td>An abstraction of recently opened documents</td>
</tr>
<tr>
<td>Tray</td>
<td>A small library for managing status icons</td>
</tr>
</tbody>
</table>

The visible layer of the desktop is written in Javascript:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cinnamon JS</td>
<td>The panels, window management, HUD, effects and most of what you see...</td>
</tr>
<tr>
<td>Applets</td>
<td>The applets within the panel</td>
</tr>
<tr>
<td>Desklets</td>
<td>The desklets on top of the desktop</td>
</tr>
</tbody>
</table>

The System Settings, its configuration modules and utility scripts are written in Python.

Cinnamon is developed in C, Python and Javascript and the source code is available on Github.

5.4.3 nemo

Nemo is Cinnamon’s file manager. When you open up your home directory or when browse files you’re running Nemo.
Another little part of Nemo is \textit{nemo-desktop}. Its role is to handle desktop icons and the desktop context menu.

When you log in, \textit{nemo-desktop} is started automatically by cinnamon-session. The \textit{nemo} process itself only starts when you're browsing through the directories and stops when you close the last opened file manager window.

Nemo is developed in C and the source code is available on Github.

\subsection*{5.4.4 \textit{nemo-extensions}}

Nemo provides a set of APIs and is very easy to extend, both in C and in Python. \textit{nemo-extensions} is the Github project where common extensions are stored.

Some Nemo extensions are developed in C and some in Python. Their source code is available on Github.

\subsection*{5.4.5 \textit{cinnamon-control-center}}

Although \textit{cinnamon-settings} (which is part of the Cinnamon project itself) and most of its modules are written in Python. A few configuration modules are still written in C.

\textbf{Note:} Historically, when Cinnamon was forked from GNOME 3, all configuration modules were written in C, as part of gnome-control-center. At the beginning of the Cinnamon project, all configurations modules were thus written in C and were part of cinnamon-control-center. Since then the vast majority of modules were rewritten from scratch in Python and moved to the Cinnamon project itself.

Nowadays, only a few modules are still in cinnamon-control-center:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Color profiles</td>
</tr>
<tr>
<td>datetime</td>
<td>Date and Time configuration</td>
</tr>
<tr>
<td>display</td>
<td>Display and monitors configuration</td>
</tr>
<tr>
<td>network</td>
<td>Network configuration</td>
</tr>
<tr>
<td>online-accounts</td>
<td>Online Accounts configuration</td>
</tr>
<tr>
<td>wacom</td>
<td>Wacom devices configuration</td>
</tr>
</tbody>
</table>

Cinnamon-control-center is developed in C and the source code is available on Github.
A project called “X-Apps” was started in 2016 to produce generic applications for traditional GTK desktop environments.

The idea behind this project is to replace applications which no longer integrate properly outside of a particular environment (this is the case for a growing number of GNOME applications) and to give our desktop environments the same set of core applications, so that each change, each new feature being developed, each little improvement made in one of them will benefit not just one environment, but all of them.

The core ideas for X-Apps are:

- To use modern toolkits and technologies (GTK3 for HiDPI support, gsettings etc.)
- To use traditional user interfaces (titlebars, menubars)
- To work everywhere (to be generic, desktop-agnostic and distro-agnostic)
- To provide the functionality users already enjoy (or enjoyed in the past for distributions which already lost some functionality)
- To be backward-compatible (in order to work on as many distributions as possible)

Within Linux Mint, users didn’t need to adapt to X-Apps, because in many cases, they were very similar or exactly the same as the applications people were already using. For instance, Totem 3.18 was radically different than Totem 3.10 which shipped with Linux Mint 17, but Xplayer 1.0 (which was the default media player in Linux Mint 18) was exactly the same. The goal of the X-Apps is not to reinvent the wheel. Quite the opposite in fact, it’s to guarantee the maintenance of applications we already enjoyed and to steer their development in a direction that benefits multiple desktop environments.

It makes no sense to develop 3 different text editors, 5 different calculators and so on. When we work on projects like these, we want to make it count. An improvement in the text editor shouldn’t benefit only one edition, it should benefit all of them.

All three editions of Linux Mint come with the same XApps libraries and applications. When working on XApps, our development efforts are focused on improving all desktops.
6.1 libxapp

This is the XApps library. Anything that is cross-desktop goes in there. It’s available in Python and JS as well, through GObject Introspection. This project is developed in C and its source code is available on Github.

6.2 python-xapp

This is a small Python library providing extra functionality. This project is developed in Python and its source code is available on Github.

6.3 xed

Xed is based on Pluma and acts as the default text editor.

6.4 xviewer

Xviewer is based on Eye of GNOME and acts as the default image viewer. This project is developed in C and its source code is available on Github.
Fig. 2: Image Viewer
6.5 xplayer

Xplayer is based on Totem and acts as the default media player for music and videos.

![Multimedia Player](image)

Fig. 3: Multimedia Player

This project is developed in C and its source code is available on Github.

6.6 xreader

Xreader is based on Atril and acts as the default document and PDF reader.

This project is developed in C and its source code is available on Github.

6.7 pix

Pix is based on gThumb, which is an application to organize your photos.

This project is developed in C and its source code is available on Github.

6.8 blueberry

The Bluetooth tool, blueberry, is a frontend to gnome-bluetooth with systray support.
Fig. 4: Document Viewer
Fig. 5: Pix

Fig. 6: Bluetooth
The GNOME Bluetooth frontend was removed from gnome-bluetooth and made part of gnome-control-center, essentially making gnome-bluetooth useless outside of GNOME. Blueberry provides that missing frontend and makes it easy for other GTK desktops to use gnome-bluetooth.

This project is developed in Python and its source code is available on Github.

### 6.9 slick-greeter

Slick-greeter is the default login screen, it’s a LightDM greeter originally forked from unity-greeter and modified to work on its own (without gnome-settings-daemon, gnome or unity).

![Fig. 7: Slick Greeter](image)

This project is developed in Vala and its source code is available on Github.

### 6.10 lightdm-settings

The lightdm-settings project provides a configuration tool to set up LightDM and slick-greeter.

This project is developed in Python and its source code is available on Github.
Fig. 8: Login Window
Once you’ve installed mint-dev-tools, building Linux Mint projects from source is extremely easy.

### 7.1 Downloading the source code

Use `git clone` to get the source from github.

For instance, to get the source for mintinstall type:

```
cd ~/Sandbox
git clone https://github.com/linuxmint/mintinstall.git
```

### 7.2 Building a project for the first time

Use `mint-build` to build a project for the first time.

`mint-build` doesn’t just build the project, it also fetches and installs the build dependencies (i.e. the packages which are required for the build to succeed).

To build mintinstall you would type:

```
cd ~/Sandbox/mintinstall
mint-build
```

When the build is complete, the resulting binary .deb packages are located in the parent directory (in this example in ~/Sandbox).
7.3 Building a project

If all the build dependencies are already installed for a particular project (this is done by mint-build the first time you build a project), you can build faster by just calling dpkg-buildpackage.

To build mintinstall you would type:

```
cd ~/Sandbox/mintinstall
dpkg-buildpackage
```

7.4 Respecting the build order

If new changes in the project you’re trying to build require new changes in another project you might need to build and install that other project first.

In general it’s a good idea to build mint-common and xapps first.
If you want to keep up to date with the latest git changes, you don’t need to rebuild everything all the time, you can use the Daily Builds PPA instead.

**Important:** The Daily Builds PPA contains daily builds of the very latest changes pushed by the developers on github. By using this PPA you’ll upgrade from stable to unstable versions which might introduce regressions or not work at all.

### 8.1 Adding the PPA

To add the PPA to your computer, open a terminal and type:

```bash
sudo add-apt-repository ppa:linuxmint-daily-build-team/daily-builds
sudo apt-get update
```

### 8.2 Upgrading software to unstable versions

Use the *Update Manager* to upgrade your software to unstable versions.

### 8.3 Reporting bugs

If you notice a regression, you can report it on the Alpha Testing Github project.

Make sure to read the disclaimers and information provided by this project before reporting issues. This project is only about regressions which appeared after the latest stable version of projects, not issues which already existed before then.
8.4 Downgrading back to stable

To go back to stable versions, either restore a Timeshift snapshot, or remove the PPA, update the APT cache and use the Maintenance -> Downgrade foreign packages section of the Software Sources tool.
9.1 Coding Style

9.1.1 Simplicity

Prefer simple instructions over complicated ones, even if that means using more lines or duplicating code.
Avoid one-liners, complicated conditions, language specificities and abstract/generic patterns.
If your code needs to be explained, comment it.. or even better, rewrite it in a simpler way.

9.1.2 Consistency

Adopt the coding style used in the project you’re contributing to.
This guarantees consistency between your new code and the existing code.

9.1.3 Indentation

In new projects or new files, do not use tabs. Use 4 space characters instead.

9.1.4 Trailing spaces

Do not leave trailing spaces in your code.

Hint: In Sublime, install the TrailingSpaces plugin to automatically highlight trailing spaces and give you the option to easily delete them.
9.1.5 Maximum Line length

Fit your code within 120 columns.
If a line of code is longer than 120 characters, break it into two or more lines.

Hint: In Sublime, select View → Ruler → 120 to show a ruler.
Cinnamon Javascript Optimization Techniques

This is a temporary chapter about Javascript optimization techniques in Cinnamon. It will be part of the dev guide when the Cinnamon design is described and the content of this section will then fit into the right place.

10.1 Notes

Some of these optimization techniques don’t make sense to us and we cannot explain them all, but they were tested methodically.

Jason is the only one in the team to see an impact from them. He’s running a slow CPU with multi-monitors and a low-latency kernel with NVIDIA drivers. These were tested on Cinnamon 4.0. Performance boosts are witnessed in terms of input lag when moving windows and selecting text in Visual Studio Code.

These changes were tested in windowManager.js, an area of Cinnamon which is run constantly and which is prominent within the single execution thread.

10.2 Reducing the overall number of gsettings signal listeners

Here’s an example: https://github.com/linuxmint/Cinnamon/commit/47bef00856e3b1f5a1e1a19e829dec498376d033
Reducing the number of listeners has a significant positive impact on performance.

10.3 Using declared functions in signal listeners

We found out that using:

```javascript
settings.connect('changed::property', (s, k) => { this.property = s.get_int(k); });
```

was slower than:
i.e. declaring a function and referring to that function within the callback was faster than using an anonymous block of instructions in the callback.

The impact was significant.

### 10.4 Factorizing callbacks

```javascript
settings.connect('changed::property', (s, k) => this.setProperty(s, k));
```

was slower than:

```javascript
settings.connect('changed::int_property1', (s, k) => this.setProperty1(s, k));
settings.connect('changed::string_property2', (s, k) => this.setProperty2(s, k));
```

It makes `setProperty()` slower of course, although that’s usually not critical, but it makes the overal project faster.

The impact wasn’t as significant and this optimization is probably only suited to critical paths such as window-manager.js.

### 10.5 Grouping properties in smaller objects

Using `this.smallobject.property` is faster than `this.property`.

The idea is to avoid adding properties to large objects such as `Main.wm`.

So instead of using:

```javascript
Main.wm.desktop_effects_enabled
```

we use:

```javascript
Main.wm.settings.desktop_effects_enabled
```

The impact is positive but subtle and this optimization is probably only suited to critical paths such as window-manager.js.

### 10.6 Using hashmaps vs properties

Using `this.smallobject['property']` is faster than `this.smallobject.property`.

The idea is confirmed by [https://jsperf.com/dot-notation-vs-square-bracket-notation](https://jsperf.com/dot-notation-vs-square-bracket-notation).

So instead of using:

```javascript
Main.wm.desktop_effects_enabled
```

we use:

```javascript
Main.wm.settings['desktop_effects_enabled']
```

The impact is positive but subtle and this optimization is probably only suited to critical paths such as window-manager.js.