APERTIS

SDK

Contents

Software Development Kit **2**
Definitions . . . 2
Software Development Kit (SDK) Purpose . . . 2
API/ABI Stability Guarantees . . . 2
Reference System Image Composition . . . 3
System Image Software Licenses . . . 3
Development Workflow . . . 3
Typical Workflow . . . 3
On-device Workflow . . . 4
Workflow-simplifying Plugins . . . 4
3D acceleration within VirtualBox . . . 5
Simulating Multi-touch in VirtualBox . . . 6
Software-based solution . . . 6
Hardware-based solution . . . 9
Third-party Application Validation Tools . . . 10
Two-step Application Validation Process . . . 10
App Validation Tool . . . 10
General approach to third-party applications . . . 11

Software Development Kit

Definitions

- **Application Binary Interface (ABI) Stability**: the library guarantees API stability and further guarantees dependent applications and libraries will not require any changes to successfully link against any future release. The library may add new public symbols freely.

- **Application Programming Interface (API) Stability**: the library guarantees to not remove or change any public symbols in a way that would require dependent applications or libraries to change their source code to successfully compile and link against later releases of the library. The library may add new public symbols freely. Later releases of the API-stable library may include ABI breaks which require dependent applications or libraries to be recompiled to successfully link against the library. Compare to **ABI Stability**.

- **Backwards compatibility**: the guarantee that a library will not change in a way that will require existing dependent applications or libraries to change their source code to run against future releases of the library. This is a more general term than ABI or API stability, so it does not necessarily imply ABI stability.

- **Disruptive release**: a release in which backwards compatibility is broken. Note that this term is unique to this project. In some development contexts, the term “major release” is used instead. However, that term is ambiguous in general.

Software Development Kit (SDK) Purpose

The primary purpose of the special SDK system image will be to enable Apertis application and third-party library development. It will include development tools and documentation to make this process as simple as possible for developers. And a significant part of this will be the ability to run the SDK within the VirtualBox PC emulator. VirtualBox runs on ordinary x86 hardware which tends to make development much simpler than a process which requires building and running in-development software directly on the target hardware which will be of significantly lower performance relative to developer computers.

API/ABI Stability Guarantees

Collabora will carry along open source software components’ API and ABI stability guarantees into the Apertis Reference SDK API. In most cases, this will be a guarantee of complete API and ABI stability for all future releases with the same major version. Because these portions of Apertis will not be upgraded to later disruptive releases, these portions will maintain API and ABI stability at least for each major release of Apertis.

The platform software included in the Reference system images will be in the form of regular Debian packages and never in the form of application-level packages, which are described in the “Apertis Supported API” document. Collabora will manage API/ABI stability of the platform libraries and prevent conflicts between libraries at this level.

See the “Apertis Supported API” document for more details of specific components’ stability guarantees and the software management of platform, core application, and third-party application software.

Reference System Image Composition

See the document “Apertis Build and Integration”, section “Reference System Image Composition”.

System Image Software Licenses

See the document “Apertis Build and Integration” for details on license checking and compliance of software contained in the system images.

Development Workflow

Typical Workflow

Most developers working on specific libraries or applications will not be strictly dependent upon the exact performance characteristics of the device hardware. And even those who are performance-dependent may wish to work within the SDK when they aren't strictly tuning performance, as it will yield a much shorter development cycle.

For these most-common use cases, a typical workflow will look like:

1. modify source code in Eclipse
2. build (for x86)
3. smoke-test within the Target Simulator
4. return to step 1. if necessary

In order to test this code on the actual device, the code will need to be cross-compiled (see the document "Apertis Build and Integration Design", section "App cross-compilation"). To do so, the developer would follow the steps above with:

1. run Install to target Eclipse plugin
2. test package contents on device
3. return to step 1. if necessary

The development workflow for the Reference and derived images themselves will be much more low-level and are outside the scope of this document.

On-device Workflow

Some work, particularly performance tuning and graphics-intense application development, will require testing on a target device. The workflow [above][Typical workflow] handles this use case, but developing on a target device can save the time of copying files from a development machine to the device.

This workflow will instead look like:

1. modify source code as needed
2. run Install to target Eclipse plugin
3. test package contents on device
4. if debugging is necessary, either
    (a) run Remote app debugging Eclipse plugin; or
    (b) open secure shell (ssh) connection to target device for multi-process or otherwise-complex debugging scenarios

5. return to step 2. if necessary

Workflow-simplifying Plugins

Some of the workflow steps [above][Typical worflow] will be simplified by streamlining repetitive tasks and automating as much as possible.

Install to Target

This Eclipse plugin will automatically:

1. build the cross-architecture Apertis app bundle
2. copy generated ARM package to target
3. Install package

It will use a sysroot staging directory (as described in the document "Apertis Build and Integration Design", section "App cross-compilation") to build the app bundle and SSH to copy and remotely and install it on the target.

App bundle signature validation will be disabled in the Debugging and SDK images, so the security system will not interfere with executing in-development apps.

Remote App Debugging

This Eclipse plugin will connect to a target device over SSH and, using information from the project manifest file, execute the application within GDB. The user will be able to run GDB commands as with local programs and will be able to interact with the application on the device hardware itself.

This plugin will be specifically created for single application debugging. Developers of multi-process services will need to connect to the device manually to configure GDB and other tools appropriately, as it would be infeasible to support a wide variety of complex setups in a single plugin.

Sysroot Updater

This Eclipse plugin will check for a newer sysroot archive. If found, the newer archive will be downloaded and installed such that it can be used by the Install to target plugin.

3D acceleration within VirtualBox

Apertis will depend heavily on the Clutter library for animations in its toolkit and for custom animations within applications themselves. Clutter requires a working 3D graphics stack in order to function. Without direct hardware support, this requires a software OpenGL driver, which is historically very slow. Our proposed SDK runtime environment, VirtualBox, offers experimental 3D hardware "pass-through" to achieve adequate performance. However, at the time of this writing, this support is unreliable and works only on very limited host hardware/software combinations.

We propose resolving this issue with the new "llvmpipe" software OpenGL driver for the Mesa OpenGL implementation. This is the community-supported solution to replace the current, significantly-slower, "swrast" software driver. Both the upcoming versions of Fedora and Ubuntu Linux distributions will rely upon the "llvmpipe" driver as a fallback in the case of missing hardware support. The latest development version of Ubuntu 12.04, which Collabora is developing our Reference system images against, already defaults to "llvmpipe". Additionally, the "llvmpipe" driver implements more portions of the OpenGL standard (which Clutter relies upon) than the "swrast" driver.

In initial testing with an animated Clutter/Clutter-GTK application, llvmpipe performance was more than adequate for development purposes. In a VirtualBox guest with 2 CPU cores and 3 GiB of RAM, demo applications using the Roller widget displayed approximately 20-30 frames per second and had very good interactivity with the llvmpipe driver. In comparison, the same program running with the swrast driver averaged 4 frames per second and had very poor interactivity.

While this approach will not perform as well as a hardware-supported implementation, and will vary depending on host machine specifications, it will be the most reliable option for a wide variety of VirtualBox host operating system, configuration, and hardware combinations.

Simulating Multi-touch in VirtualBox

Because Apertis will support multi-touch events and most VirtualBox hosts will only have single pointing devices, the system will need a way to simulate multi-touch events in software. Even with adequate hardware on the host system, VirtualBox does not support multiple cursors, so the simulating software must be fully-contained within the system images themselves.

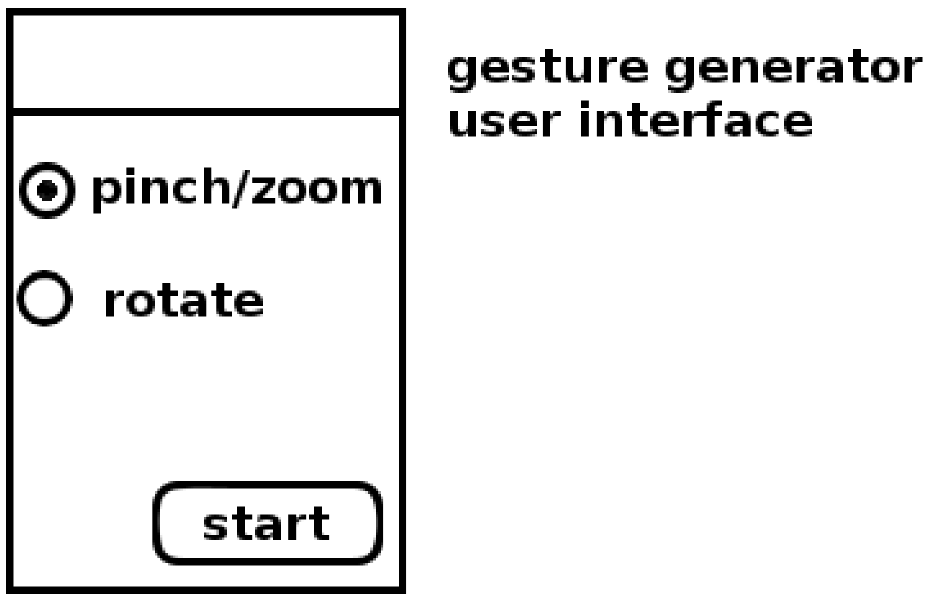
Software-based solution

We propose a software-based solution for generating multi-touch events within the SDK. This will require a few new, small components, outlined below.

In the intended usage, the user would use the Multi-touch gesture generator to perform a gesture over an application running in the Target Simulator as if interacting with the hardware display(s) in an Apertis system. The Gesture Generator will then issue commands through its uinput device and the Uinput Gesture Device Xorg Driver will use those commands to generate native X11 multi-touch events. Applications running within the Target Simulator will then interpret those multi-touch events as necessary (likely through special events in the Apertis application toolkit).

Multi-touch Gesture Generator

This will be a very simple user interface with a few widgets for each type of gesture to generate. The developer will click on a button in the generator to start a gesture, then perform a click-drag anywhere within VirtualBox to trigger a set of multi-touch events. The generator will draw simple graphics on the screen to indicate the type and magnitude of the gesture as the developer drags the mouse cursor.



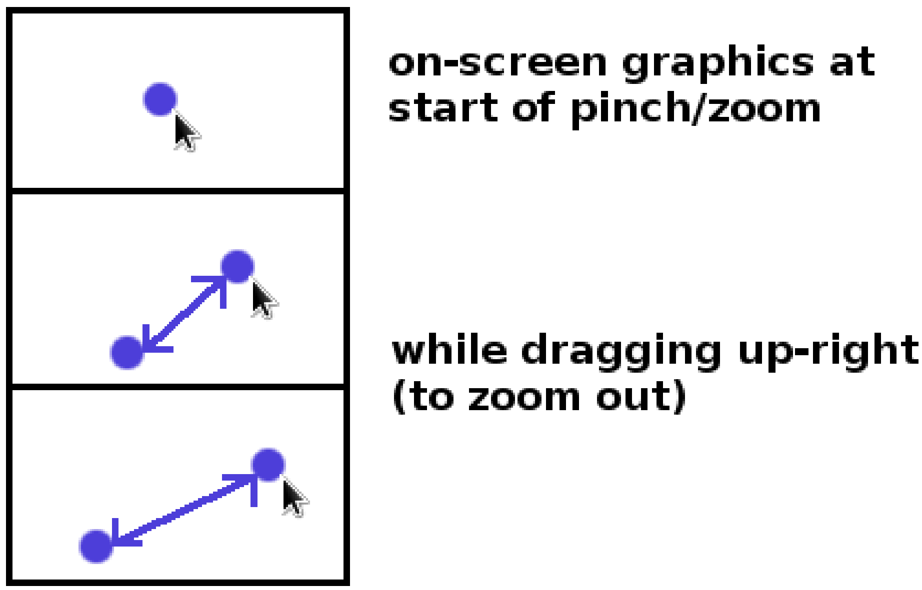
We anticipate the need for two gestures commonly used in popular multi-touch user interfaces:

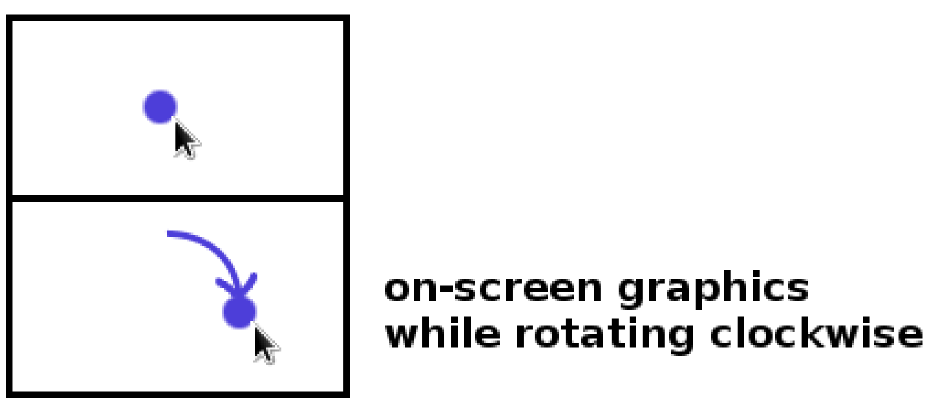
- **Pinch/zoom**: the movement of a thumb and forefinger toward (zoom-out) or away (zoom-in) from each other. This gesture has a magnitude and position. The position allows, e.g., a map application to zoom in on the position being pinched rather than requiring a separate zoom into the center of the viewable area, then a drag of the map.
  - Zoom-in: simulated by initiating the pinch/zoom gesture from the Gesture Generator, then click-dragging up-right. The distance dragged will determine the magnitude of the zoom.
  - Zoom-out: the same process as for zoom-in, but in the opposite direction
- **Rotate**: the movement of two points around an imaginary center point. Can be performed either in a clockwise or counter-clockwise direction. This gesture has a magnitude and position. The position allows, e.g., a photo in a gallery app to be rotated independent of the other photos.

– Clockwise: simulated by initiating the rotate gesture, then click-dragging to the right. This can be imagined as drag affecting the top of a wheel.

– Counter-clockwise: the same process as for clockwise, but in the opposite direction.

Additional gestures could be added during the specification process, if necessary.





Upon the user completing the simulated gesture, the Gesture Generator would issue a small number of key or movement events through a special uinput device (which we will refer to as the Uinput Gesture Device). Uinput is a kernel feature which allows “userland” software (any software which runs directly or indirectly on top of the kernel) to issue character device actions, such as key presses, releases, two-dimensional movement events, and so on. This uinput device will be interpreted by the Uinput Gesture Device Xorg Driver.

Uinput Gesture Device Xorg Driver

This component will interpret the input stream from our Uinput Gesture Device and generate X11 multi-touch events. These events would, in turn, be handled by the windows lying under the events.

X11 Multi-touch Event Handling

Windows belonging to applications running within the Target Simulator will need to handle multi-touch events as they would single-touch events, key presses, and so on. This would require to add support for multi-touch events in the Apertis application toolkit for applications to simply handle multi-touch events the same as single-touch events.

Hardware-based solution

An alternative to the software-based solution [above][Software-based solution] would be to use a hardware multi-touch pad on the host machine. This is a simpler solution requiring less original development though it brings a risk of Windows driver issues which would be outside of our control. Because of this, we recommend Collabora perform further research before finalizing upon this solution if this is preferred over the Software-based solution.

The touch pad hardware would need to be well-supported in Linux but not necessarily the host operating system (including Windows) because VirtualBox supports USB pass-through. This means that output from the touch pad would simply be copied from the host operating system into VirtualBox, where Xorg would generate multi-touch events for us.

The best-supported multi-touch device for Linux is Apple's Magic Trackpad. This device uses a Bluetooth connection. Many Bluetooth receivers act as USB devices, allowing pass-through to VirtualBox. In case a host machine does not have a built-in Bluetooth receiver or has a Bluetooth receiver but does not route Bluetooth data through USB, an inexpensive Bluetooth-to-USB adapter could be used.

Collabora has verified that multi-touch gestures on an Apple Magic Trackpad plugged into a Linux host can be properly interpreted within Debian running within VirtualBox. This suggests that a hardware-based solution is entirely feasible.

Hardware Sourcing Risks

Collabora investigated risks associated with selecting a single hardware provider for this multi-touch solution. The known risks at this point include:

1. Apple has a history of discontinuing product lines with little warning
2. As of this writing, there appear to be few alternative multi-touch pointing devices which are relatively inexpensive and support arbitrary multi-touch movements

In the worst case scenario, Apple could discontinue the Magic Trackpad or introduce a new version which does not (immediately) work as expected within Linux. With no immediate drop-in replacement for the Magic Trackpad, there would not be a replacement to recommend internally and to third-party developers using the Apertis SDK.

However, there are several mitigating factors that should make these minor risks:

1. Inventory for existing Magic Trackpads would not disappear immediately upon discontinuation of the product
2. Discontinuation of a stand-alone multi-touch trackpad entirely is very unlikely due to Apple's increasingly-strong integration of multi-touch gestures within iOS and Mac OS itself.
3. In case Apple replaces the Magic Trackpad with a Linux-incompatible version, there is significant interest within the Linux community to fix existing drivers to support the new version in a timely manner. For instance, Canonical multi-touch developers use the Magic Trackpad for their development and will share Apertis's sourcing concerns as well.
4. As an ultimate fallback, Multi-touch gesture generator can be recommended as an alternative source of multi-touch input.

Third-party Application Validation Tools

Two-step Application Validation Process

The third-party application process will contain two main validation steps which mirror the application submission processes for Android and iOS apps. The first, SDK-side validation checks will be performed by a tool described below. Developers may perform SDK-side validation as often as they like before submitting their application for approval. This is meant to automatically catch as many errors in an application as soon as possible to meet quality requirements for application review.

The second step of the application validation process is to validate that an application meets the app store quality requirements. It is recommended to set up a process where new applications automatically get run through this same Eclipse plugin as an initial step in review. This will guarantee applications meet the latest automated validation checks (which may not have been met within the developer's SDK if their Eclipse plugin were old). Developers will be able to easily stay up-to-date with the validation tool by applying system package updates within the SDK, so this difference can be minimized by a small amount of effort on the developer's part. Applications which pass this initial check will then continue to a manual evaluation process.

App Validation Tool

To streamline the third-party application submission process (which will be detailed in another document), Collabora will provide an Eclipse plugin to perform a number of

SDK-side validation checks up on the application in development. Collabora proposed checks are:

- **Application contains valid developer signing key** – developers must create a certificate to sign their application releases so verifying the source of application updates can be guaranteed. This check will ensure that the certificate configured for the application meets basic requirements on expiration date and other criteria.
- **Manifest file is valid** – the application manifest file, which will be used in the software management of third-party applications on the platform, must meet a number of basic requirements including a developer name, application categories, permissions, minimum SDK API, and more.
- **Application builds from cleaned source tree** – this step will delete files in the source tree which are neither included in the project nor belong to the version control system and perform a full release build for the ARMHF architecture. Build warnings will be permitted but build errors will fail this check.
- **AppArmor profile is valid** – the application's AppArmor profile definition must not contain invalid syntax or conflict with the Apertis global AppArmor configuration

Third-party application validation will be specified in depth in another document.

General approach to third-party applications

In most cases, third-party applications should not need to explicitly validate their access to specific system resources, delegating as much as possible to the SDK API or to other parts of the system. Preferably, these applications will specify system resource requirements in their manifest, such as permissions the application needs to function, network requirements, and so on. The main advantages of having these in the manifest file are using shared code to perform some of the actual run-time resource requests.

Note that this strategy implies a trade-off between how simple it is to write an application and how complex the supporting SDK and system components need to be to provide that simplicity. That is to say, it often makes sense to impose complexity onto applications, in particular when it's expected that only a few will have a given requirement or use case. This general approach should be kept in mind while designing the SDK and any other interfaces the system has with third-party applications and their manifests.