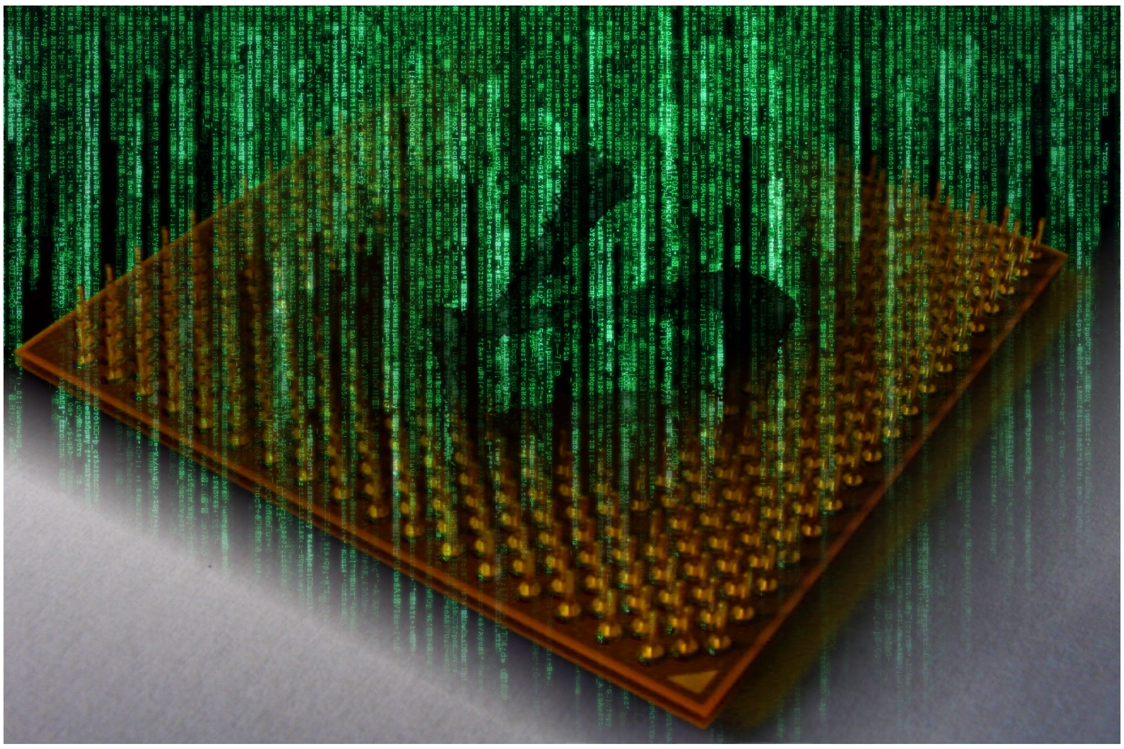
Matura Project 2013

Homemade Tablet

Kantonsschule Zürcher Oberland

Roman Brunner C6b and Dominik Schilling C6b
Tutor: Tobias Kohn

15th January 2013



Info:
Dominik Schilling
Nauenstrasse 30
8632 Tann

Roman Brunner
Schulstrasse 21
8632 Tann

shirotab.info@gmail.com

www.shiroproject.info
www.facebook.com/ShiRoProject
www.twitter.com/ShiRoTab
https://plus.google.com/117822643547183102003
www.youtube.com/user/projectshiro

Mit herzlichem Dank unserem Betreuer Tobias Kohn,
welcher uns bei diesem Projekt stets mit Rat und Tat unterstützte.

Dominik Schilling Roman Brunner

Homemade Tablet

A documentation about building a tablet computer
and programming a linux-based,
touchscreen adapted user interface.

Table of Contents

1 Preface

We are both keen software engineers and often develop programs just for fun, so it was obvious that for our Matura project we would do something in the field of computer science. But why exactly a tablet computer?

In the fourth grade we started to plan a device like the Microsoft Surface table of the first generation. At that point we did not think of doing this as our Matura project, we did it just for fun. We both had some experience in programming, so it was obvious for us to program a touch adapted user interface on our own. The first plans were developed, and after one semester we had a folder full of different ideas for a touch table and for what the software could look like. We improved the cooling system, took a closer look at the mechanical aspects, and finally started to design a touchscreen. Through this project, which unfortunately was never realized, we gained knowledge of what is important when it comes to designing hardware and software for touch enabled devices. It was at this point that the Matura project was beginning to take shape. In the meantime, Apple had released their first tablet and Google had started to ship out their first OS for devices with larger touchscreens. With the appearance of the first iPad, tablets became a significant player on the computer market. We were enthusiastic about the possibilities a tablet computer offered. So we dropped the idea of the touch table and started work on a tablet computer. Now, about nine months later, we have finished the development and are proud that we can present you with a fully functional tablet computer which runs our own linuxbased software.

Now it is time to thank all the people who supported us in the process. First we want to thank Mr Tobias Kohn. He was our tutor for the project and supported us with a lot of good ideas and always had time to discuss with us the problems which had to be solved. Then we would also like to thank Mr Fridolin Berger, who read our paper to help us with the language. We also express thanks to our families. They showed respect for our project and, from time to time, had to put up with a night when they could not sleep calmly. And, last but not least, we wish to thank Mr Claudio Assandri. He was the person in charge of the production of the case for our tablet at Astromec AG. He took a lot of time for us, helped us with the 3D-model of the case, and finished the 3D-model so it was ready to be produced. And he offered us a special price for all that.

2 Introduction

Tablet computers have been around for quite a while. The first devices called tablets came up about ten years ago. They were thick, heavy, had short battery life, and the OS was mainly a desktop browser. So it does not really seem necessary to mention that they were not very popular. Some of them already supported mobile internet services[1], but what was the use of a device that was so slow that it was almost quicker for you to return home and fire up your PC. Two years later Microsoft tried to push tablets to the end customer market. But they had the same problem as the actual Windows OS's: small buttons, a lot of menus, no natural interaction with the content. So Microsoft's first attempt with tablet computers[2] failed. Some other companies tried and did not succeed, until Apple released their first version of the iPad in 2010. It was a kind of revelation; it was just the way a tablet should be. It was thin, light, and had 10 hours of battery life. The OS was so simple that you were tempted to say they developed it for little children. Thanks to the improvements in mobile internet technologies it was possible to browse the internet anywhere, anytime, and to get your content in no time. So the most important things you have to bother about when designing a tablet are: simplicity, long battery life, easy to use interface and connectivity. We took these four points and tried our best to build a tablet that combines these qualities in a useful way. When designing software, hardware or when making plans for the implementation of software from third parties, we kept these four points in mind at all times.

[1]Wireless Application Protocol, WAP or sometimes also called as Wait And Pay
[2]They try it again now with a better adapted UI

3 Objectives

For our Matura project, we set ourselves the following objectives:

- The user interface should be easy to use and understand: Anybody who has ever seen a computer from somewhat nearer than 10 feet should be able to understand how to interact with the software and how to use it productively
- The user interface should support the content so that the user can always focus on things which are important and relevant without being bothered by software issues
- The software-part should be reduced as far as possible and stay compatible with Gnome Shell extensions and programs that could reduce the energy consumption of our software
- Social networks should be integrated right into the UI: for the simple reason that so many users use social networks today, we wanted to make them easy to access or even better, so that you can always see the newest updates
- Building a hardware prototype out of a laptop so that we can show the software on a running touchscreen device
- Promoting our software as well as our project and pushing them into the market

The following points are improvements which we want to provide after finishing the Matura project:

- Fully integrated social networks, also within settings, and the possibility to share everything from everywhere
- Putting our software on ARM devices
- Integrating a modified version of the android emulator so that users can install and use the apps they know already from their android devices
- Porting our software to the newest version of Fedora and then to Ubuntu
- Providing the software through a repository so that we can get updates through to our users and get more users

In the course of this project people often asked us on the internet when we would start selling our tablets and how much they would cost. We were also asked if there would be developers' devices that would be available before a general launch. But as you can see above, we never planned to sell any devices. But if anybody was interested, we would be pleased to see our software on a commercial device. But so far not even the big organisations like KDE[3] and Canonical had any success in getting their software onto commercial tablets.

[3]http://community.kde.org/Plasma

4 Hardware

When working on our project we were often asked, “What is a tablet computer?”, so we will start with a short definition of a tablet computer. It is a flat gadget with just one input device, the screen itself. It is a monolithic block of hardware with the screen on top of it. This form factor is called a slate.

4.1 Hardware Precondition

In computer science you will always find certain limits imposed through physics or hardware specifications. Sometimes, as is the case with tablets, the demand limits the hardware and the hardware limits the software. In mobile computing the biggest problem is for you to turn up with a device which you can use on the go. This means that it needs long battery-life and should not be too big or too heavy. For these reasons you need to adjust the hardware specifications. Most modern tablet computers have very large batteries, they normally take between $\frac{2}{3}$ and $\frac{4}{5}$ of the available space in the case. The rest of the hardware is really compact and you can find nearly everything on one single chip. These systems are called SoC, for System on a Chip. On such a chip you will find everything except for memory and antennas. In most cases, the lingering hardware, i.e. the processor, some co-processors for special use, graphical unit, audio de- and encoder, controller for different interfaces and periphery devices, the busses, the clock and really fast memory (SRAM), are mounted on this chip. This technology saves a lot of space so that there is enough space for the huge battery. Then the manufacturers always try to produce technologies with lower power consumption, otherwise it would not be possible for today’s tablets to run up to 10 hours without charging.

4.2 Hardware Used

For our project we did not have the technology mentioned above because we could not get hold of these parts. If you want to build a computer on your own, you will normally find hardware parts for personal computers and only in some cases parts for notebooks. So cannot be sure to find any parts for tablet computers. We had to be content with normal notebook hardware, which means normal processors[4], no SoC, a small battery and, compared to the tablets, heavy and thick hardware. For our tablet we used the hardware of a HP notebook, the HP Compaq 6820s to be precise. So our hardware specifications are nearly the same as the ones of the HP notebook. Our tablet now has the following specifications [1]:

- Intel Core 2 Duo Processors T8100 / T8300 (2.1 to 2.4 GHz, 800 MHZ FSB, 3 MB L2 cache)
- Battery life of up to two hours, 6-cell Lithium-Ion battery with 55 WHr

[4] we used $\times 86$ instead of ARM processor, but we will talk about that later

- 17.0" WXGA+ BrightView touchpanel(1440 $\times$ 900 resolution)
- 160 GB Serial ATA hard drive (5400 rpm)
- 3072 MB of DDR II RAM at 667 MHz
- Mobile Intel PM965 Chipset
- weight of 6.8 kilograms
- Intel PRO/Wireless 802.11 a/b/g/draft-n and Bluetooth 2.0
- 3 USB 2.0 ports, VGA, stereo microphone in, headphone/line out, RJ-45
- ATI Mobility Radeon X1350
- 128 MB video memory
- mono speaker
- 1.3 MP webcam

For our tablet, we included the webcam and the touchpanel, but we left out some parts which do not fit into a tablet like the keyboard, the touchpad and the DVD drive. We have also thrown out as many parts of the original case as possible and replaced them with our own self-designed case.

4.3 Processor

One of the most important parts of a computer, except for the motherboard (which provides all connections between the separate parts), is the central processing unit or, in short, the CPU. It is the processor with the most control of the system and which does, depending on the design of the system, most calculations by itself. So the CPU is the core of all the action in your computer. As mentioned above, there are two main types of design for processors, the ARM and the x86 architecture. To make this more understandable, we will give approximate explanations of these two architectures and illustrate the main differences. It may seem that we use x86 processors in our computers. At first, x86 and ARM competed each other. x86 was victorious and ARM nearly disappeared until very low voltage processors were needed. Intel, the holder of the rights for the architecture of x86 processors did not develop their processor for low voltage use and ARM came back in mobile devices. ARM processors require less energy because they use a less complicated way of calculation, which in fact means less fast. Even though these processors are slower than the x86 processors, they are not that bad, because the software on mobile devices is adapted to these processors. So they run smoothly. Due to the difference of architecture and the way to execute the calculations, different types of machine languages are necessary. Also for programmer developing applications, which are not running directly on the hardware, it is important to know which architecture their software is programmed for because an application written and compiled for x86 architecture will not run on devices with an ARM processor (except for machine independent languages as web specific languages or high level languages as JAVA). Another plus for ARM processors is that they do not produce much heat while running, which means that they need less energy for cooling. In fact, ARM devices need no active cooling system as do most of the devices with x86 architecture. So it became clear that the first choice for mobile devices as a tablet would be an ARM-based system [2].

4.4 Touchscreen

Nowadays a lot of interaction with computers happens through touchscreens. If you want to travel by train and need a ticket, you buy one from the vending machine, interacting with the machine through the touchscreen; when you use a smartphone, you normally use a touchscreen; when you buy something in the supermarket, the sales assistant uses a touch panel for adding up the prices; and if the postal service delivers some special mail, you have to sign on a touch device. Especially in industry touch screens are often used because you can momentarily upload a new program, which means that you can adapt the device to totally different needs without changing any hardware. The interface of a keyboard is not exchangeable by update. And touchscreen interaction can be much more intuitive than interaction with a computer by keyboard or by mouse because you can actually tap on the thing you want.
So for our tablet we needed a touchscreen too. For that reason we decided to attach a so-called touch panel, which is in fact just the touch technology without the screen. For our purpose we chose a resistive touch panel because it is the simplest way to integrate a touch panel into a system.

As mentioned before, there are different technologies for touchscreens. For this reason we are giving you a short presentation of the most common technologies available:

- Capacitive: Between two films, one made of a conducting material, the other not, an electromagnetic field is laid out. Due to the fact that the the human body is conductive, it interferes with the electromagnetic field. This interference is measured in every corner of the screen. Out of the ratio between the different values measured, the touch controller calculates the exact position of the finger on the screen. Because this technology needs a conductive object to recognize where the touching action takes place, it is not possible to use these devices when wearing gloves or other insulating coverings. The advantage is that this technology registers even very delicate touching events and more than one touching event at the time.

- Resistive: On a resistive touchscreen, a plastic film is attached above a transparent surface made of glass or any other stable material. On the film and on the surface you find a lot of thin wires. These wires are live, so when the upper film is pressed down, it creates contact and the flow of the electricity changes. This change is measurable and so the controller can calculate the position by comparing the different values measured. For our tablet we used a touch panel with this technology. Its advantage is that it can be used with anything.

- FTIR: FTIR stands for Frustrated Total Internal Reflection. This name already nearly explains the basics of this technology. In a two-dimensional coordinate system, an array of infra-red LED's is placed on each side or at least on one X and one Y side of a glass or another transparent surface which is thick enough, so that they shine into the surface. The angle at which the light floats into the surface is important, it should be big enough for the light to get fully reflected internally. Now if something touches the surface, the light is reflected towards the bottom so that it floats out of the surface. Under the surface a camera is attached which recognizes the infra-red light coming from the surface. Now the computer can calculate where the touching action happened by scaling up the pixels of the camera to the pixels of the screen. For our touch table we wanted to use this touchscreen technology. Its advantage is the nearly unlimited number of touchpoints that are recognizable, but it is not applicable for mobile devices. The problem with this technology is the control of reflection caused by possible interference from the environment.
- SAW stands for Surface Acoustic Wave. It works with ultrasonic waves sent across the screen. On all edges there are sensors which check if the signal comes in as expected. If something gets in the way of the wave, it breaks. The object reflects some of the waves and also prevents the waves from colliding with the other waves sent. The sensors now receive a signal, where they can measure the difference of time and recreate the pattern. Out of the deviation of the normal pattern from the one where the user has touched the screen, the computer can extrapolate where the screen has been touched. The big advantage of this technology is that it can be used with anything, gloves, pens, whatever breaks the waves, and that it does not show any wear and tear.

4.5 Case

In our case it took a long time to figure out how we wanted to build it. There were a lot of questions to answer. We had to think of the cooling system, how to attach the hardware to the case, which material to use and mainly how to put it all together. As far as the material is concerned, it was soon clear that we wanted to make our case out of metal because metal is a good heat conductor. But that did not make things easier because we had no possibility of working on metal ourselves. At first we thought that we could probably make the case out of a mailbox. But due to its size and unstable construction it was not possible. So we asked René Weber of PackSysGlobal[5], where custom parts are

[5]http://www.packsysglobal.com/

built, for advice, and he gave us the address of Astromec[6]. They produce metal parts on the basis of a 3D model sent in. So we had to draw a 3D model, but unfortunately neither of us had ever done that before for machine CAD systems, so it took us a long time to figure out how the program worked and what information we had to put into the model. For our purpose we used the student version of AutoCAD 2012. Then we started thinking of how big the case would have to be and how we would attach the hardware. We decided to stick the hardware parts into the case. The case consists of two parts, held together by four screws. To work out the size of the case, we had to do a lot measuring on the laptop. Often we had to dismantle the laptop so as to get the values for just some parts of the notebook. But after we got everything into one 3D file, we sent it in to Astromec. We are very happy, that Mr Claudio Assandri, CEO of Astromec, helped us to finish the file. He added the information we had forgotten and helped us to draw the screw thread in the model. The production of the case took four weeks. Afterwards we received the case by postal service and started with the fitting of the hardware.
We had to change a few things on the original hardware and to saw some parts out, very close to the Motherboard. But when we had finished, we were happy not to have damaged anything. Now that the tablet is finished, we are proud of what it looks like although we are not designers and developed the case mainly for functionality, not design.

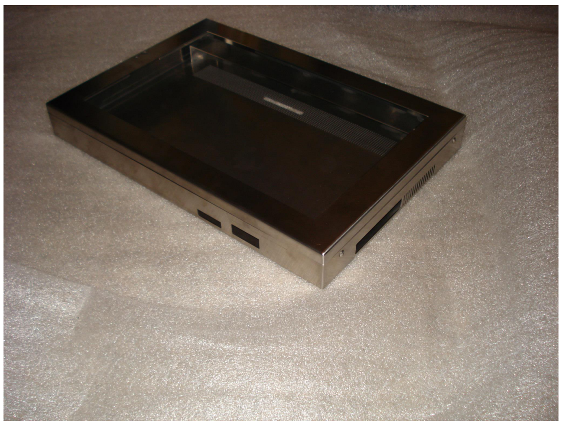


Figure 1: Case

[6]http://astromec.ch/

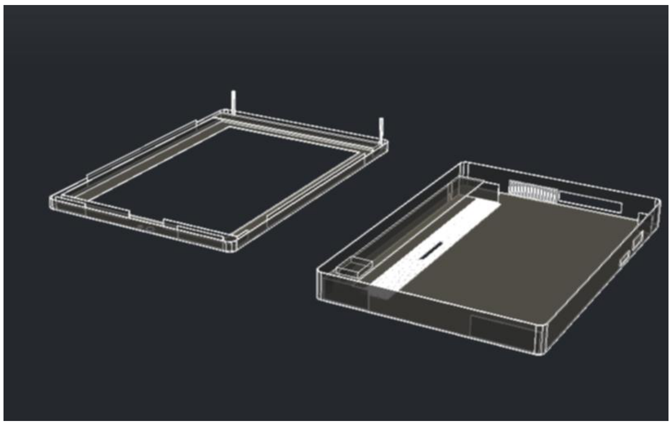


Figure 2: 3D model of the case

5 Software

For our tablet we also needed some touch-friendly software, so we thought of developing our own graphical user interface, called UI in short, based on an existing linux operating system. The user recognizes the software as the most important part of a computer system, for all he sees of the system is the software on the screen. This means that for successfully coding software for users, the software should look nice, should be easy to understand and provide just the functions the user wants, not more and not less.

5.1 General thoughts about touch software

For the programmer this means that he ought to try to find a middle way between all the needs of his users. But even if every programmer and every company tries to accomplish this, totally different results will come out. Before we started programming our UI, we took a closer look at the different mobile tablet OS' that existed. On the basis of these ideas we tried to figure out what would be important to the user of a mobile OS, so that we could program our software for the needs of the users. Apple designed the first really successful software for modern smartphones, adapted it for tablets and were successful again. Why is their software so popular? Because it is really easy to use. If the user gets his hands on an Apple device he does not have to bother about how to use the software for a long time, it takes just a few moments for him to understand how the software works. Then the UI is also minimalistic, showing just the installed applications and some of the most important information. There is nothing irritating the user or distracting him. So we can assume that most users pre-eminently want a UI which is easy to understand.

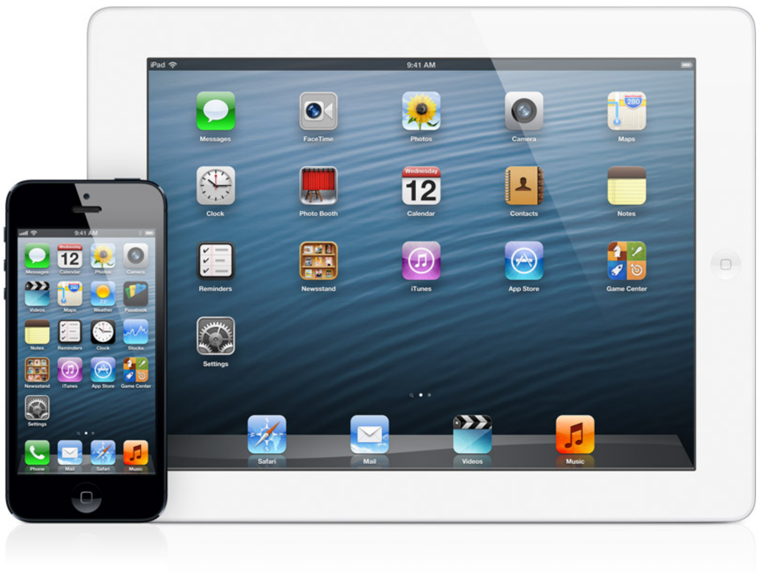
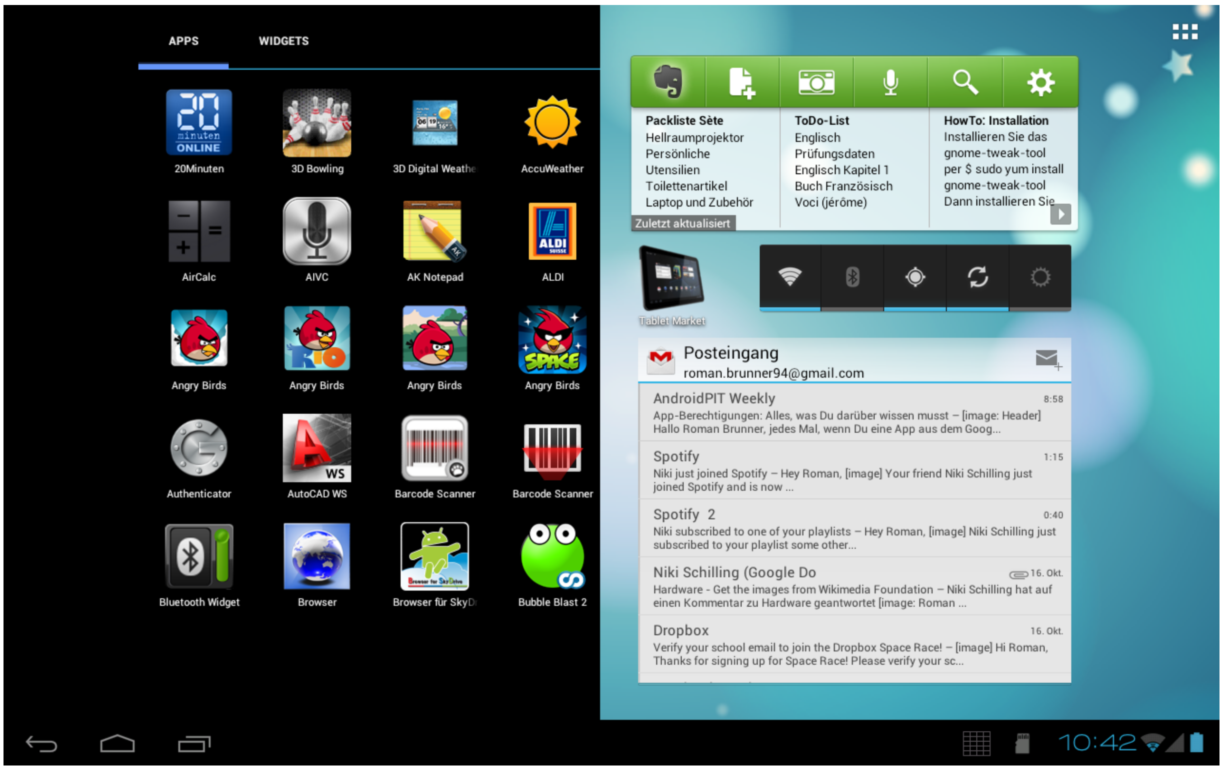


Figure 3: Easily understandable Interface of iOS [3]

For Android the situation is significantly different. It is quite complex with the multiple homescreens, the installed apps view and the difference between apps and widgets on the homescreen. But even this solution, which is more complicated, finds a lot of fans. So why is Android so successful? Firstly, with the widgets, it is possible to see all the important things at one glance, with no need to open an application. Secondly, you can sort in the most important and biggest things on the homescreens. To be fair, in iOS it is also possible to sort the apps, but under Android you can put your things on top with the content and not just the program icon.

And one of the newest players on the market, Microsofts Windows 8, does not yet have any fans, but we will take a closer look to this UI too. The new Start Menu under Windows 8 is in a way the combination of the two ideas presented above. On the one hand, Windows 8 shows the app icons in a free, sortable raster, while providing the most important things on so-called life tiles, where new e-mails or incoming chat messages are displayed. Windows 8 does not show all the installed apps in this menu so as to keep it simple for the user. At first glance, it seems acceptable that it hides these apps. But if you need one of the hidden apps, it gets really hard to find them.



Packliste Sète
Hellraumprojektor
Persönliche
Utensilien
Toilettenartikel
Laptop und Zubehör

ToDo-List
Englisch
Prüfungsdaten
Englisch Kapitel 1
Buch Französisch
Voci (jérôme)

HowTo: Installation
Installieren Sie das
gnome-tweak-tool
per $ sudo yum install
gnome-tweak-tool
Dann installieren Sie

Posteingang
roman.brunner94@gmail.com

AndroidPIT Weekly 8:58
App-Berechtigungen: Alles, was Du darüber wissen musst – [image: Header] Hallo Roman Brunner, jedes Mal, wenn Du eine App aus dem Goog...

Spotify 1:15
Niki just joined Spotify – Hey Roman, [image] Your friend Niki Schilling just joined Spotify and is now ...

Spotify 2 0:40
Niki subscribed to one of your playlists – Hey Roman, [image] Niki Schilling just subscribed to your playlist some other...

Niki Schilling (Google Do 16. Okt.
Hardware - Get the images from Wikimedia Foundation – Niki Schilling hat auf einen Kommentar zu Hardware geantwortet [image: Roman ...

Dropbox 16. Okt.
Verify your school email to join the Dropbox Space Race! – [image] Hi Roman, Thanks for signing up for Space Race! Please verify your sc...

Figure 4: Androids app drawer and part of a home screen with some widgets

Windows 8 and Android have provided some sort of push notification from the beginning, Android with its information bar, showing when something happened, whilst Windows 8 shows up the most important information in the top right corner. iOS did not really have a notification system until version 5 appeared in which Apple introduced the Notification Center, which is similar to the Android notifications. Summing up, we can see that the perfect tablet UI provides three things at the same time: easy access to the installed apps, all the important information at one glance, and a good notification system that does not interrupt.

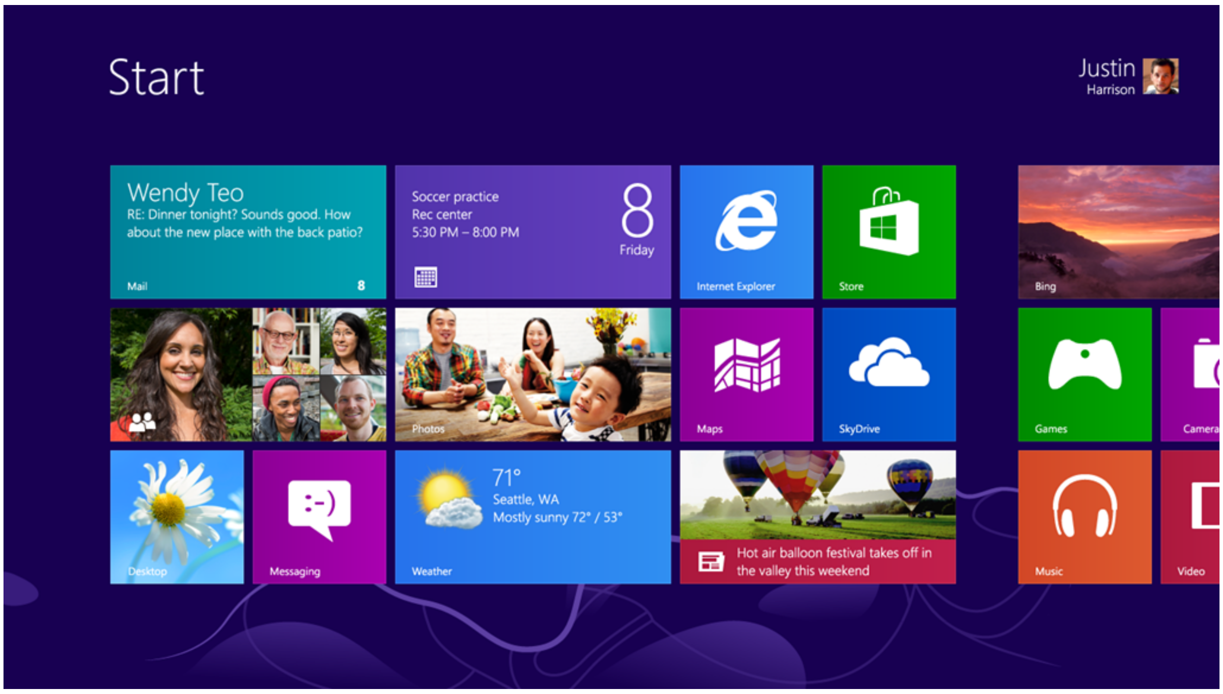


Figure 5: Windows 8: New start menu with program icons and live tiles [4]

5.2 Theoretical thoughts about the Software

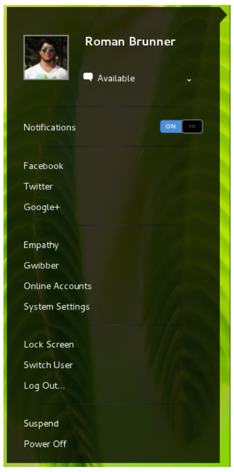


Figure 6: User menu

So we tried to develop a UI that would give easy access to the apps, show the important information at one glance and display the notifications immediately without annoying the user. For hardware reasons, we cannot recognize the screen orientation. So we had to choose one orientation because it is not changeable whilst the system is running. We thought that most of the content should be displayed in landscape mode. Just think of all the videos, most of the pictures and almost any website displayed in-landscape mode. Only text pages like this one and probably some websites are designed to be displayed in portrait mode. So we decided that our software should run in landscape mode.

If you now think of using a tablet in landscape mode, how would you hold the tablet? Normally, you would hold it on both sides with your hands. If you want to interact with the content or the software, in normal mobile OS you have to take one hand and use it for interaction while holding your tablet with the other hand. We found this very impractical and decided to put the most important things on the left and right of the screen. We also decided to put the favorite apps on the left panel and the rest, for interaction with important stuff, on the right panel. There you can find the User Menu which displays options for visibility in chats and access to social networks. Also the settings and the shutdown menu can be accessed through the User Menu. Below the User Menu, the calendar has been placed. We did not change a lot in the calendar itself. Then the Files Menu shows up. There the user has shortcuts to the most important folders, the favorites and the mounted devices on his computer. This menu should help the user to access the files without opening any apps. At the bottom of the screen, the System Tray and the Notification Tray are located. If the user clicks on the Notifications button, they will get a tray with all the important information at one glance. The reason why we did not integrate these pieces of information right into the panel is that if someone uses a lot of different services at the same time, there can be a huge amount of icons and it would not look nice if the whole panel was filled up with these icons. Urgent notifications as the ones from chat or from the calendar are shown up in the lower left corner. The user has the possibility to answer a chat without leaving the actual applications, so they do not have to stop what they are doing at the moment. These pop-ups can be turned off in the User Menu.

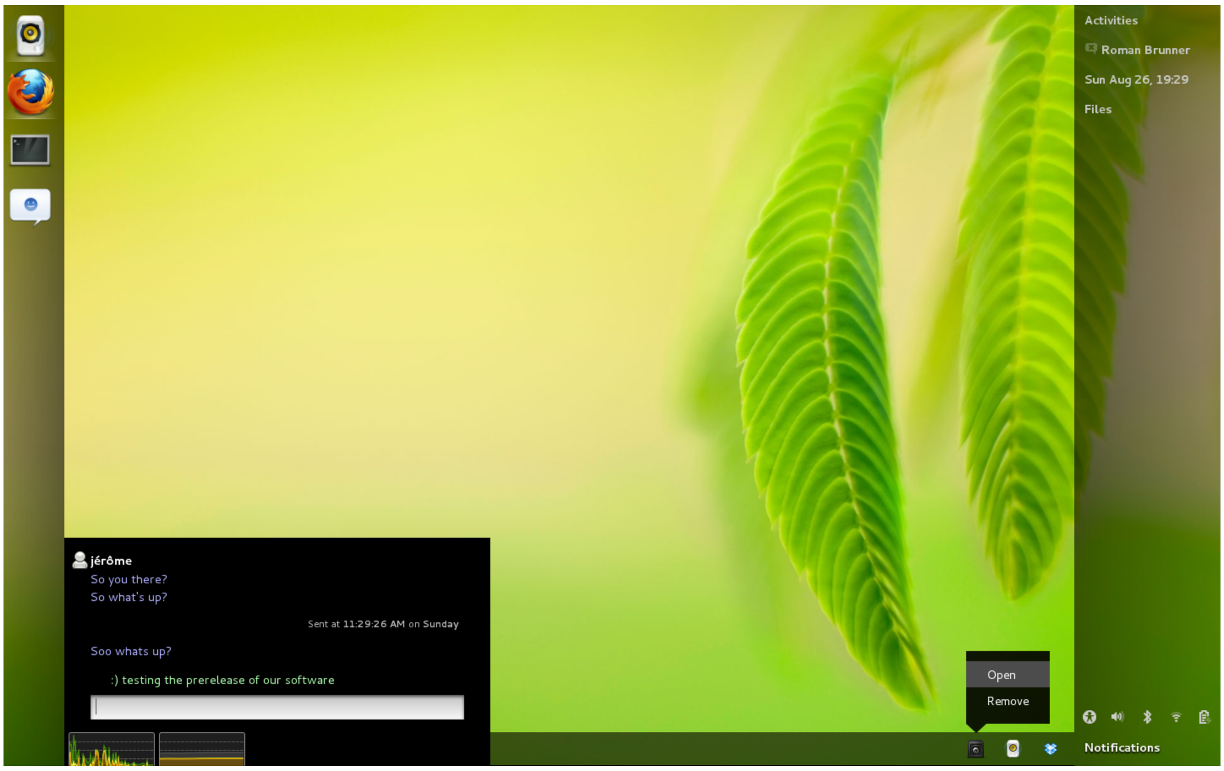

jérôme
So you there?
So what's up?
Sent at 11:29:26 AM on Sunday
Soo whats up?
:) testing the prerelease of our software

Figure 7: Left panel with the favorites, right panel, the notification tray and an urgent chat notification

The content is located in the middle of the screen. As this is the most important thing for the user, the UI should not attract too much attention so that the user is not distracted from the content. To accomplish this, we totally relinquished any 3D effects or round shapes because rounded shapes point towards the middle of the shape and attract the user's attention more than rectangular shapes, which point outwards, in this case towards the content. That the case itself has a rounded shape is OK because it should point towards the content, but the UI should not put more rounded shapes on top.

As we used a Linux distribution, we had to deal with real multitasking. For switching between applications and closing them, the user goes into the activity view or the overview. In the overview two tabs are seen, the windows tab and the applications tab. Under the windows tab all open windows in the actual workspace are shown. If the user wants, they can create a new workspace right away by dragging the window into the blank workspace at the bottom. So the user can sort the running applications into different workspaces. Under the applications tab he finds all the installed applications in one list. On the right side the user has the possibility to sort out just the applications for one field. That is already the whole user interface, and most of the objects should be self-explanatory.

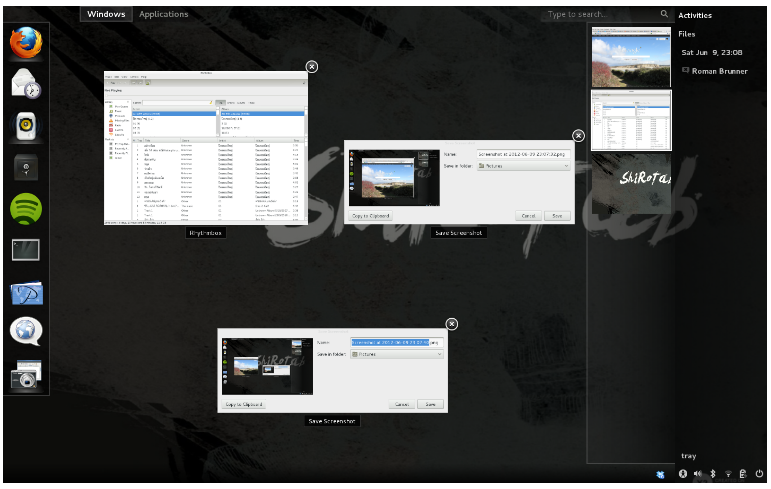


Figure 8: Windows tab in the overview. Next to the right panel the switch for the work-spaces

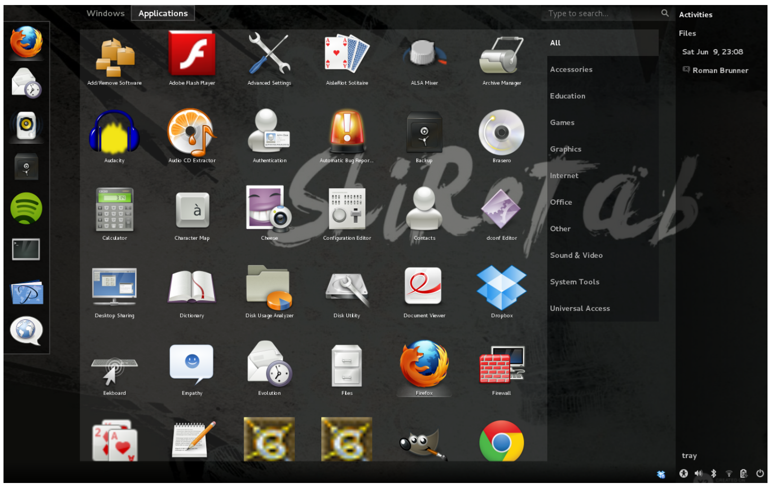


Figure 9: Apps tab with the selector for different fields on the right side

5.3 Technical realisation

Because it was not possible for us to program a UI from scratch, we took an existing project and changed the UI in such a way that it was applicable for tablet devices. After we looked through all Linux desktop environments we knew, we decided to base our software on the newest Gnome version 3, so-called Gnome Shell. Because Fedora came out of the box with a nearly unchanged version of the Gnome Shell, we decided to use Fedora OS for our project. The Gnome Shell already provided the most important functions, so we just added a few functions and mainly changed the design.

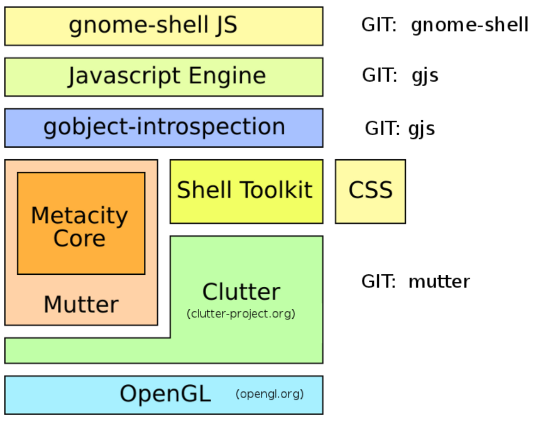


Figure 10: Overview of the different parts of the Gnome Shell [5]

At first we will take a brief look at how the Gnome Shell is designed. The basic libraries are the OpenGL and the Clutter library. The OpenGL library is a graphic library that provides hardware acceleration, which is needed by the Gnome Shell because it supports 3D graphics. Clutter is a library for graphical user interfaces which also provides bindings for gobject-introspection, which is nothing but JavaScript. Mutter is the windows manager and in fact it is a version of Metacity adapted on clutter, the window manager from the previous versions. The Shell Toolkit imports the information from the .css files and adds them to the information from the Gnomeshell

JavaScript. The gobjectintrospection, which is based on GLib Object System, is a library that provides different bindings between C and other programming languages, like JavaScript in this particular case. Then the JavaScript and the runtime environment follow. According to Owen Taylor[7], the developers have chosen JavaScript because it is easier to adapt the Shell to different needs and because it is easier to test the software right away as no recompiling is necessary.

We were glad to have made that decision because we nearly exclusively had to change or add JavaScript files. The source code is provided in appendix C.

5.4 Optimizing

We have also made changes to the system itself, which means not to the program code, but to settings and configuration files. The following changes were made without writing any source code:

- Fixing Fedora 16 Bluetooth bug: Bluetooth was activated, but visibility was not changeable. Solution: Not all required services were started, so we wrote a Shell script which is executed every time the user logs in so that all the services needed are started.
- Making all supporting applications start in fullscreen mode
- Changing the title bar of the windows: As we provide all window options in overview, we do not need the close, maximize and minimize buttons, and the title bar itself is not really useful because seeing the window you know what to do, nobody really looks at the title. So we have deleted the title bar in the fullscreen mode and made it smaller in case the application does not support fullscreen mode as a start option.
- Add AddOns for scrolling in the browser
- Powertop: Powertop is a terminal tool which helped us to improve battery life by disabling all devices running unnecessarily
- Mousepointer: On a touchscreen device you normally have no mousepointers, so we have changed the configuration files of the evdev driver in such a way that if the touchpanel is used, no mousepointer is visible.
- Bootsplash: We have replaced the default bootsplash with our own so that it shows our icon.

[7]For more informations: http://blog.fishsoup.net/2008/10/22/implementing-the-next-Gnome Shell/, 16.10.12, 02.24

6 Final word

We are really happy about what we have achieved with this project. Not only have we developed a new user interface, but we have also built our own tablet device. As we started out, we were not sure how easy it would be, and, in the course of the project, it was sometimes really arduous. When you are sitting there, a few thousand lines of source code to read without any comment, and you have no clue what the file is for, you tend to lose hope. When you try to find a documentation, and you just find a page with some "bla bla bla" at the top, you cannot believe what you see and start to fear that you might never understand it. But we forced ourselves to read the source code over and over again, changing variables and commenting out functions until we understood what a file was for and how it worked. All in all, we are really happy with the product.

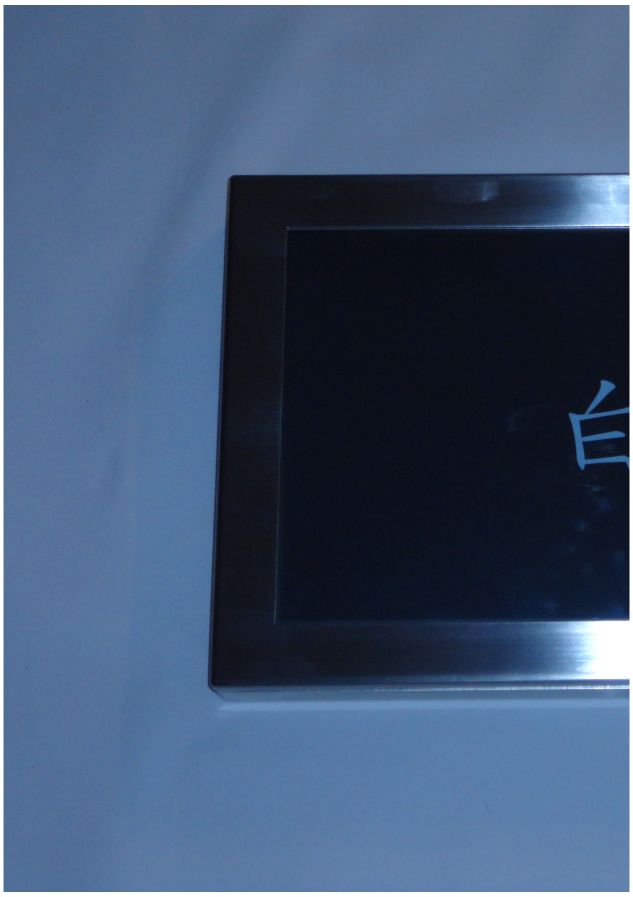


Figure 11: The final product

There are a few points we want to improve in the future. The first one is the integration of social networks. Right now this integration is just good enough, for our first idea was to integrate social networks just as the Google products are integrated. So you log in once in the settings and then the calendar gets you all the appointments on Facebook, and the chat program automatically gets your account information, and so on. We also want to integrate a social stream into the right panel. This stream should also support RSS feeds so that the user can get everything about news and social networks at a glance without leaving the current application. It would probably be nice to integrate the possibility to post a status line to different networks without opening an additional program. But in the six months' time that we had there was no time to integrate social networks so deeply into the operating system, also because we had to change a lot of programs not belonging to the user interface, so that they supported sharing information or getting the information from the settings. So we decided that this was not possible yet and we would just give shortcuts to the corresponding apps for the social networks, which is already sufficient. Another thing that would be nice on a tablet running a Linux operating system would be the direct integration of Android applications. We thought here of a modification of the original Android emulator, so there is no difference for the user between Android applications and normal Linux applications.
We are happy with the hardware because without the right components it was not possible to build a better tablet. But for the future we will have to think of a commercial device where we can run our software. But except for the few points mentioned above, everything works well. And as we are planning to develop the project further, we can correct these things for forthcoming releases.

7 Appendix A: References

1. Hewlett-Packard Development Company: HP, Compaq 6820s Business Notebook PC - specifications and warranty,

   `http://h10010.www1.hp.com/wwpc/ca/en/sm/WF06a/12139188-12139280-12139280-12139280-12434660-80583797.html?dnr=1,`

   14.10.12, 21.57

2. J. F. Amprimoz: ARM vs x86 Processors: What's the Difference?,

   `http://www.brighthub.com/computing/hardware/articles/107133.aspx,`

   15.10.12, 00:15

3. Apple Inc.: Elegant and intuitive interface.,

   `http://images.apple.com/ios/shared/what-is/images/interface.jpg,`

   17.10.12, 11:00

4. Microsoft: Windows 8 has reached the RTM milestone,

   `http://blogs.windows.com/cfs-filesystemfile.ashx/__key/CommunityServer-Blogs-Components-WeblogFiles/00-00-00-59-23metablogapi/2084.Start _5F00_Default_5F00_RTM_5F00_3ROW_5F00_7E38C905.png,`

   17.10.12, 11.16

5. Finnbarr P. Murphy: More Gnome Shell Customization,

   `http://blog.fpmurphy.com/2011/05/more-Gnome Shell-customization.html,`

   16.10.12, 01:33

6. Athlon xp 1600 pins.jpg,

   `http://upload.wikimedia.org/wikipedia/commons/3/3c/Athlon_xp_1600_pins.jpg,`

   17.10.2012, 14:15

7. Wallpaper Stock, for fresh desktops,

   `http://img.wallpaperstock.net:81/matrix-wallpapers_34861_1680x1050.jpg,`

   17.10.12, 14:26

8 Appendix B

List of Figures

For more media check our website or social network channels:
www.shiroproject.info
On Facebook: www.facebook.com/ShiRoProject
On Twitter: www.twitter.com/ShiRoTab
On Google+: https://plus.google.com/117822643547183102003
On Youtube: www.youtube.com/user/projectshiro
On Sourceforge: https://sourceforge.net/projects/shiroproject/

9 Appendix C: Sourcecode

Please note that for our project we just changed and added a few files. The following source code is the whole working program, including all the files needed for running our software. This means that there are a lot of files where we did not change anything. **We will provide these files as a download through our website because they contain more than 800 pages of source code. Download Link: http://goo.gl/x3Puz**
The following source code shows two important files where we have made changes.

/usr/share/gnome-shell/js/ui/layout.js

```
// -*- mode: js; js-indent-level: 4; indent-tabs-mode: nil -*-

const Clutter = imports.gi.Clutter;
const Lang = imports.lang;
const Mainloop = imports.mainloop;
const Meta = imports.gi.Meta;
const Shell = imports.gi.Shell;
const Signals = imports.signals;
const St = imports.gi.St;

const Main = imports.ui.main;
const Params = imports.misc.params;
const ScreenSaver = imports.misc.screenSaver;
const Tweener = imports.ui.tweener;

const HOT_CORNER_ACTIVATION_TIMEOUT = 0.5;
const STARTUP_ANIMATION_TIME = 0.2;
const KEYBOARD_ANIMATION_TIME = 0.5;

function LayoutManager() {
    this._init.apply(this, arguments);
}

LayoutManager.prototype = {
    _init: function () {
        this._rtl = (St.Widget.get_default_direction() == St.TextDirection.RTL);
        this.monitors = [];
        this.primaryMonitor = null;
        this.primaryIndex = -1;
        this._hotCorners = [];
        this._leftPanelBarrier = 0;
        this._rightPanelBarrier = 0;
        this._dashPanelBarrier = 0;
        this._trayBarrier = 0;

        this._chrome = new Chrome(this);

        this.panelBox = new St.BoxLayout({ name: 'panelBox', });
        this.dashPanelBox = new St.BoxLayout({ name: 'dashPanelBox', });
        this.addChrome(this.dashPanelBox, { affectsStruts: true });
        this.dashPanelBox.connect('allocation-changed',
                                Lang.bind(this, this._updateDashPanelBarriers));
        this.addChrome(this.panelBox, { affectsStruts: true });
        this.panelBox.connect('allocation-changed',
                              Lang.bind(this, this._updatePanelBarriers));

        this.trayBox = new St.BoxLayout({ name: 'trayBox' });
        this.addChrome(this.trayBox, { visibleInFullscreen: true });
        this.trayBox.connect('allocation-changed',
                             Lang.bind(this, this._updateTrayBarrier));

        this.keyboardBox = new St.BoxLayout({ name: 'keyboardBox',
                                              reactive: true,
                                              track_hover: true });
        this.addChrome(this.keyboardBox, { visibleInFullscreen: true });
        this._keyboardHeightNotifyId = 0;

        global.screen.connect('monitors-changed',
                              Lang.bind(this, this._monitorsChanged));
        this._monitorsChanged();
    },

    // This is called by Main after everything else is constructed;
    // Chrome.init() needs access to Main.overview, which didn't exist
    // yet when the LayoutManager was constructed.
    init: function() {
        this._chrome.init();

        this._startupAnimation();
    },

    _updateMonitors: function() {
        let screen = global.screen;

        this.monitors = [];
        let nMonitors = screen.get_n_monitors();
        for (let i = 0; i < nMonitors; i++)
            this.monitors.push(screen.get_monitor_geometry(i));

        if (nMonitors == 1) {
            this.primaryIndex = this.bottomIndex = 0;
        } else {
            // If there are monitors below the primary, then we need
            // to split primary from bottom.
            this.primaryIndex = this.bottomIndex = screen.get_primary_monitor();
            for (let i = 0; i < this.monitors.length; i++) {
                let monitor = this.monitors[i];
                if (this._isAboveOrBelowPrimary(monitor)) {
                    if (monitor.y > this.monitors[this.bottomIndex].y)
                        this.bottomIndex = i;
                }
            }
        }
        this.primaryMonitor = this.monitors[this.primaryIndex];
        this.bottomMonitor = this.monitors[this.bottomIndex];
    },

    _updateHotCorners: function() {
        // destroy old hot corners
        for (let i = 0; i < this._hotCorners.length; i++)
            this._hotCorners[i].destroy();
        this._hotCorners = [];

        // build new hot corners
        for (let i = 0; i < this.monitors.length; i++) {
            if (i == this.primaryIndex)
                continue;

            let monitor = this.monitors[i];
            let cornerX = this._rtl ? monitor.x + monitor.width : monitor.x;
            let cornerY = monitor.y;

            let haveTopLeftCorner = true;

            // Check if we have a top left (right for RTL) corner.
            // I.e. if there is no monitor directly above or to the left(right)
            let besideX = this._rtl ? monitor.x + 1 : cornerX - 1;
            let besideY = cornerY;
            let aboveX = cornerX;
            let aboveY = cornerY - 1;

            for (let j = 0; j < this.monitors.length; j++) {
                if (i == j)
                    continue;
                let otherMonitor = this.monitors[j];
                if (besideX >= otherMonitor.x &&
                    besideX < otherMonitor.x + otherMonitor.width &&
                    besideY >= otherMonitor.y &&
                    besideY < otherMonitor.y + otherMonitor.height) {
                    haveTopLeftCorner = false;
                    break;
                }
                if (aboveX >= otherMonitor.x &&
                    aboveX < otherMonitor.x + otherMonitor.width &&
                    aboveY >= otherMonitor.y &&
                    aboveY < otherMonitor.y + otherMonitor.height) {
                    haveTopLeftCorner = false;
                    break;
                }
            }

            if (!haveTopLeftCorner)
                continue;

            let corner = new HotCorner();
            this._hotCorners.push(corner);
            corner.actor.set_position(cornerX, cornerY);
            this._chrome.addActor(corner.actor);
        }
    },
//PanelBox entspricht TopPanel
    _updateBoxes: function() {
        if (this.bottomMonitor.width>this.primaryMonitor.width)
        {
        this.panelBox.set_position(this.primaryMonitor.width
        -this.primaryMonitor.width*0.12, 0);
        this.panelBox.set_size(this.primaryMonitor.width*0.12
        , this.primaryMonitor.height);
        this.dashPanelBox.set_position(0,0);
        this.dashPanelBox.set_size(72, this.primaryMonitor.height);
        }
        else
        {
        this.panelBox.set_position(this.bottomMonitor.width
        -this.bottomMonitor.width*0.12, 0);
        this.panelBox.set_size(this.bottomMonitor.width*0.12,
         this.bottomMonitor.height);
        this.dashPanelBox.set_position(0,0);
        this.dashPanelBox.set_size(72, this.bottomMonitor.height);
        }

        this.keyboardBox.set_position(72,
                                      this.bottomMonitor.y +
                                      this.bottomMonitor.height);
        this.keyboardBox.set_size(this.bottomMonitor.width*0.83, -1);

        this.trayBox.set_position(this.primaryMonitor.x,
                                  this.bottomMonitor.y +
                                  this.bottomMonitor.height);
        this.trayBox.set_size(this.bottomMonitor.width, -1);

        // Set trayBox's clip to show things above it, but not below
        // it (so it's not visible behind the keyboard). The exact
        // height of the clip doesn't matter, as long as it's taller
        // than any Notification.actor.
        this.trayBox.set_clip(0, -this.bottomMonitor.height,
                              this.bottomMonitor.width, this.bottomMonitor.height);
    },
        _updateDashPanelBarriers: function() {
            global.destroy_pointer_barrier(this._dashPanelBarrier);
            let primary = this.primaryMonitor;
            this._dashPanelBarrier = 0;
    },
    _updatePanelBarriers: function() {
        if (this._leftPanelBarrier)
            global.destroy_pointer_barrier(this._leftPanelBarrier);
        if (this._rightPanelBarrier)
            global.destroy_pointer_barrier(this._rightPanelBarrier);

        if (this.panelBox.height) {
            let primary = this.primaryMonitor;
            this._leftPanelBarrier =
                global.create_pointer_barrier(primary.x, primary.y,
                                              primary.x, primary.y +
                                              this.panelBox.height,
                                              1 /* BarrierPositiveX */);
            this._rightPanelBarrier =
                global.create_pointer_barrier(primary.x + primary.width, primary.y,
                                              primary.x + primary.width,
                                              primary.y + this.panelBox.height,
                                              4 /* BarrierNegativeX */);
        } else {
            this._leftPanelBarrier = 0;
            this._rightPanelBarrier = 0;
        }
    },

    _updateTrayBarrier: function() {
        let monitor = this.bottomMonitor;

        if (this._trayBarrier)
            global.destroy_pointer_barrier(this._trayBarrier);

        if (Main.messageTray) {
            this._trayBarrier =
                global.create_pointer_barrier(monitor.x+monitor.width,
                monitor.y+monitor.height-Main.messageTray.actor.height,
                                              monitor.x + monitor.width,
                                              monitor.y + monitor.height,
                                              4 /* BarrierNegativeX */);
        } else {
            this._trayBarrier = 0;
        }
    },

    _monitorsChanged: function() {
        this._updateMonitors();
        this._updateBoxes();
        this._updateHotCorners();

        this.emit('monitors-changed');
    },

    _isAboveOrBelowPrimary: function(monitor) {
        let primary = this.monitors[this.primaryIndex];
        let monitorLeft = monitor.x, monitorRight = monitor.x + monitor.width;
        let primaryLeft = primary.x, primaryRight = primary.x + primary.width;

        if ((monitorLeft >= primaryLeft && monitorLeft < primaryRight) ||
            (monitorRight > primaryLeft && monitorRight <= primaryRight) ||
            (primaryLeft >= monitorLeft && primaryLeft < monitorRight) ||
            (primaryRight > monitorLeft && primaryRight <= monitorRight))
            return true;

        return false;
    },

    get focusIndex() {
        let focusWindow = global.display.focus_window;

        if (focusWindow) {
            let wrect = focusWindow.get_outer_rect();
            for (let i = 0; i < this.monitors.length; i++) {
                let monitor = this.monitors[i];

                if (monitor.x <= wrect.x && monitor.y <= wrect.y &&
                    monitor.x + monitor.width > wrect.x &&
                    monitor.y + monitor.height > wrect.y)
                    return i;
            }
        }

        return this.primaryIndex;
    },

    get focusMonitor() {
        return this.monitors[this.focusIndex];
    },

    _startupAnimation: function() {
        // Don't animate the strut
        this._chrome.freezeUpdateRegions();

        this.panelBox.anchor_y = this.panelBox.height;
        Tweener.addTween(this.panelBox,
                         { anchor_y: 0,
                           time: STARTUP_ANIMATION_TIME,
                      transition: 'easeOutQuad',
                      onComplete: this._startupAnimationComplete,
                      onCompleteScope: this
                    });
},

_startupAnimationComplete: function() {
    this._chrome.thawUpdateRegions();
},

showKeyboard: function () {
    this.keyboardBox.raise_top();
    Tweener.addTween(this.keyboardBox,
                    { anchor_y: this.keyboardBox.height,
                      time: KEYBOARD_ANIMATION_TIME,
                      transition: 'easeOutQuad',
                      onComplete: this._showKeyboardComplete,
                      onCompleteScope: this
                    });
    Tweener.addTween(this.trayBox,
                    { anchor_y: this.keyboardBox.height,
                      time: KEYBOARD_ANIMATION_TIME,
                      transition: 'easeOutQuad'
                    });
},

_showKeyboardComplete: function() {
    // Poke Chrome to update the input shape; it doesn't notice
    // anchor point changes
    this._chrome.updateRegions();

    this._keyboardHeightNotifyId = this.keyboardBox.connect
    ('notify::height', Lang.bind(this, function () {
        this.keyboardBox.anchor_y = this.keyboardBox.height;
        this.trayBox.anchor_y = this.keyboardBox.height;
    }));
},

hideKeyboard: function (immediate) {
    if (this._keyboardHeightNotifyId) {
        this.keyboardBox.disconnect(this._keyboardHeightNotifyId);
        this._keyboardHeightNotifyId = 0;
    }
```

```
        Tweener.addTween(this.keyboardBox,
                         { anchor_y: 0,
                           time: immediate ? 0 : KEYBOARD_ANIMATION_TIME,
                           transition: 'easeOutQuad',
                           onComplete: this._hideKeyboardComplete,
                           onCompleteScope: this
                         });
        Tweener.addTween(this.trayBox,
                         { anchor_y: 0,
                           time: immediate ? 0 : KEYBOARD_ANIMATION_TIME,
                           transition: 'easeOutQuad'
                         });
    },

    _hideKeyboardComplete: function() {
        this._chrome.updateRegions();
    },

    // addChrome:
    // @actor: an actor to add to the chrome
    // @params: (optional) additional params
    //
    // Adds @actor to the chrome, and (unless %affectsInputRegion in
    // @params is %false) extends the input region to include it.
    // Changes in @actor's size, position, and visibility will
    // automatically result in appropriate changes to the input
    // region.
    //
    // If %affectsStruts in @params is %true (and @actor is along a
    // screen edge), then @actor's size and position will also affect
    // the window manager struts. Changes to @actor's visibility will
    // NOT affect whether or not the strut is present, however.
    //
    // If %visibleInFullscreen in @params is %true, the actor will be
    // visible even when a fullscreen window should be covering it.
    addChrome: function(actor, params) {
        this._chrome.addActor(actor, params);
    },

    // trackChrome:
    // @actor: a descendant of the chrome to begin tracking
    // @params: parameters describing how to track @actor
    //
    // Tells the chrome to track @actor, which must be a descendant
    // of an actor added via addChrome(). This can be used to extend the
    // struts or input region to cover specific children.
    //
    // @params can have any of the same values as in addChrome(),
    // though some possibilities don't make sense (eg, trying to have
    // a %visibleInFullscreen child of a non-%visibleInFullscreen
    // parent). By default, @actor has the same params as its chrome
    // ancestor.
    trackChrome: function(actor, params) {
        this._chrome.trackActor(actor, params);
    },

    // untrackChrome:
    // @actor: an actor previously tracked via trackChrome()
    //
    // Undoes the effect of trackChrome()
    untrackChrome: function(actor) {
        this._chrome.untrackActor(actor);
    },

    // removeChrome:
    // @actor: a chrome actor
    //
    // Removes @actor from the chrome
    removeChrome: function(actor) {
        this._chrome.removeActor(actor);
    },

    findMonitorForActor: function(actor) {
        return this._chrome.findMonitorForActor(actor);
    }
};
Signals.addSignalMethods(LayoutManager.prototype);


// HotCorner:
//
// This class manages a "hot corner" that can toggle switching to
// overview.
function HotCorner() {
    this._init();
}
```

```
HotCorner.prototype = {
    _init : function() {
        // We use this flag to mark the case where the user has entered the
        // hot corner and has not left both the hot corner and a surrounding
        // guard area (the "environs"). This avoids triggering the hot corner
        // multiple times due to an accidental jitter.
        this._entered = false;

        this.actor = new Clutter.Group({ name: 'hot-corner-environs',
                                         width: 3,
                                         height: 3,
                                         reactive: true });

        this._corner = new Clutter.Rectangle({ name: 'hot-corner',
                                               width: 1,
                                               height: 1,
                                               opacity: 0,
                                               reactive: true });
        this._corner._delegate = this;

        this.actor.add_actor(this._corner);

        if (St.Widget.get_default_direction() == St.TextDirection.RTL) {
            this._corner.set_position(this.actor.width - this._corner.width, 0);
            this.actor.set_anchor_point_from_gravity(Clutter.Gravity.NORTH_EAST);
        } else {
            this._corner.set_position(0, 0);
        }

        this._activationTime = 0;

        this.actor.connect('leave-event',
                           Lang.bind(this, this._onEnvironsLeft));

        // Clicking on the hot corner environs should result in the
        // same behavior as clicking on the hot corner.
        this.actor.connect('button-release-event',
                           Lang.bind(this, this._onCornerClicked));

        // In addition to being triggered by the mouse enter event,
        // the hot corner can be triggered by clicking on it. This is
        // useful if the user wants to undo the effect of triggering
        // the hot corner once in the hot corner.
        this._corner.connect('enter-event',
                             Lang.bind(this, this._onCornerEntered));
        this._corner.connect('button-release-event',
                             Lang.bind(this, this._onCornerClicked));
        this._corner.connect('leave-event',
                             Lang.bind(this, this._onCornerLeft));

        // Cache the three ripples instead of dynamically creating
        and destroying them.
        this._ripple1 = new St.BoxLayout
        ({ style_class: 'ripple-box', opacity: 0 });
        this._ripple2 = new St.BoxLayout
        ({ style_class: 'ripple-box', opacity: 0 });
        this._ripple3 = new St.BoxLayout
        ({ style_class: 'ripple-box', opacity: 0 });

        Main.uiGroup.add_actor(this._ripple1);
        Main.uiGroup.add_actor(this._ripple2);
        Main.uiGroup.add_actor(this._ripple3);
    },

    destroy: function() {
        this.actor.destroy();
    },

    _animRipple : function(ripple, delay, time, startScale, startOpacity,
    finalScale) {
        // We draw a ripple by using a source image and animating it scaling
        // outwards and fading away. We want the ripples to move linearly
        // or it looks unrealistic, but if the opacity of the ripple goes
        // linearly to zero it fades away too quickly, so we use Tweener's
        // 'onUpdate' to give a non-linear curve to the fade-away and make
        // it more visible in the middle section.

        ripple._opacity = startOpacity;

        if (ripple.get_direction() == St.TextDirection.RTL)
            ripple.set_anchor_point_from_gravity(Clutter.Gravity.NORTH_EAST);

        ripple.visible = true;
        ripple.opacity = 255 * Math.sqrt(startOpacity);
        ripple.scale_x = ripple.scale_y = startScale;
        let [x, y] = this._corner.get_transformed_position();
        ripple.x = x;
        ripple.y = y;

        Tweener.addTween(ripple, { _opacity: 0,
                                   scale_x: finalScale,
                                   scale_y: finalScale,
                                   delay: delay,
                                   time: time,
                                   transition: 'linear',
                                   onUpdate: function() { ripple.opacity =
                                   255 * Math.sqrt(ripple._opacity); },
                                   onComplete: function() { ripple.visible =
                                   false; } });
    },

    rippleAnimation: function() {
        // Show three concentric ripples expanding outwards; the exact
        // parameters were found by trial and error, so don't look
        // for them to make perfect sense mathematically

        //                              delay  time  scale opacity => scale
        this._animRipple(this._ripple1, 0.0,   0.83,  0.25,  1.0,     1.5);
        this._animRipple(this._ripple2, 0.05,  1.0,   0.0,   0.7,     1.25);
        this._animRipple(this._ripple3, 0.35,  1.0,   0.0,   0.3,     1);
    },

    handleDragOver: function(source, actor, x, y, time) {
        if (source != Main.xdndHandler)
            return;

        if (!Main.overview.visible && !Main.overview.animationInProgress) {
            this.rippleAnimation();
            Main.overview.showTemporarily();
            Main.overview.beginItemDrag(actor);
        }
    },

    _onCornerEntered : function() {
        if (!this._entered) {
            this._entered = true;
            if (!Main.overview.animationInProgress) {
                this._activationTime = Date.now() / 1000;

                this.rippleAnimation();
                Main.overview.toggle();
            }
        }
        return false;
    },

    _onCornerClicked : function() {
        if (this.shouldToggleOverviewOnClick())
            Main.overview.toggle();
        return true;
    },

    _onCornerLeft : function(actor, event) {
        if (event.get_related() != this.actor)
            this._entered = false;
        // Consume event, otherwise this will confuse onEnvironsLeft
        return true;
    },

    _onEnvironsLeft : function(actor, event) {
        if (event.get_related() != this._corner)
            this._entered = false;
        return false;
    },

    // Checks if the Activities button is currently sensitive to
    // clicks. The first call to this function within the
    // HOT_CORNER_ACTIVATION_TIMEOUT time of the hot corner being
    // triggered will return false. This avoids opening and closing
    // the overview if the user both triggered the hot corner and
    // clicked the Activities button.
    shouldToggleOverviewOnClick: function() {
        if (Main.overview.animationInProgress)
            return false;
        if (this._activationTime == 0 || Date.now() / 1000 -
        this._activationTime > HOT_CORNER_ACTIVATION_TIMEOUT)
            return true;
        return false;
    }
};
```

```
// This manages the shell "chrome"; the UI that's visible in the
// normal mode (ie, outside the Overview), that surrounds the main
// workspace content.

const defaultParams = {
    visibleInFullscreen: false,
    affectsStruts: false,
    affectsInputRegion: true
};

function Chrome() {
    this._init.apply(this, arguments);
}

Chrome.prototype = {
    _init: function(layoutManager) {
        this._layoutManager = layoutManager;

        this._monitors = [];
        this._inOverview = true;
        this._updateRegionIdle = 0;
        this._freezeUpdateCount = 0;

        this._trackedActors = [];

        this._layoutManager.connect('monitors-changed',
                                    Lang.bind(this, this._relayout));
        global.screen.connect('restacked',
                              Lang.bind(this, this._windowsRestacked));

        // Need to update struts on new workspaces when they are added
        global.screen.connect('notify::n-workspaces',
                              Lang.bind(this, this._queueUpdateRegions));

        this._screenSaverActive = false;
        this._screenSaverProxy = new ScreenSaver.ScreenSaverProxy();
        this._screenSaverProxy.connect('ActiveChanged', Lang.bind(this,
        this._onScreenSaverActiveChanged));
        this._screenSaverProxy.GetActiveRemote(Lang.bind(this,
            function(result, err) {
                if (!err)
                this._onScreenSaverActiveChanged
                (this._screenSaverProxy, result);
        }));

    this._relayout();
},

init: function() {
    Main.overview.connect('showing',
                          Lang.bind(this, this._overviewShowing));
    Main.overview.connect('hidden',
                          Lang.bind(this, this._overviewHidden));
},

addActor: function(actor, params) {
    Main.uiGroup.add_actor(actor);
    this._trackActor(actor, params);
},

trackActor: function(actor, params) {
    let ancestor = actor.get_parent();
    let index = this._findActor(ancestor);
    while (ancestor && index == -1) {
        ancestor = ancestor.get_parent();
        index = this._findActor(ancestor);
    }
    if (!ancestor)
        throw new Error('actor is not a descendent of
        a chrome actor');

    let ancestorData = this._trackedActors[index];
    if (!params)
        params = {};
    // We can't use Params.parse here because we want to drop
    // the extra values like ancestorData.actor
    for (let prop in defaultParams) {
        if (!params.hasOwnProperty(prop))
            params[prop] = ancestorData[prop];
    }

    this._trackActor(actor, params);
},
    untrackActor: function(actor) {
        this._untrackActor(actor);
    },

    removeActor: function(actor) {
        Main.uiGroup.remove_actor(actor);
        this._untrackActor(actor);
    },

    _findActor: function(actor) {
        for (let i = 0; i < this._trackedActors.length; i++) {
            let actorData = this._trackedActors[i];
            if (actorData.actor == actor)
                return i;
        }
        return -1;
    },

    _trackActor: function(actor, params) {
        if (this._findActor(actor) != -1)
            throw new Error('trying to re-track existing chrome actor');

        let actorData = Params.parse(params, defaultParams);
        actorData.actor = actor;
        actorData.isToplevel = actor.get_parent() == Main.uiGroup;
        actorData.visibleId = actor.connect('notify::visible',
                                            Lang.bind(this,
                                            this._queueUpdateRegions));
        actorData.allocationId = actor.connect('notify::allocation',
                                               Lang.bind(this,
                                               this._queueUpdateRegions));
        actorData.parentSetId = actor.connect('parent-set',
                                              Lang.bind(this,
                                              this._actorReparented));
        // Note that destroying actor will unset its parent, so we don't
        // need to connect to 'destroy' too.

        this._trackedActors.push(actorData);
        this._queueUpdateRegions();
    },

    _untrackActor: function(actor) {
        let i = this._findActor(actor);
        if (i == -1)
            return;
        let actorData = this._trackedActors[i];

        this._trackedActors.splice(i, 1);
        actor.disconnect(actorData.visibleId);
        actor.disconnect(actorData.allocationId);
        actor.disconnect(actorData.parentSetId);

        this._queueUpdateRegions();
    },

    _actorReparented: function(actor, oldParent) {
        let newParent = actor.get_parent();
        if (!newParent)
            this._untrackActor(actor);
        else
            actorData.isToplevel = (newParent == Main.uiGroup);
    },

    _updateVisibility: function() {
        for (let i = 0; i < this._trackedActors.length; i++) {
            let actorData = this._trackedActors[i], visible;
            if (!actorData.isToplevel)
                continue;

            if (this._screenSaverActive)
                visible = false;
            else if (this._inOverview)
                visible = true;
            else if (!actorData.visibleInFullscreen &&
                   this.findMonitorForActor(actorData.actor).inFullscreen)
                visible = false;
            else
                 visible = true;
            Main.uiGroup.set_skip_paint(actorData.actor, !visible);
        }
    },

    _overviewShowing: function() {
        this._inOverview = false;
        this._updateVisibility();
        this._queueUpdateRegions();
    },

    _overviewHidden: function() {
        this._inOverview = false;
        this._updateVisibility();
        this._queueUpdateRegions();
    },

    _relayout: function() {
        this._monitors = this._layoutManager.monitors;
        this._primaryMonitor = this._layoutManager.primaryMonitor;

        this._updateFullscreen();
        this._updateVisibility();
        this._queueUpdateRegions();
    },

    _onScreenSaverActiveChanged: function(proxy, screenSaverActive) {
        this._screenSaverActive = screenSaverActive;
        this._updateVisibility();
        this._queueUpdateRegions();
    },

    _findMonitorForRect: function(x, y, w, h) {
        // First look at what monitor the center of the rectangle is at
        let cx = x + w/2;
        let cy = y + h/2;
        for (let i = 0; i < this._monitors.length; i++) {
            let monitor = this._monitors[i];
            if (cx >= monitor.x && cx < monitor.x + monitor.width &&
                cy >= monitor.y && cy < monitor.y + monitor.height)
                return monitor;
        }
        // If the center is not on a monitor, return the first
        // overlapping monitor
        for (let i = 0; i < this._monitors.length; i++) {
            let monitor = this._monitors[i];
            if (x + w > monitor.x && x < monitor.x + monitor.width &&
                y + h > monitor.y && y < monitor.y + monitor.height)
                return monitor;
        }
        // otherwise on no monitor
        return null;
    },

    _findMonitorForWindow: function(window) {
        return this._findMonitorForRect(window.x, window.y,
        window.width, window.height);
    },

    // This call guarantees that we return some monitor to
    // simplify usage of it
    // In practice all tracked actors should be visible on
    // some monitor anyway
    findMonitorForActor: function(actor) {
        let [x, y] = actor.get_transformed_position();
        let [w, h] = actor.get_transformed_size();
        let monitor = this._findMonitorForRect(x, y, w, h);
        if (monitor)
            return monitor;
        return this._primaryMonitor; // Not on any monitor,
        // pretend its on the primary
    },

    _queueUpdateRegions: function() {
        if (!this._updateRegionIdle && !this._freezeUpdateCount)
            this._updateRegionIdle = Mainloop.idle_add
            (Lang.bind(this, this.updateRegions),
            Meta.PRIORITY_BEFORE_REDRAW);
    },

    freezeUpdateRegions: function() {
        if (this._updateRegionIdle)
            this.updateRegions();
        this._freezeUpdateCount++;
    },

    thawUpdateRegions: function() {
        this._freezeUpdateCount--;
        this._queueUpdateRegions();
    },

    _updateFullscreen: function() {
        let windows = Main.getWindowActorsForWorkspace
        (global.screen.get_active_workspace_index());
        // Reset all monitors to not fullscreen
        for (let i = 0; i < this._monitors.length; i++)
            this._monitors[i].inFullscreen = false;

        // Ordinary chrome should be visible unless there is a window
        // with layer FULLSCREEN, or a window with layer
        // OVERRIDE_REDIRECT that covers the whole screen.
        // ('override_redirect' is not actually a layer above all
        // other windows, but this seems to be how mutter treats it
        // currently...) If we wanted to be extra clever, we could
        // figure out when an OVERRIDE_REDIRECT window was trying to
        // partially overlap us, and then adjust the input region and
        // our clip region accordingly...

        // @windows is sorted bottom to top.

        for (let i = windows.length - 1; i > -1; i--) {
            let window = windows[i];
            let layer = window.get_meta_window().get_layer();

            // Skip minimized windows
            if (!window.showing_on_its_workspace())
                continue;

            if (layer == Meta.StackLayer.FULLSCREEN) {
                let monitor = this._findMonitorForWindow(window);
                if (monitor)
                    monitor.inFullscreen = true;
            }
            if (layer == Meta.StackLayer.OVERRIDE_REDIRECT) {
                // Check whether the window is screen sized
                let isScreenSized =
                    (window.x == 0 && window.y == 0 &&
                    window.width == global.screen_width &&
                    window.height == global.screen_height);

                if (isScreenSized) {
                    for (let i = 0; i < this._monitors.length; i++)
                        this._monitors[i].inFullscreen = true;
                }

                // Or whether it is monitor sized
                let monitor = this._findMonitorForWindow(window);
                if (monitor &&
                    window.x <= monitor.x &&
                    window.x + window.width >= monitor.x + monitor.width &&
                    window.y <= monitor.y &&
                    window.y + window.height >= monitor.y + monitor.height)
                    monitor.inFullscreen = true;
            } else
                break;
        }
    },

    _windowsRestacked: function() {
        let wasInFullscreen = [];
        for (let i = 0; i < this._monitors.length; i++)
            wasInFullscreen[i] = this._monitors[i].inFullscreen;

        this._updateFullscreen();

        let changed = false;
        for (let i = 0; i < wasInFullscreen.length; i++) {
            if (wasInFullscreen[i] != this._monitors[i].inFullscreen) {
                changed = true;
                break;
            }
        }
        if (changed) {
            this._updateVisibility();
            this._queueUpdateRegions();
        }
    },

    updateRegions: function() {
        let rects = [], struts = [], i;

        if (this._updateRegionIdle) {
            Mainloop.source_remove(this._updateRegionIdle);
            delete this._updateRegionIdle;
        }

        for (i = 0; i < this._trackedActors.length; i++) {
            let actorData = this._trackedActors[i];
            if (!actorData.affectsInputRegion && !actorData.affectsStruts)
            continue;

        let [x, y] = actorData.actor.get_transformed_position();
        let [w, h] = actorData.actor.get_transformed_size();
        x = Math.round(x);
        y = Math.round(y);
        w = Math.round(w);
        h = Math.round(h);
        let rect = new Meta.Rectangle({ x: x, y: y, width: w, height: h});

        if (actorData.affectsInputRegion &&
            actorData.actor.get_paint_visibility() &&
            !Main.uiGroup.get_skip_paint(actorData.actor))
            rects.push(rect);

        if (!actorData.affectsStruts)
            continue;

        // Limit struts to the size of the screen
        let x1 = Math.max(x, 0);
        let x2 = Math.min(x + w, global.screen_width);
        let y1 = Math.max(y, 0);
        let y2 = Math.min(y + h, global.screen_height);

        // NetWM struts are not really powerful enought to handle
        // a multi-monitor scenario, they only describe what happens
        // around the outer sides of the full display region. However
        // it can describe a partial region along each side, so
        // we can support having the struts only affect the
        // primary monitor. This should be enough as we only have
        // chrome affecting the struts on the primary monitor so
        // far.
        //
        // Metacity wants to know what side of the screen the
        // strut is considered to be attached to. If the actor is
        // only touching one edge, or is touching the entire
        // border of the primary monitor, then it's obvious which
        // side to call it. If it's in a corner, we pick a side
        // arbitrarily. If it doesn't touch any edges, or it spans
        // the width/height across the middle of the screen, then
        // we don't create a strut for it at all.
        let side;
        let primary = this._primaryMonitor;
```

```
if (x1 <= primary.x && x2 >= primary.x + primary.width) {
    if (y1 <= primary.y)
        side = Meta.Side.TOP;
    else if (y2 >= primary.y + primary.height)
        side = Meta.Side.BOTTOM;
    else
        continue;
} else if (y1 <= primary.y && y2 >= primary.y + primary.height) {
    if (x1 <= 0)
        side = Meta.Side.LEFT;
    else if (x2 >= global.screen_width)
        side = Meta.Side.RIGHT;
    else
        continue;
} else if (x1 <= 0)
    side = Meta.Side.LEFT;
else if (y1 <= 0)
    side = Meta.Side.TOP;
else if (x2 >= global.screen_width)
    side = Meta.Side.RIGHT;
else if (y2 >= global.screen_height)
    side = Meta.Side.BOTTOM;
else
    continue;

// Ensure that the strut rects goes all the way to the screen edge,
// as this really what mutter expects.
switch (side) {
case Meta.Side.TOP:
    y1 = 0;
    break;
case Meta.Side.BOTTOM:
    y2 = global.screen_height;
    break;
case Meta.Side.LEFT:
    x1 = 0;
    break;
case Meta.Side.RIGHT:
    x2 = global.screen_width;
    break;
}

let strutRect = new Meta.Rectangle({ x: x1, y: y1,
            width: x2 - x1, height: y2 - y1});
            let strut = new Meta.Strut({ rect: strutRect, side: side });
            struts.push(strut);
        }

        global.set_stage_input_region(rects);

        let screen = global.screen;
        for (let w = 0; w < screen.n_workspaces; w++) {
            let workspace = screen.get_workspace_by_index(w);
            workspace.set_builtin_struts(struts);
        }

        return false;
    }
};
```

/usr/share/gnome-shell/js/ui/panel.js

```
// -*- mode: js; js-indent-level: 4; indent-tabs-mode: nil -*-
const AccountsService = imports.gi.AccountsService;
const Cairo = imports.cairo;
const Clutter = imports.gi.Clutter;
const Gio = imports.gi.Gio;
const DBus = imports.dbus;
const Lang = imports.lang;
const Mainloop = imports.mainloop;
const Pango = imports.gi.Pango;
const Shell = imports.gi.Shell;
const St = imports.gi.St;
const Signals = imports.signals;
const GLib = imports.gi.GLib;

const Files = imports.ui.filesMenu;
const Config = imports.misc.config;
const CtrlAltTab = imports.ui.ctrlAltTab;
const Layout = imports.ui.layout;
const Overview = imports.ui.overview;
const PopupMenu = imports.ui.popupMenu;
const PanelMenu = imports.ui.panelMenu;
const DateMenu = imports.ui.dateMenu;
const Main = imports.ui.main;
const MessageTray = imports.ui.messageTray;
const Tweener = imports.ui.tweener;
const UserMenuButton = imports.ui.userMenu.UserMenuButton;
const userMenu = imports.ui.userMenu;
const StatusIconDispatcher = imports.ui.statusIconDispatcher;
const PANEL_ICON_SIZE = 24;
const PlaceDisplay = imports.ui.placeDisplay;

const BUTTON_DND_ACTIVATION_TIMEOUT = 250;

const ANIMATED_ICON_UPDATE_TIMEOUT = 100;
const SPINNER_ANIMATION_TIME = 0.2;
const STANDARD_STATUS_AREA_ORDER = ['a11y', 'keyboard', 'volume',
'bluetooth', 'network', 'battery'];
const STANDARD_STATUS_AREA_SHELL_IMPLEMENTATION = {
    'a11y': imports.ui.status.accessibility.ATIndicator,
    'volume': imports.ui.status.volume.Indicator,
    'battery': imports.ui.status.power.Indicator,
    'keyboard': imports.ui.status.keyboard.XKBIndicator
};
if (Config.HAVE_BLUETOOTH)
    STANDARD_STATUS_AREA_SHELL_IMPLEMENTATION['bluetooth'] =
    imports.ui.status.bluetooth.Indicator;

try {
    STANDARD_STATUS_AREA_SHELL_IMPLEMENTATION['network'] =
    imports.ui.status.network.NMApplet;
} catch(e) {
    log('NMApplet is not supported. It is possible that your
    NetworkManager version is too old');
}

const GDM_STATUS_AREA_ORDER = ['a11y', 'display', 'keyboard',
'volume', 'battery', 'powerMenu'];
const GDM_STATUS_AREA_SHELL_IMPLEMENTATION = {
    'a11y': imports.ui.status.accessibility.ATIndicator,
    'volume': imports.ui.status.volume.Indicator,
    'battery': imports.ui.status.power.Indicator,
    'keyboard': imports.ui.status.keyboard.XKBIndicator,
    'powerMenu': imports.gdm.powerMenu.PowerMenuButton
};
let placesManager = new PlaceDisplay.PlacesManager();
// To make sure the panel corners blend nicely with the panel,
// we draw background and borders the same way, e.g. drawing
// them as filled shapes from the outside inwards instead of
// using cairo stroke(). So in order to give the border the
// appearance of being drawn on top of the background, we need
// to blend border and background color together.
// For that purpose we use the following helper methods, taken
// from st-theme-node-drawing.c
function _norm(x) {
    return Math.round(x / 255);
}

function _over(srcColor, dstColor) {
    let src = _premultiply(srcColor);
    let dst = _premultiply(dstColor);
    let result = new Clutter.Color();

    result.alpha = src.alpha + _norm((255 - src.alpha) * dst.alpha);
    result.red = src.red + _norm((255 - src.alpha) * dst.red);
    result.green = src.green + _norm((255 - src.alpha) * dst.green);
    result.blue = src.blue + _norm((255 - src.alpha) * dst.blue);

    return _unpremultiply(result);
}

function _premultiply(color) {
    return new Clutter.Color({ red: _norm(color.red * color.alpha),
                               green: _norm(color.green * color.alpha),
                               blue: _norm(color.blue * color.alpha),
                               alpha: color.alpha });
};

function _unpremultiply(color) {
    if (color.alpha == 0)
        return new Clutter.Color();

    let red = Math.min((color.red * 255 + 127) / color.alpha, 255);
    let green = Math.min((color.green * 255 + 127) / color.alpha, 255);
    let blue = Math.min((color.blue * 255 + 127) / color.alpha, 255);
    return new Clutter.Color({ red: red, green: green,
                               blue: blue, alpha: color.alpha });
};
```

```
function AnimatedIcon(name, size) {
    this._init(name, size);
}

AnimatedIcon.prototype = {
    _init: function(name, size) {
        this.actor = new St.Bin({ visible: false });
        this.actor.connect('destroy', Lang.bind(this, this._onDestroy));
        this.actor.connect('notify::visible', Lang.bind(this, function() {
            if (this.actor.visible) {
                this._timeoutId = Mainloop.timeout_add
                (ANIMATED_ICON_UPDATE_TIMEOUT, Lang.bind(this, this._update));
            } else {
                if (this._timeoutId)
                    Mainloop.source_remove(this._timeoutId);
                this._timeoutId = 0;
            }
        }));

        this._timeoutId = 0;
        this._i = 0;
        this._animations = St.TextureCache.get_default().load_sliced_image
        (global.datadir + '/theme/' + name, size, size);
        this.actor.set_child(this._animations);
    },

    _update: function() {
        this._animations.hide_all();
        this._animations.show();
        if (this._i && this._i < this._animations.get_n_children())
            this._animations.get_nth_child(this._i++).show();
        else {
            this._i = 1;
            if (this._animations.get_n_children())
                this._animations.get_nth_child(0).show();
        }
        return true;
    },

    _onDestroy: function() {
        if (this._timeoutId)
            Mainloop.source_remove(this._timeoutId);
    }
```

```
};

function TextShadower() {
    this._init();
}

TextShadower.prototype = {
    _init: function() {
        this.actor = new Shell.GenericContainer();
        this.actor.connect('get-preferred-width', Lang.bind(this,
        this._getPreferredWidth));
        this.actor.connect('get-preferred-height', Lang.bind(this,
        this._getPreferredHeight));
        this.actor.connect('allocate', Lang.bind(this, this._allocate));

        this._label = new St.Label();
        this.actor.add_actor(this._label);
        for (let i = 0; i < 4; i++) {
            let actor = new St.Label({ style_class: 'label-shadow' });
            actor.clutter_text.ellipsize = Pango.EllipsizeMode.END;
            this.actor.add_actor(actor);
        }
        this._label.raise_top();
    },

    setText: function(text) {
        let children = this.actor.get_children();
        for (let i = 0; i < children.length; i++)
            children[i].set_text(text);
    },

    _getPreferredWidth: function(actor, forHeight, alloc) {
        let [minWidth, natWidth] = this._label.get_preferred_width(forHeight);
        alloc.min_size = minWidth + 2;
        alloc.natural_size = natWidth + 2;
    },

    _getPreferredHeight: function(actor, forWidth, alloc) {
        let [minHeight, natHeight] = this._label.get_preferred_height(forWidth);
        alloc.min_size = minHeight + 2;
        alloc.natural_size = natHeight + 2;
    },
    _allocate: function(actor, box, flags) {
        let children = this.actor.get_children();

        let availWidth = box.x2 - box.x1;
        let availHeight = box.y2 - box.y1;

        let [minChildWidth, minChildHeight, natChildWidth, natChildHeight] =
            this._label.get_preferred_size();

        let childWidth = Math.min(natChildWidth, availWidth - 2);
        let childHeight = Math.min(natChildHeight, availHeight - 2);

        for (let i = 0; i < children.length; i++) {
            let child = children[i];
            let childBox = new Clutter.ActorBox();
            // The order of the labels here is arbitrary, except
            // we know the "real" label is at the end because Clutter.Group
            // sorts by Z order
            switch (i) {
                case 0: // top
                    childBox.x1 = 1;
                    childBox.y1 = 0;
                    break;
                case 1: // right
                    childBox.x1 = 2;
                    childBox.y1 = 1;
                    break;
                case 2: // bottom
                    childBox.x1 = 1;
                    childBox.y1 = 2;
                    break;
                case 3: // left
                    childBox.x1 = 0;
                    childBox.y1 = 1;
                    break;
                case 4: // center
                    childBox.x1 = 1;
                    childBox.y1 = 1;
                    break;
            }
            childBox.x2 = childBox.x1 + childWidth;
            childBox.y2 = childBox.y1 + childHeight;
            child.allocate(childBox, flags);
        }
    }
};

//Places button
//Resource: http://blog.fpmurphy.com/2011/05/more-Gnome-
//shell-customization.html 5 May 2012, 23:25

function Places() {
        this._init();
}
        Places.prototype = {
        __proto__:PanelMenu.Button.prototype,
               _init: function() {
                PanelMenu.Button.prototype._init.call(this, 0.0);
                let containerPlaces = new Shell.GenericContainer();
                containerPlaces.connect('get-preferred-width',
                Lang.bind(this, this._containerGetPreferredWidth));
                containerPlaces.connect('get-preferred-height',
                Lang.bind(this, this._containerGetPreferredHeight));
                containerPlaces.connect('allocate',
                Lang.bind(this, this._containerAllocate));
                this.actor.add_actor(containerPlaces);
                this.actor.name = 'panelActivities';
                this._label = new St.Label({ text: _("Files") });
                containerPlaces.add_actor(this._label);
                let placeid;
        this.placeItems = [];

        this.defaultPlaces = placesManager.getDefaultPlaces();
        this.bookmarks     = placesManager.getBookmarks();
        this.mounts        = placesManager.getMounts();

        for ( placeid = 0; placeid < this.defaultPlaces.length; placeid++) {
            this.placeItems[placeid] = new PopupMenu.PopupMenuItem
            (_(this.defaultPlaces[placeid].name));
            this.placeItems[placeid].place = this.defaultPlaces[placeid];
            this.menu.addMenuItem(this.placeItems[placeid]);
            this.placeItems[placeid].connect('activate', function(actor,event) {
                actor.place.launch();
            });

        }
        this.menu.addMenuItem(new PopupMenu.PopupSeparatorMenuItem());
        for ( let bookmarkid = 0; bookmarkid < this.bookmarks.length;
        bookmarkid++, placeid++) {
            this.placeItems[placeid] = new PopupMenu.PopupMenuItem
            (_(this.bookmarks[bookmarkid].name));
            this.placeItems[placeid].place = this.bookmarks[bookmarkid];
            this.menu.addMenuItem(this.placeItems[placeid]);
            this.placeItems[placeid].connect('activate',
            function(actor,event) {
                actor.place.launch();
            });
        }

        if (this.mounts.length > 0) {
            this.menu.addMenuItem(new PopupMenu.PopupSeparatorMenuItem());
        }
        for ( let mountid = 0; mountid < this.mounts.length;
        placeid++, mountid++ ) {
            this.placeItems[placeid] = new PopupMenu.PopupMenuItem
            (_(this.mounts[mountid].name));
            this.placeItems[placeid].place = this.mounts[mountid];
            this.menu.addMenuItem(this.placeItems[placeid]);
            this.placeItems[placeid].connect('activate', function(actor,event) {
                actor.place.launch();
            });
        }
        },

    _containerGetPreferredWidth: function(actor, forHeight, alloc) {
        [alloc.min_size, alloc.natural_size] =
        this._label.get_preferred_width(forHeight);
    },

    _containerGetPreferredHeight: function(actor, forWidth, alloc) {
        [alloc.min_size, alloc.natural_size] =
        this._label.get_preferred_height(forWidth);
    },

    _containerAllocate: function(actor, box, flags) {
        this._label.allocate(box, flags);
    },
};
```

```
// Activities button. Because everything else in the top bar is a
// PanelMenu.Button, it simplifies some things to make this be one too.
// We just hack it up to not actually have a menu attached to it.
function ActivitiesButton() {
    this._init.apply(this, arguments);
}

ActivitiesButton.prototype = {
    __proto__: PanelMenu.Button.prototype,

    _init: function() {
        PanelMenu.Button.prototype._init.call(this, 0.0);

        let container = new Shell.GenericContainer();
        container.connect('get-preferred-width',
        Lang.bind(this, this._containerGetPreferredWidth));
        container.connect('get-preferred-height',
        Lang.bind(this, this._containerGetPreferredHeight));
        container.connect('allocate',
        Lang.bind(this, this._containerAllocate));
        this.actor.add_actor(container);
        this.actor.name = 'panelActivities';

        /* Translators: If there is no suitable word for "Activities"
           in your language, you can use the word for "Overview". */
        this._label = new St.Label({ text: _("Activities") });
        container.add_actor(this._label);
        // Hack up our menu...
        this.menu.open = Lang.bind(this, this._onMenuOpenRequest);
        this.menu.close = Lang.bind(this, this._onMenuCloseRequest);
        this.menu.toggle = Lang.bind(this, this._onMenuToggleRequest);

        this.actor.connect('captured-event',
         Lang.bind(this, this._onCapturedEvent));
        this.actor.connect_after('button-release-event',
        Lang.bind(this, this._onButtonRelease));
        this.actor.connect_after('key-release-event',
        Lang.bind(this, this._onKeyRelease));

        Main.overview.connect('showing', Lang.bind(this, function() {
            this.actor.add_style_pseudo_class('overview');
            this._escapeMenuGrab();
        }));
        Main.overview.connect('hiding', Lang.bind(this, function() {
            this.actor.remove_style_pseudo_class('overview');
            this._escapeMenuGrab();
        }));

        this._xdndTimeOut = 0;
    },

    _containerGetPreferredWidth: function(actor, forHeight, alloc) {
        [alloc.min_size, alloc.natural_size] =
        this._label.get_preferred_width(forHeight);
    },

    _containerGetPreferredHeight: function(actor, forWidth, alloc) {
        [alloc.min_size, alloc.natural_size] =
        this._label.get_preferred_height(forWidth);
    },

    _containerAllocate: function(actor, box, flags) {
        this._label.allocate(box, flags);
    },

       handleDragOver: function(source, actor, x, y, time) {
        if (source != Main.xdndHandler)
            return;

        if (this._xdndTimeOut != 0)
            Mainloop.source_remove(this._xdndTimeOut);
        this._xdndTimeOut = Mainloop.timeout_add
        (BUTTON_DND_ACTIVATION_TIMEOUT,
        Lang.bind(this, this._xdndShowOverview, actor));
    },

    _escapeMenuGrab: function() {
        if (this.menu.isOpen)
            this.menu.close();
    },

    _onCapturedEvent: function(actor, event) {
        if (event.type() == Clutter.EventType.BUTTON_PRESS) {
        }
        return false;
    },
    _onMenuOpenRequest: function() {
        this.menu.isOpen = true;
        this.menu.emit('open-state-changed', true);
    },

    _onMenuCloseRequest: function() {
        this.menu.isOpen = false;
        this.menu.emit('open-state-changed', false);
    },

    _onMenuToggleRequest: function() {
        this.menu.isOpen = !this.menu.isOpen;
        this.menu.emit('open-state-changed', this.menu.isOpen);
    },

    _onButtonRelease: function() {
        if (this.menu.isOpen) {
            this.menu.close();
            Main.overview.toggle();
        }
    },

    _onKeyRelease: function(actor, event) {
        let symbol = event.get_key_symbol();
        if (symbol == Clutter.KEY_Return ||
        symbol == Clutter.KEY_space) {
            if (this.menu.isOpen)
                this.menu.close();
            Main.overview.toggle();
        }
    },

    _xdndShowOverview: function(actor) {
        let [x, y, mask] = global.get_pointer();
        let pickedActor = global.stage.get_actor_at_pos
        (Clutter.PickMode.REACTIVE, x, y);

        if (pickedActor == this.actor) {
            if (!Main.overview.visible &&
            !Main.overview.animationInProgress) {
                Main.overview.showTemporarily();
                Main.overview.beginItemDrag(actor);
            }
        }

        Mainloop.source_remove(this._xdndTimeOut);
        this._xdndTimeOut = 0;
    }
};

function PanelCorner(panel, side) {
    this._init(panel, side);
}

PanelCorner.prototype   = {
    _init: function(box, side) {
        this._side = side;

        this._box = box;
        this._box.connect('style-changed',
        Lang.bind(this, this._boxStyleChanged));

        this.actor = new St.DrawingArea
        ({ style_class: 'panel-corner' });
        this.actor.connect('style-changed',
        Lang.bind(this, this._styleChanged));
        this.actor.connect('repaint',
        Lang.bind(this, this._repaint));
    },

    _findRightmostButton: function(container) {
        if (!container.get_children)
            return null;

        let children = container.get_children();

        if (!children || children.length == 0)
            return null;

        // Start at the back and work backward
        let index = children.length - 1;
        while (!children[index].visible && index >= 0)
            index--;

        if (index < 0)
```

```
            return null;

        if (!(children[index].has_style_class_name('panel-menu')) &&
            !(children[index].has_style_class_name('panel-button')))
            return this._findRightmostButton(children[index]);

        return children[index];
    },

    _findLeftmostButton: function(container) {
        if (!container.get_children)
            return null;

        let children = container.get_children();

        if (!children || children.length == 0)
            return null;

        // Start at the front and work forward
        let index = 0;
        while (!children[index].visible && index < children.length)
            index++;

        if (index == children.length)
            return null;

        if (!(children[index].has_style_class_name('panel-menu')) &&
            !(children[index].has_style_class_name('panel-button')))
            return this._findLeftmostButton(children[index]);

        return children[index];
    },

    _boxStyleChanged: function() {
        let side = this._side;

        let rtlAwareContainer = this._box instanceof St.BoxLayout;
        if (rtlAwareContainer &&
            this._box.get_direction() == St.TextDirection.RTL) {
            if (this._side == St.Side.LEFT)
                side = St.Side.RIGHT;
            else if (this._side == St.Side.RIGHT)
                side = St.Side.LEFT;
        }

        let button;
        if (side == St.Side.LEFT)
            button = this._findLeftmostButton(this._box);
        else if (side == St.Side.RIGHT)
            button = this._findRightmostButton(this._box);

        if (button) {
            if (this._button && this._buttonStyleChangedSignalId) {
                this._button.disconnect(this._buttonStyleChangedSignalId);
                this._button.style = null;
            }

            this._button = button;

            button.connect('destroy', Lang.bind(this,
                function() {
                    if (this._button == button) {
                        this._button = null;
                        this._buttonStyleChangedSignalId = 0;
                    }
                }));

            // Synchronize the locate button's pseudo classes with this corner
            this._buttonStyleChangedSignalId = button.connect('style-changed',
            Lang.bind(this,
                function(actor) {
                    let pseudoClass = button.get_style_pseudo_class();
                    this.actor.set_style_pseudo_class(pseudoClass);
                }));

            // The corner doesn't support theme transitions, so override
            // the .panel-button default
            button.style = 'transition-duration: 0';
        }
    },

    _repaint: function() {
        let node = this.actor.get_theme_node();

        let cornerRadius = node.get_length("-panel-corner-radius");
        let innerBorderWidth = node.get_length('-panel-corner-inner-border-width');
```

```
let outerBorderWidth = node.get_length('-panel-corner-outer-border-width');

let backgroundColor = node.get_color('-panel-corner-background-color');
let innerBorderColor = node.get_color('-panel-corner-inner-border-color');
let outerBorderColor = node.get_color('-panel-corner-outer-border-color');

let cr = this.actor.get_context();
cr.setOperator(Cairo.Operator.SOURCE);

cr.moveTo(0, 0);
if (this._side == St.Side.LEFT)
    cr.arc(cornerRadius,
           innerBorderWidth + cornerRadius,
           cornerRadius, Math.PI, 3 * Math.PI / 2);
else
    cr.arc(0,
           innerBorderWidth + cornerRadius,
           cornerRadius, 3 * Math.PI / 2, 2 * Math.PI);
cr.lineTo(cornerRadius, 0);
cr.closePath();

let savedPath = cr.copyPath();

let over = _over(innerBorderColor,
                 _over(outerBorderColor, backgroundColor));
Clutter.cairo_set_source_color(cr, over);
cr.fill();

let xOffsetDirection = this._side == St.Side.LEFT ? -1 : 1;
let offset = outerBorderWidth;
over = _over(innerBorderColor, backgroundColor);
Clutter.cairo_set_source_color(cr, over);

cr.save();
cr.translate(xOffsetDirection * offset, - offset);
cr.appendPath(savedPath);
cr.fill();
cr.restore();

if (this._side == St.Side.LEFT)
    cr.rectangle(cornerRadius - offset, 0, offset, outerBorderWidth);
else
    cr.rectangle(0, 0, offset, outerBorderWidth);
```

```
        cr.fill();

        offset = innerBorderWidth;
        Clutter.cairo_set_source_color(cr, backgroundColor);

        cr.save();
        cr.translate(xOffsetDirection * offset, - offset);
        cr.appendPath(savedPath);
        cr.fill();
        cr.restore();
    },

    _styleChanged: function() {
        let node = this.actor.get_theme_node();

        let cornerRadius = node.get_length("-panel-corner-radius");
        let innerBorderWidth = node.get_length('-panel-corner-inner-border-width');

        this.actor.set_size(cornerRadius, innerBorderWidth + cornerRadius);
        this.actor.set_anchor_point(0, innerBorderWidth);
    }
};

function Panel() {
    this._init();
}

Panel.prototype = {
    _init : function() {
        this.actor = new Shell.GenericContainer({ name: 'panel',
                                                  reactive: true });
        this.actor._delegate = this;

        this._statusArea = {};
        this._userMenu = {};

        Main.overview.connect('shown', Lang.bind(this, function () {
            this.actor.add_style_class_name('in-overview');
        }));
        Main.overview.connect('hiding', Lang.bind(this, function () {
            this.actor.remove_style_class_name('in-overview');
        }));
```

```
this._menus = new PopupMenu.PopupMenuManager(this);

this._firstTopBox = new St.BoxLayout({ name: 'firstTop' });
this.actor.add_actor(this._firstTopBox);
this._secondTopBox = new St.BoxLayout({ name: 'secondTop' });
this.actor.add_actor(this._secondTopBox);
this._thirdTopBox = new St.BoxLayout({ name: 'thirdTop' });
this.actor.add_actor(this._thirdTopBox);
this._fourthTopBox = new St.BoxLayout({ name: 'fourthTop' });
this.actor.add_actor(this._fourthTopBox);
this._fifthTopBox = new St.BoxLayout({ name: 'fifthTop' });
this.actor.add_actor(this._fifthTopBox);
this._sixthTopBox = new St.BoxLayout({ name: 'sixthTop' });
this.actor.add_actor(this._sixthTopBox);

if (this.actor.get_direction() == St.TextDirection.RTL)
    this._leftCorner = new PanelCorner(this._thirdTopBox, St.Side.LEFT);
else
    this._leftCorner = new PanelCorner(this._firstTopBox, St.Side.LEFT);

this.actor.add_actor(this._leftCorner.actor);

if (this.actor.get_direction() == St.TextDirection.RTL)
    this._rightCorner = new PanelCorner(this._firstTopBox, St.Side.RIGHT);
else
    this._rightCorner = new PanelCorner(this._thirdTopBox, St.Side.RIGHT);
this.actor.add_actor(this._rightCorner.actor);

this.actor.connect('get-preferred-width',
Lang.bind(this, this._getPreferredWidth));
this.actor.connect('get-preferred-height',
Lang.bind(this, this._getPreferredHeight));
this.actor.connect('allocate', Lang.bind(this, this._allocate));

/* Button on top of the panel */
if (global.session_type == Shell.SessionType.USER) {
    this._activitiesButton = new ActivitiesButton();
    this._activities = this._activitiesButton.actor;
    this._firstTopBox.add(this._activities, { x_fill:true });
    this._menus.addMenu(this._activitiesButton.menu);

}
```

```
        /* 2nd top */
        if (global.session_type == Shell.SessionType.USER)
            this._dateMenu = new DateMenu.DateMenuButton({ showEvents: true });
        else
            this._dateMenu = new DateMenu.DateMenuButton({ showEvents: false });
        this._secondTopBox.add(this._dateMenu.actor, { x_fill: true });
        this._menus.addMenu(this._dateMenu.menu);

        /* bottom */
        if (global.session_type == Shell.SessionType.GDM) {
            this._status_area_order = GDM_STATUS_AREA_ORDER;
            this._status_area_shell_implementation =
            GDM_STATUS_AREA_SHELL_IMPLEMENTATION;
        } else {
            this._status_area_order =
            STANDARD_STATUS_AREA_ORDER;
            this._status_area_shell_implementation =
            STANDARD_STATUS_AREA_SHELL_IMPLEMENTATION;
        }
        Main.statusIconDispatcher.connect
        ('status-icon-added', Lang.bind(this, this._onTrayIconAdded));
        Main.statusIconDispatcher.connect
        ('status-icon-removed', Lang.bind(this, this._onTrayIconRemoved));

        Main.layoutManager.panelBox.add(this.actor);
        Main.ctrlAltTabManager.addGroup(this.actor, _("Left Bar"), 'start-here',
                                        { sortGroup: CtrlAltTab.SortGroup.TOP });

        //fourthTopBox
        if (global.session_type == Shell.SessionType.USER) {
                this._userMenu = new UserMenuButton();
                this._user = this._userMenu.actor;
                this._fourthTopBox.add(this._user, { x_fill:true });
                this._menus.addMenu(this._userMenu.menu);
        }
        //fifthTopBox
        if (global.session_type == Shell.SessionType.USER) {
                this._trayButton = new MessageTray.MessageTrayButton();
                this._notify = this._trayButton.actor;
                this._notify.connect('button-press-event',
                Lang.bind(this, function () {
            Main.messageTray.toggle();
        }));
```

```
                this._fifthTopBox.add(this._notify, { x_fill:true });
        }
        //sixthTopBox
      if (global.session_type == Shell.SessionType.USER) {
            this._placesButton = new Places();
            this._placesMenu = this._placesButton.actor;
            this._sixthTopBox.add(this._placesMenu);
            this._menus.addMenu(this._placesButton.menu);

        }
    },

    _getPreferredWidth: function(actor, forHeight, alloc) {
        alloc.min_size = -1;
        alloc.natural_size = Main.layoutManager.primaryMonitor.width*0.12;
    },

    _getPreferredHeight: function(actor, forWidth, alloc) {
        // We don't need to implement this; it's forced by the CSS
        alloc.min_size = -1;
        alloc.natural_size = -1;
    },

    _allocate: function(actor, box, flags) {
        let monitor = Main.layoutManager.primaryMonitor;
        let allocWidth = monitor.width*0.12;
        let allocHeight = monitor.height;
        let availWidth = monitor.width*0.12;

        let [firstTopMinWidth, firstTopNaturalWidth] =
        this._firstTopBox.get_preferred_width(-1);
        let [secondTopMinWidth, secondTopNaturalWidth] =
        this._secondTopBox.get_preferred_width(-1);
        let [thirdTopMinWidth, thirdTopNaturalWidth] =
        this._thirdTopBox.get_preferred_width(-1);
        let [fourthTopMinWidth, fourthTopNaturalWidth] =
        this._fourthTopBox.get_preferred_width(-1);
        let [fifthTopMinWidth, fifthTopNaturalWidth] =
        this._fifthTopBox.get_preferred_width(-1);
        let [sixthTopMinWidth, sixthTopNaturalWidth] =
        this._sixthTopBox.get_preferred_width(-1);

        let sideWidth, centerWidth;
    centerWidth = secondTopNaturalWidth;
    sideWidth = (allocWidth - centerWidth) / 2;

    let childBox = new Clutter.ActorBox();

    childBox.y1 = 0;
    childBox.y2 = 35;
    childBox.x1=0;
    childBox.x2=allocWidth;
    this._firstTopBox.allocate(childBox, flags);
     if (global.session_type == Shell.SessionType.GDM)
           {
    childBox.y1 = 0;
    childBox.y2 = 35;
    childBox.x1=0;
    childBox.x2=allocWidth;
    this._secondTopBox.allocate(childBox, flags);}
    else
    {
    childBox.y1 = 70;
    childBox.y2 = 105;
    childBox.x1=0;
    childBox.x2=allocWidth;
    this._secondTopBox.allocate(childBox, flags);
    }

  if (global.session_type == Shell.SessionType.USER) {
    childBox.y1 = allocHeight-70;
    childBox.y2 = allocHeight -45;
  }
  else {
    childBox.y1 = allocHeight-45;
    childBox.y2 = allocHeight;
  }
    childBox.x1=0;
    childBox.x2=allocWidth;
    this._thirdTopBox.allocate(childBox, flags);

    childBox.y1 = 35;
    childBox.y2 = 70;
    childBox.x1 = 0;
    childBox.x2 = allocWidth;
    this._fourthTopBox.allocate(childBox, flags);
```

```
        childBox.y1 =allocHeight-45;
        childBox.y2 = allocHeight;
        childBox.x1 = 0;
        childBox.x2 = allocWidth;
        this._fifthTopBox.allocate(childBox, flags);

        childBox.y1 = 105;
        childBox.y2 = 140;
        childBox.x1 = 0;
        childBox.x2 = allocWidth;
        this._sixthTopBox.allocate(childBox, flags);

        let [cornerMinWidth, cornerWidth] =
        this._leftCorner.actor.get_preferred_width(-1);
        let [cornerMinHeight, cornerHeight] =
        this._leftCorner.actor.get_preferred_width(-1);
        childBox.x1 = 0;
        childBox.x2 = cornerWidth;
        childBox.y1 = allocHeight;
        childBox.y2 = allocHeight + cornerHeight;
        this._leftCorner.actor.allocate(childBox, flags);

        let [cornerMinWidth, cornerWidth] =
        this._rightCorner.actor.get_preferred_width(-1);
        let [cornerMinHeight, cornerHeight] =
        this._rightCorner.actor.get_preferred_width(-1);
        childBox.x1 = allocWidth - cornerWidth;
        childBox.x2 = allocWidth;
        childBox.y1 = allocHeight;
        childBox.y2 = allocHeight + cornerHeight;
        this._rightCorner.actor.allocate(childBox, flags);
    },

    startStatusArea: function() {
        for (let i = 0; i < this._status_area_order.length; i++) {
            let role = this._status_area_order[i];
            let constructor = this._status_area_shell_implementation[role];
            if (!constructor) {
                // This icon is not implemented (this is a bug)
                continue;
            }
            let indicator = new constructor();
            this.addToStatusArea(role, indicator, i);
        }
    },
    _insertStatusItem: function(actor, position) {
        let children = this._thirdTopBox.get_children();
        let i;
        for (i = children.length - 1; i >= 0; i--) {
            let rolePosition = children[i]._rolePosition;
            if (position > rolePosition) {
                this._thirdTopBox.insert_actor(actor, i + 1);
                break;
            }
        }
        if (i == -1) {
            // If we didn't find a position, we must be first
            this._thirdTopBox.insert_actor(actor, 0);
        }
        actor._rolePosition = position;
    },
    addToStatusArea: function(role, indicator, position) {
        if (this._statusArea[role])
            throw new Error('Extension point conflict: there is already a
            status indicator for role ' + role);

        if (!(indicator instanceof PanelMenu.Button))
            throw new TypeError('Status indicator must be an instance of
            PanelMenu.Button');

        if (!position)
            position = 0;
        this._insertStatusItem(indicator.actor, position);
        this._menus.addMenu(indicator.menu);

        this._statusArea[role] = indicator;
        let destroyId = indicator.connect('destroy', Lang.bind(this,
        function(emitter) {
            this._statusArea[role] = null;
            emitter.disconnect(destroyId);
        }));

        return indicator;
    },
    _onTrayIconAdded: function(o, icon, role) {
        if (this._status_area_shell_implementation[role]) {
            // This icon is legacy, and replaced by a Shell version
            // Hide it
            return;
        }

        icon.height = PANEL_ICON_SIZE;
        let buttonBox = new PanelMenu.ButtonBox();
        let box = buttonBox.actor;
        box.add_actor(icon);

        this._insertStatusItem(box, this._status_area_order.indexOf(role));
    },
    _onTrayIconRemoved: function(o, icon) {
        let box = icon.get_parent();
        if (box && box._delegate instanceof PanelMenu.ButtonBox)
            box.destroy();
    },
};
```