# Ambience

Alex Scheufele  
Bryan Tait  
Max Tighe  
Vladimir Ventura

## Overview

Ambience [23] is a cross-platform wallpaper manager. In this case, Ambience allows the user to schedule wallpapers to switch dynamically based on weather conditions. Ambience targets Windows, Linux, and Android. Ambience on Linux currently targets distributions that use the Gnome[21] for their desktop interface (Arch Linux, Debian, Ubuntu, etc). This makes Ambience a handy one-stop shop wallpaper manager given it supports a wide range of platforms.

## Motivation

Ambience was born of the search for a cross-platform wallpaper manager allowing the user to automate system background changes. The parameters we desired would trigger a wallpaper change not only at a particular time of day but also the day of the week and weather conditions. An app called SmartWallpaper once offered this to users, but the app is no longer available to users since being abandoned by the developer [14]. The core functionality of our project revolves around changing a device’s wallpaper programmatically [15] across Windows, Linux, and Android.

Of the applications currently available to users, not one checks every box required. Wallpaper Engine [17] is the most popular wallpaper manager for Windows. While it does offer many features such as scripting and animations for images, it does not include the core features we wish to include in our project, specifically: scheduling background changes by date, time, and weather conditions. Wallpaper Engine is available on Android, but it does not operate without a connection to the desktop software.

Wallpaper Engine’s goals and features are disjoint from that of our project.

Wallpaper Studio [16] is another popular wallpaper manager for Windows, but it requires users to create an account and manually upload their pictures to the cloud. Wallpaper Studio is neither cross-platform nor designed to change the wallpaper based on weather conditions.

WeatherDesk [18] is the most similar utility to Ambience. WeatherDesk integrates with OpenWeather’s API, like Ambience, to schedule wallpaper changes. The tool is written in Python and targets Windows, Linux, and MacOS. WeatherDesk is a command-line interface tool, which limits users to those with increased technical knowledge. Given WeatherDesk’s purpose overlapped with our vision, a code review was conducted to see if a modified version could seed the work to develop Ambience. The consensus of the review was that WeatherDesk is outdated, uses deprecated libraries, and contains code that raises security concerns. For example, WeatherDesk changes the wallpaper on Windows by editing registry keys, posing security and access issues. Ambience requires no elevated permissions to run, so this was a red flag from the onset. Due to these issues, we designated WeatherDesk unfit to act as a starting point for our project.
Currently, there are no cross-platform solutions that fully achieve our goals. The closest program to Ambience is WeatherDesk [18] but users are required to compromise for an insecure CLI experience. All alternatives to Ambience, WeatherDesk included, have some but not all of its features. This is the central motivation for our project. We have integrated the best features of wallpaper managers to deliver the user one cohesive, cross-platform, solution- Ambience.

Objectives

Ambience is a program that changes your device’s wallpaper automatically based on time and weather conditions. The program is lightweight, scheduling wallpaper changes through the host operating system’s API to minimize resource use over time. This design choice means that the visible Ambience executable does not need to be running in order for wallpapers to change successfully. Once a user has specified a condition to trigger a wallpaper change, it is immediately scheduled within the kernel. Ambience gives users a “set it and forget it” feel to their wallpaper changes. Ambience compiles to a single executable for the target system whether it be Windows, Linux, or Android. The user experience across platforms is identical. Customizability and freedom of choice are at the core of what we deliver to the user. We have supplied them with freedom to schedule their wallpaper to change to any of their images, at any specific time of day, on any specific day of the week, and for any specific weather condition.

Interface

The interface of the program is designed for ease of use, allowing for any form of input devices to be used, such as mouse and keyboard, keyboard or mouse exclusively, or touchscreen. Touch screens may interact with displayed components by tapping on them, with the use of an on-screen keyboard to enter text where necessary. The UI can be interacted with through Keyboard alone with the use of the tab key to alternate through interactive on-screen elements, pressing enter to activate fields or buttons. Finally, users may also interact with the UI through the use of a mouse only, by clicking on components with the cursor, and by entering fields through an on-screen keyboard. The usage of a mouse on its own, however, is dependent on the Operating System’s support and provision of mouse-only controls.

To utilize Ambience, users must first log in to their Ambience account, or create a new account. This can be done through the login screen (figure 3.1) that the user first encounters when starting the application. The user must enter their username and password into the appropriate fields, and then select either the “log in” button to log into an existing account, or the “sign up” button, to register a new account. In the event that a user does not enter the correct login credentials for an existing account, or the account does not exist, a warning message will be displayed, informing them that the entered fields are invalid. Additionally, if a user attempts to sign up with a username that already exists, a warning will be displayed, indicating that the account already exists. In the event that the user forgets their password, the
user may click “Forgot Your Password?” to have a password-reset email sent to their account’s email address.

![Login Screen of Ambience](image)

**Figure 3.1, The Login screen of Ambience**

Once the user has logged in, users will be navigated to the main screen (figure 3.2), where users may create a new wallpaper with the Create button, list existing wallpapers with the List button, or quit the program with the Quit button. The main screen displays the current weather, the current time, and the location of the user. Displayed below this information is the current wallpaper, if any, that is shown under the current conditions. There is no way for the location of the user to be automatically retrieved, as some devices lack geolocation capabilities, and the user may not wish to grant the necessary permissions to get them. Because of this, the user must manually set their location by clicking on the button on the top left of the screen, revealing a side-menu (Figure 3.2.1). The user may also log out of their account from this side menu.
Figure 3.2, *The Main screen, showing cloudy weather at 1:35 in Lowell, MA*

Figure 3.2.1, *The revealed Side menu on the left side of the Main screen*

The location screen itself, as seen in figure 3.3, allows the user to enter their country and state through two dropdown menus, and the name of their city in the text field below. The textfield will show autocomplete suggestions as the user enters text. The user may cancel and return to the main screen at any time, however an entered location is required in order for the program to function properly.
If the user clicks the List button in the main screen, they will be taken to the List screen (Figure 3.3), where all of the currently saved wallpapers are displayed. Each entry in the list of saved wallpapers shows its thumbnail, the time and weather it occurs under, as well as a small menu of options the user can perform on the entry. Users have the option to either delete, or edit entries in the List screen by selecting their respective options in the small menu in each entry. Furthermore, users may create a new wallpaper from this screen as well by clicking the “Create” button at the bottom of the screen.

Figure 3.3, *The Location screen, showing Lowell, MA as the entered location*
Figure 3.4, *The List screen, showing three saved wallpapers*

If the user wishes to create a new wallpaper and selects the “Create” button from either the Main screen or the List screen, they will be shown the Create screen (Figure 3.5). Users may enter the days they wish their new wallpaper to occur on by using the toggle buttons at the top of the screen. The user may set the wallpaper to any day(s) of the week, Sunday through Saturday. Users may also enter the time anywhere between 12:00 A.M. to 11:59 P.M., however, Military Time formats are not accepted. Furthermore, the user can set AM or PM, and set the weather via a drop-down menu, with the options such as sunny, cloudy, rainy, snowing, or thunderstorm. Finally, users can select the image they wish to change their wallpaper to by clicking the area beneath the menu, which is empty by default, and turns gray when hovered over. Once the user is finished filling out all of the fields, the user may click the “Confirm” button. In the event that the user has not filled all of the fields correctly, the user will receive a warning. The Create screen is also accessible by the Edit menu options of entries in the List screen, as selecting it will show the Create screen with the selected entry’s information loaded in for the user to change.

Figure 3.5, *The Create screen, set for 4:56 PM during a rainstorm on Mon/Wed/Fri*

**Backend Interface**

Since Ambience is a cross-platform tool, it needs to interact with the three target platforms (Android, Linux, and Windows). Ambience interfaces with both Linux and Windows through their C language API from Dart using Dart’s foreign function interchange module. Flutter and consequently Dart were developed primarily for mobile and thus have extensive wrappers to interact with Android as well.
Ambience’s main way of interacting with the host OS is through daemons, or background programs, that wait until they are summoned to execute. This is done through Windows by an intermediary Powershell script that talks to Task Scheduler (Windows’ daemon manager) to set up a daemon. For Linux, Ambience actually does not talk to the daemon manager (systemd) but instead to the Cron daemon[20]. Cron is a daemon whose job is to execute the provided command at the given time(s) with the given argument(s). It provides a way for the local user to schedule daemons, thus it is a simpler way that doesn’t require elevated permissions. Android has Work Manager[19] and Alarm Manager[21] to manage their daemons. Ambience utilizes Alarm Manager which externally works the same way as Windows and Linux daemons.

Ambience also accepts command line inputs from the OS. In fact, the daemons are doing nothing more than executing Ambience at a certain trigger condition in a headless fashion via the command line. Headless execution means executing without the graphical user interface element. Headless execution allows the daemons to seamlessly run in the background without randomly creating windows whenever the daemons execute. The OS passes the arguments on file in the daemon manager to Ambience through the command line which triggers the headless execution mode.

**Technology**

Flutter [1] is Google’s cross-platform graphical framework. It enables one GUI to work across mobile and desktop platforms, without requiring separate codebases for each platform. Flutter uses a language called Dart [2]. Dart is a platform-independent language that enables a smaller codebase for Ambience by allowing the backend codebase to be shared across platforms through Dart. Dart also offers Just-in-Time compilation [3] which allows for incremental recompilation for rapid testing of code changes, by only recompiling parts of the code that have changed. Dart is also widely supported and expanded with a plethora of libraries enabling interaction with other technologies, like the HTTP request module [4] enabling usage of HTTP APIs.

Ambience uses OpenWeather’s Weather API [7] and also its Geocoder API [8]. They are HTTP [5] request APIs which means they work on all platforms and don’t require the user to download a separate program to utilize them. The Geocoder API resolves the user’s locational input (city name and country code) to longitude and latitude which is then taken by the weather API to return a JSON [9] of weather data. Importantly, the weather API returns a 3-hour step, 5-day forecast which provides an offline backup if the APIs are down or otherwise unreachable. Ambience makes a new call to the weather API every time, so it uses up-to-date weather data, switching to using longer-term forecast data as an offline backup.

Firebase [6] is a set of backend cloud computing services provided by Google. In particular, Ambience leverages Firestore and Firestore. Firestore provides the database functionality for Ambience allowing users to store their Ambience configuration in the cloud and download it on another device or platform. While Firestore doesn’t support images, it does support strings enabling a workaround with an encoding/decoding scheme for images. Firestore handles the accounting system layered over
the database. Fireauthicate also provides password hashing and salting, proving the needed security for the accounting system.

GitHub [13] is used for version control of the Ambience software. GitHub enables the merging of different components allowing easier collaboration in the development of Ambience. GitHub also offers a service for their repositories called GitHub Actions [12] which allows users to execute a group (called “workflow”) of scripts (called “actions”) in a sequential fashion whenever an event occurs inside a repository, including its branches. This is used as our continuous integration pipeline by automatically running our unit tests whenever a branch is proposed to be merged into our master branch through a pull request.

Data/Results

There are several tangible results to show that Ambience is running as per specifications. GitHub Actions keeps track of successful and failed executions for the master branch on the repository, and this history log demonstrates, together with our unit tests, that several components of our project are working as intended [see Appendix Section Results R3 for test results]. Further Ambience has 3 visible actions it does. Firstly, it changes the wallpaper which is readily observable proof of its function. Everything Ambience does is to support the end result of changing the wallpaper. Changing the wallpaper is the last step Ambience does, it takes the user information, processes it, creates rules, creates daemons, and gets weather data. Then the daemons, and only when the daemons check the rules and see that the weather conditions match the conditions specified in the rules will the wallpaper change. Since changing the wallpaper is the last step Ambience does, it is dependent on a lot of other components executing correctly. This makes a successful wallpaper change a very good indicator due to the amount of dependency that the operation has. Secondly, when Ambience schedules daemons on Windows and Linux, those daemons, their arguments, and triggers can be checked and confirmed from Windows’ Task Scheduler [Appendix Section Results Figure R2] and Linux’s [Appendix Section Results Figure R1] Cron daemon respectively. Thirdly, for persistent data across sessions and power cycles Ambience, both write data to file locally and upload data to the cloud. The creation of these files and their respective data is further proof of Ambience’s functioning. The UI loads data from these json files directly including rules[see Figure 3.4 in the UI section] and location data(Visible at the top of the main screen)[See Figure 3.2 in the section]. When using the create rule screen, it updates the JSON which is then read by the list screen to list all current rules. When on the main screen the location data displayed at the top is also pulled from JSON as well. This proves the correct data manipulation and creation operations are being performed on these JSON and validates that the rest of Ambience is working as intended.

Lessons Learned
During the development of this project, we learned multiple lessons that are best taught through firsthand experience. The challenges we faced while developing Ambience brought to light many of the areas where we can improve, both as individuals and as a team.

While writing the code for this project, we each worked on separate components, merging them together when completed. This approach required us to write very clear documentation in order for our components to be written with the correct interactions between them. Furthermore, to minimize the time spent discussing the functionality of each others’ components, we needed to ensure the documentation we wrote for our code was as understandable as possible. This requirement taught us the value of documentation, as the people we had to write documentation for were our teammates, instead of a number of unknown individuals in the future who view our code.

This is our first time writing a full-fledged program, and this is also our first time working together as a team. We have previously worked on our own group projects in previous courses, however, they were of a much smaller scale than this. This project, of this scale and in only a period of fifteen weeks, demanded us to learn how to effectively cooperate in order to complete our program on time.

Furthermore, this is the first project in our careers as software engineers to have support for multiple platforms. As such, we have learned many of the crucial design techniques required in order to support multiple platforms. One major design technique we learned to apply to our program is to make it modular. This allowed for our program to reuse components that are independent of the platform it is on, and switch out components that had to be tailored for a specific platform, such as the daemons and scripts for the Windows platform versus the daemon used for the Android platform.

Future Risks

Like many other programs, Ambience will be subject to many risks in the near future. Firstly, because this project may not see continued support, the program may become deprecated as the systems and the hardware it was initially designed to run on are updated and upgraded, drifting further from the possibility of being still useful on future systems.

Furthermore, as our project becomes public for all to see, others may feel inspired to write their own version of the program. The new programs that others write to behave like ours may outshine it in every way, rendering ours obsolete.

Another risk our project faces is due to the external services it uses, such as Firebase [6], and OpenWeather [7]. These services have the chance to shut down or limit services in the future for many reasons. In fact, the Firedart package used for Firebase ceased development earlier this month. Meaning for the future Ambience will have to switch how it does its cross-platform synchronization or run the risk of losing the ability to do so. Additionally, these services may update their APIs in a fundamental way, leading to our code involving these services to malfunction. In the event that such things occur, our program will not perform as intended, making it ultimately useless.

Finally, there is the potential for our program to reach a large audience, with many users being registered to use our program. While this amount of attention to our program would be nice, it would incur difficulties with the external services our program uses. Our OpenWeather license is currently under a free model and allows us to make a limited number of API calls per minute. Many users running our program at once may cause our license’s limits to be reached, causing the updates of current weather conditions for each user to be delayed. Additionally, our Firebase license is free as well, meaning there is
a storage limit that can be plausibly reached if there are enough users storing enough wallpapers. In the event this limit is reached, users will no longer be able to store any new wallpapers.

Should this program become popular in the future, there may be an incentive to offer a paid version, with continued support and updates. This would easily remedy the risks that would come from the free licenses we use for this program, as some of the revenue of this program can be used for paid licenses for both Firebase and OpenWeather.

Appendix

Results
This section of the appendix contains figures pertaining to demonstrating the functionality of our product on several platforms.

![Figure R1](image)

*Figure R1. Proof of Ambience daemon being successfully scheduled via a listing in Crontab on Ubuntu.*
Figure R2. Proof of Ambience daemon successfully scheduled via a listing in Windows Task Scheduler on Windows.
Figure R3. Proof successful unit testing.
Bibliography


( accessesd Apr. 11, 2023).

https://openweathermap.org/api (accessed Apr. 11, 2023)

https://openweathermap.org/api/geocoding-api (accessed Apr. 11, 2023)


[10]“gRPC,” gRPC. https://grpc.io/ (accessed Apr. 11, 2023)


https://docs.github.com/en/actions (accessed Apr. 11, 2023)


