



DirectFB Software User's Guide

Revision History

Revision	Date	Change Description
DirectFB-SWUM104-R	02/11/14	Updated: <ul style="list-style-type: none">• Version of the software to 1.7.1 v1.1.• “Main Changes From DirectFB-1.4.17 Phase 1.0” on page 13.• Table 1: “Software Deliverables,” on page 14.• Table 2: “Documentation Deliverables,” on page 14.
DirectFB_SWUM103-R	10/21/13	Updated: <ul style="list-style-type: none">• Version of the software to 1.7.1 v1.0.• “Main Changes From DirectFB-1.4.17 Phase 1.0” on page 12.• Table 1: “Software Deliverables,” on page 13.• Table 2: “Documentation Deliverables,” on page 13.• Table 5: “Make Flags, Part 2,” on page 24. Added: <ul style="list-style-type: none">• “4K Graphics Support” on page 30.
DirectFB_SWUM102-R	11/07/13	Updated: <ul style="list-style-type: none">• Version of the software to 1.7.0 v0.5 and version of the User’s Guide to v1.5 throughout the manual.• Updated “Main Changes From DirectFB-1.4.17 Phase 1.0” on page 12 Added: <ul style="list-style-type: none">• BCM97145 to the list of supported platforms in “Supported Platforms” on page 59 Removed: <ul style="list-style-type: none">• All unsupported devices and platforms.• All but the final paragraph of the last note in “DirectFB Nexus Input Router” on page 52

Revision	Date	Change Description
DirectFB-SWUM101-R	03/01/13	<p>Updated:</p> <p>Version of the software to 1.4.17 v1.6 and version of the User's Guide to v1.4 throughout the manual.</p> <p>All occurrences of "refsw_server" to "nxserver," and of "refsw_serverlib.c" to "nxserverlib.c."</p> <p>Software version numbers in Table 1 on page 13</p> <p>"Step 4C: Building DirectFB-XS" on page 19</p> <p>"DirectFB-XS (Nexus Surface Compositor)" on page 31</p> <p>The dfb_clientid row in Table 6: "Run-time Environment Variables," on page 38</p> <p>Figure 2 on page 46 and Figure 3 on page 47</p> <p>"DirectFB Nexus Input Router" on page 54</p> <p>"DirectFB Unit Tests" on page 58</p> <p>"Supported Platforms", adding the BCM97445</p> <p>Added:</p> <ul style="list-style-type: none"> • BCM7445 to the list of supported chips in "Main Changes From DirectFB-1.4.15 Phase 2.1" on page 11 • Two more paragraphs to "Main Changes From DirectFB-1.4.15 Phase 2.1" on page 11 • BUILD_DIRECTFB_UNITTEST to Table 5: "Make Flags, Part 2," on page 25 • A CLIENT=y option to the note in "Running OpenGL ES 2.0 Graphics Applications" on page 33 and the "join" option to the code that comes two paragraphs after. • "DirectFB Google Test Framework:" on page 55 <p>Removed:</p> <ul style="list-style-type: none"> • All but the final paragraph of the last note in "DirectFB Nexus Input Router" on page 54
DirectFB-SWUM100-R	12/19/12	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

© 2014 by Broadcom Corporation
All rights reserved
Printed in the U.S.A.

Broadcom®, the pulse logo, Connecting everything®, and the Connecting everything logo are among the trademarks of Broadcom Corporation and/or its affiliates in the United States, certain other countries and/or the EU. Any other trademarks or trade names mentioned are the property of their respective owners.

Table of Contents

About This Document.....	10
Purpose and Audience	10
Acronyms and Abbreviations	10
Document Conventions	10
References	11
Technical Support.....	11
Section 1: Introduction.....	12
Overview	12
Prerequisites.....	12
Main Changes From DirectFB-1.4.17 Phase 1.0.....	13
Section 2: Deliverables	14
Section 3: Installation.....	15
Introduction.....	15
Section 4: Building.....	16
Step 1: Host Machine Tools Check	16
Step 2: Environment Variables.....	16
Step 3: Driver Build Check	18
Step 4A: Building DirectFB in Single-Application Mode	19
Step 4B: Building DirectFB in Multiapplication Mode	20
Step 4C: Building DirectFB-XS	21
Building DirectFB Tests.....	21
Building DirectFB Examples.....	22
Building ++DFB.....	22
Building Insignia Test Harness.....	22
Building the Tacho Test Harness	23
Building External Applications	23
Additional Make Targets	23
Additional make Flags	25
Section 5: Running DirectFB on the Target Platform	29
Standard DirectFB Single Application Mode	29
DirectFB Multiapplication Mode.....	30
DirectFB-XS (Nexus Surface Compositor or NxClient)	31
Running with Nexus Surface Compositor (NSC).....	31

Running with NxClient	32
4K Graphics Support	32
Running OpenGL ES 2.0 Graphics Applications	33
Running SaWMan (Multiapplication Mode).....	34
Running DirectFB Examples	34
Running ++DFB	35
Running Audio/Video Tests	35
playback_dfb.....	35
decode_server_dfb.....	36
decode_client_dfb.....	36
Run-Time Environment Variables	37
Section 6: Additional Information	38
Build System Information	38
Multiapplication Support with DirectFB.....	40
Running Non-DirectFB and DirectFB Applications	41
Running a Non-DirectFB Master Application.....	41
Running a DirectFB Master Application.....	42
Running a Non-DirectFB Slave Application	42
Running a DirectFB Slave Application	43
Terminating a Non-DirectFB application	43
Terminating a DirectFB application	43
Multiapplication Support with DirectFB-XS.....	43
DirectFB Memory Management	46
Section 7: Broadcom DirectFB Software Architecture	48
Platform Library Usage for Standard DirectFB	49
Platform Library Usage for DirectFB-XS	49
Graphics Driver	50
IR and Front Panel Drivers.....	50
DirectFB Nexus Input Router	52
DirectFB Google Test Framework	53
System Driver	54
ImageProvider Driver	55
Additions to the Stock DirectFB	55
Broadcom Specific DirectFB Unit Tests.....	55
Build System	56
Input Devices	56

Section 8: Testing DirectFB	57
Testing the IR Input	57
Testing Different Blitting and Drawing Modes	57
Performance Tests	58
Supported Platforms	58
Section 9: Frequently Asked Questions.....	59
How Do I Enable Debugging on a Per Module Basis in DirectFB?	59
How Do I Enable Back-Tracing in DirectFB?	59
How Can I Disable Hardware Acceleration and Use the Generic DirectFB Software Graphics Functions Instead?	60
How Can I Tell What Size Surfaces Are Being Created? Why Can't I See Memory for my Surface Being Allocated on Creation?	60
Blending Multiple Windows Together Does Not Look Right — Why?	60
How Do I Change the Cursor in DirectFB?	61

List of Figures

Figure 1: Steps in the Make Process.....39

Figure 2: A Virtual Layer/Frame Buffer Used in DirectFB Access44

Figure 3: DirectFB Multiapplication Architecture Using the Nexus Surface Compositor45

Figure 4: Overall DirectFB Software Architecture48

Figure 5: Platform Library Usage49

Figure 6: IR and Front Panel Drivers Software Architecture50

Figure 7: Key System Driver Responsibilities.....54

List of Tables

Table 1: Software Deliverables.....14

Table 2: Documentation Deliverables14

Table 3: Make Targets23

Table 4: Make Flags, Part 125

Table 5: Make Flags, Part 226

Table 6: Run-time Environment Variables.....37

Table 7: Supported Platforms.....58

About This Document

Purpose and Audience

This document describes how to build, install, and run DirectFB on a Broadcom set-top box reference platform. DirectFB is a thin library that provides hardware graphics acceleration, input device handling and abstraction, integrated windowing system with support for translucent windows and multiple display layers.

This document is aimed for individuals who have an engineering background and already know how to build the standard Broadcom reference software for a set-top platform. This document assumes the user is familiar with a standard Unix environment and build tools such as “make” and “gcc.”

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to:
<http://www.broadcom.com/press/glossary.php>.

Document Conventions

The following conventions may be used in this document:

<i>Convention</i>	<i>Description</i>
Bold	User input and actions: for example, type exit , click OK , press Alt+C
Monospace	Code: <code>#include <iostream></code> HTML: <code><td rowspan = 3></code> Command line commands and parameters: <code>wl [-1] <command></code>
<code>< ></code>	Placeholders for <i>required</i> elements: enter your <code><username></code> or <code>wl <command></code>
<code>[]</code>	Indicates <i>optional</i> command-line parameters: <code>wl [-1]</code> Indicates bit and byte ranges (inclusive): <code>[0:3]</code> or <code>[7:0]</code>

References

The references in this section may be used in conjunction with this document.



Note: Broadcom provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site (see [Technical Support](#)).

For Broadcom documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

<i>Document (or Item) Name</i>	<i>Number</i>	<i>Source</i>
Broadcom Items		
[1] <i>DirectFB Driver Feature List, Application Note</i>	DirectFB-AN10X-R	Broadcom CSP
[2] <i>Reference Platform Installation Guide for Linux Systems, Software User Manual</i>	STB_Platform-SWUM10x-R	Broadcom CSP
[3] <i>Nexus Usage Manual, Software User Manual</i>	STB_Nexus-SWUM20x-R	Broadcom CSP
[4] <i>Nexus Architecture Guide, Software User Manual</i>	STB_Nexus-SWUM10xR	Broadcom CSP
[5] <i>Nexus Development Guide</i>	Nexus_Development.pdf	Nexus code release
[6] <i>Nexus Multiprocessing Guide</i>	Nexus_MultiProcess.pdf	Nexus code release
Other Items		
[7] <i>DirectFB website</i>	http://www.directfb.org	–

Technical Support

Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (<https://support.broadcom.com>). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (<http://www.broadcom.com/support/>).

Section 1: Introduction

Overview

This document describes how to build, install, and run Direct Frame Buffer (DirectFB) on a Broadcom set-top box reference platform.

DirectFB is a software library for Linux®- and UNIX®-based operating systems with a small memory footprint that provides graphics acceleration, input device handling and abstraction layer, and integrated windowing system with support for translucent windows and multiple display layers on top of the Linux framebuffer without requiring any kernel modifications.

It is a complete hardware abstraction layer with software fallbacks for every graphics operation that is not supported by the underlying hardware. DirectFB adds graphical power to embedded systems and sets a new standard for graphics under Linux. See www.directfb.org for more details.

Prerequisites

Users must have the following before building and running DirectFB on a reference platform:

- A host PC or build server upon which to install the DirectFB source code and build it. It must have the Broadcom cross-compilation toolchain installed.
- A DHCP server running on the same network as the reference platform.
- Knowledge of the vi UNIX editor to be able to edit text on the reference platform.
- A Bash shell to execute the installation and build steps on the host/build server.
- It may be desirable to have a USB mouse and keyboard to use with the reference platform.

Main Changes From DirectFB-1.4.17 Phase 1.0

This release officially supports a subset of Broadcom STB chipsets, namely the BCM7231, BCM7241, BCM7358, BCM7425, BCM7429, BCM7435, and BCM7445. Other chipsets or platforms may work without problems, but these have not been fully tested and a warning message will be issued informing the user that the platform may not be usable.

DirectFB-1.7 has major changes from the earlier DirectFB 1.4 series.

- New core architecture with support for multiple graphics M2MC engines.
- Task Manager support which can be enabled with a run time option. When the Task Manager is enabled, DirectFB can use the graphics acceleration engine in parallel with the CPU to render graphics; improving graphics throughput. This feature is still under development and is not enabled by default. Task Manager is useful for low-end platforms.
- Videoprovider support. This DirectFB release has a feature supporting video playback from file/HTTP/RTP/UDP sources. This is only available in DirectFB-XS mode.
- Open-source DirectFB 1.7.x source tree now includes SaWMan, Fusiondale, Fusionsound, Voodoo, and One modules. Previously these modules were available as separate releases.
- All Broadcom-specific patches to the DirectFB components have been removed. The release uses the open-source components as is, from directfb.org.
- Support for 4K graphics with 3840x2160 and 4096x2160 display resolution on supported Broadcom platforms.
- Option to disable building of open-source libraries like libpng, jpeg, freetype, zlib by DirectFB. With this option, required open-source libraries can be built before building DirectFB. DirectFB will assume the presence of open-source libraries in the target directory and will not attempt to build them.
- Deep standby support. DirectFB now supports s3 standby mode. Fixes have been added to graphics driver for DirectFB to recover correctly from s3 standby mode.
- LXC support. Multiple DirectFB processes can be run from separate Linux containers. DirectFB uses IPC mechanism to share the Nexus handles among DirectFB processes, removing earlier shared memory usage.
- A version number is added to the DirectFB-Broadcom component. The version number can be found in `directfb/build/directfb_common.inc` as `DIRECTFB_BRCM_VERSION`. It is different than the open-source DirectFB version number (`DIRECTFB_VERSION`). The DirectFB-Broadcom version number is normally aligned to the Broadcom Trellis release version number.

Section 2: Deliverables

This section describes what is contained in the release, including software and documents.

Table 1: Software Deliverables

Description	Vendor Version	Licensing
DirectFB 1.7.1 reference software	v1.1	Broadcom SLA

Table 2: Documentation Deliverables

Description	Version/Date
<i>DirectFB Software Users Guide</i> (this document)	1.7
<i>DirectFB Feature List</i> (DirectFB-AN10x-R)	A18

Section 3: Installation

Introduction

This DirectFB release contains both, the open-source software and the Broadcom SLA-specific driver code.

All of the code inside the DirectFB-Broadcom directory, typically the DirectFB graphics, system, input, and image provider drivers, comes under the Broadcom SLA and is built as shared libraries.

The standard reference software (Nexus/Magnum) source code should be available (untarred) prior to installation of the DirectFB reference software.

On the host (build) machine, navigate to the root of the reference software source tree and type the following:

```
tar xzvf DirectFB-1.7.1_vxx_release_date.tgz
```

This overwrites any existing AppLibs/opensource/directfb directories (and subdirectories). Users are now ready to build the DirectFB source code from the AppLibs/opensource/directfb/build directory.

Section 4: Building

This section explains how to build standard DirectFB in single, multiapplication, and DirectFB-XS modes.

Step 1: Host Machine Tools Check

Before commencing the build, ensure that the version of GNU make is 3.80 or higher on your host build machine. To find which version of make is being used, type the following command:

```
make -version
```

An earlier version of make will result in DirectFB not being built. Upgrade your make package on the host build machine.

Step 2: Environment Variables

Ensure that the reference software environment variables are setup correctly. The important ones are as follows:

- NEXUS_PLATFORM
- BCHP_VER
- LINUX

Example:

```
export NEXUS_PLATFORM=97425
export BCHP_VER=B2
export LINUX=/opt/brcm/linux-3.3-3.0
```

By default, the Nexus and magnum drivers are built in user-space. This behavior can be overwritten by setting the NEXUS_MODE envvar to proxy. This ensures that the drivers are built in kernel-space with a proxy shim layer to translate the API calls to Linux syscalls (IOCTLs) and back again.

Example:

```
export NEXUS_MODE=proxy
```



Note: Speed up the build process on multiprocessor machines by ensuring that either MULTI_BUILD=y or MAKE_OPTIONS=-j? is set where ? specifies how many make jobs can be run in parallel. For example, make MAKE_OPTIONS=-j4.

By default, DirectFB and the drivers are built in DEBUG mode. This can have performance penalties and it is strongly recommended that the user switch to a non-DEBUG mode after monitoring the drivers and DirectFB for any warnings or errors. To switch to a non-DEBUG mode (a.k.a. RELEASE mode), ensure that the environment variable `B_REFSW_DEBUG` is set to `n`.

Example:

```
export B_REFSW_DEBUG=n
```

For MIPS systems, the default is to build DirectFB and the drivers in little-endian mode, this is the preferred option. If it is required to run the platform and DirectFB/drivers in BIG endian mode, then the `B_REFSW_ARCH` environment variable must be specified as `mips-linux`.

For ARM based systems `B_REFSW_ARCH` should be set to `arm-linux`.

Example:

```
export B_REFSW_ARCH=mips-linux
```

Finally, ensure that the `PATH` environment variable is set up correctly to point to the cross-compilation toolchain.

Example:

```
export PATH=/opt/toolchains/stbgcc-4.5.4-2.5/bin/:$PATH
```

Step 3: Driver Build Check

The DirectFB build system builds the Nexus/magnum drivers automatically, honoring the environment variable settings setup in [“Step 1: Host Machine Tools Check” on page 16](#). For special cases, it can be done manually by typing the command:

```
make -C nexus/build
```



Note: The build process can be sped up on multiprocessor machines by specifying the `-j` option to `make` (for example, `make -j4`)

Step 4A: Building DirectFB in Single-Application Mode

Standard DirectFB can be built in different modes of operation, known as single-application and multiapplication. Single-application mode is typically used in situations where there is only a single application accessing the DirectFB APIs. A single application is typically a single process with or without multiple threads. If more than one application or process is required to access DirectFB concurrently, then DirectFB must be built in multiapplication mode; see [“Step 4B: Building DirectFB in Multiapplication Mode” on page 20](#).

To build DirectFB in single application mode follow these steps:

```
cd AppLibs/opensource/directfb/build
make default tarball
```



Note: This will build DirectFB, along with the zlib, libpng, libjpeg, and freetype libraries from the AppLibs/opensource directory.

1. The build process first checks to see whether your host build tools are at least at the correct minimum version before proceeding. It then builds and installs the Nexus/magnum drivers with the correct configuration based on the environment variables set in [“Step 2: Environment Variables” on page 16](#).
2. The build process checks to see whether the DirectFB source tree already exists in AppLibs/opensource/directfb/src/DirectFB-version. If not, then the generic DirectFB-version.tar.gz tarball from the AppLibs/opensource/directfb/src/directfb_tarballs directory is untarred to create the AppLibs/opensource/directfb/src/DirectFB-version directory.
There is a build option DIRECTFB_BUILD_COMPONENTS=n that disables the building of open-source libraries and Nexus reference software. The option is useful when the required libraries are pre-built and are available in the target lib directory before starting the DirectFB build.
By default DIRECTFB_BUILD_COMPONENTS=y is set, which builds the required open-source libraries and Nexus reference software.
3. Before the DirectFB source code can be built, the freetype, jpeg, zlib, and png libraries must be built. The DirectFB source code is then configured to autogenerate the Makefiles and finally the DirectFB source code is built and installed.
4. After the vanilla DirectFB code is built, the Broadcom-specific DirectFB drivers are configured automatically for the platform chosen based on build-time options and parsing of the magnum/Nexus driver code. The Broadcom-specific DirectFB drivers, for applications that use the Broadcom Nexus drivers such as the Still Image decoder (SID), M2MC, and platform layer are now built from within the folder AppLibs/opensource/directfb/src/DirectFB-Broadcom.
5. The final build stage produces a tarball that can then be copied to the target platform for extracting and running. This tarball is given the name similar to the example below:
DirectFB-version_debug_build.97425B2.tgz



Note: In non-DEBUG (RELEASE) mode, the word debug is replaced with release.



Note: The string version in DirectFB-version refers to the default DIRECTFB_VERSION string defined in the AppLibs/opensource/directfb/build/directfb_common.inc file. For example, DirectFB-1.7.1.

Step 4B: Building DirectFB in Multiapplication Mode

Multiapplication mode allows multiple applications/processes to use the DirectFB API concurrently. The build process is the same as for [“Step 4A: Building DirectFB in Single-Application Mode” on page 19](#), except that the additional make build option called `DIRECTFB_MULTI=y` needs to be set.

Example:

```
cd AppLibs/opensource/directfb/build
make DIRECTFB_MULTI=y default tarball
```

Multiapplication mode is supported for both kernel-space (proxy mode) and user-space Nexus drivers.

The steps in the build process are slightly different, in that the fusion IPC kernel module (`linux-fusion`) will be built prior to freetype (and the other libraries) being built. The last stage of the build process is also different in that the SaWMan window manager is built. SaWMan allows finer control over the placement and life cycle of multiple applications' windows on the screen. It replaces the default window manager that comes with DirectFB.



Note: By default, when DirectFB is built for multiapplication mode, the SaWMan window manager is also built and overrides the default window manager.

If the user would rather use the default window manager instead of SaWMan for multiapplication mode, then the following make options can be specified:

`DIRECTFB_MULTI=y BUILD_SAWMAN=n`

The resulting tarball is named slightly differently to the one produced in step 4A. The word `multi` will appear immediately after `DirectFB-version`.

Example:

```
DirectFB-version_multi_debug_build.97425B2.tgz
```

This tarball can be copied to the target platform in the same way as [“Step 4A: Building DirectFB in Single-Application Mode” on page 19](#).

Step 4C: Building DirectFB-XS

DirectFB-XS allows non-DirectFB Nexus applications to be composited together with DirectFB applications using the Nexus Surface Compositor (NSC).

In this mode, DirectFB is built in single-application mode and multiple instances of DirectFB can run concurrently with other non-DirectFB applications. The Broadcom Trellis framework architecture uses this build mode of DirectFB to allow a mix of DFB and non-DFB applications to execute simultaneously.

To distinguish between DirectFB and DirectFB-XS builds, a new environment variable has been introduced named `DIRECTFB_NSC_SUPPORT`. When this is set to `y`, DFB is built in DirectFB-XS mode and all DirectFB applications are run as Nexus clients.

Example:

```
cd AppLibs/opensource/directfb/build
make DIRECTFB_NSC_SUPPORT=y default tarball
```

To run a DirectFB application, a Nexus server application must be built and launched first. For Nexus reference software with `nxClient` support, URSR Phase 13.1 and onwards, DirectFB builds the `nxserver` application from the `nexus/nxclient/server` directory when `DIRECTFB_NSC_SUPPORT=y` and `DIRECTFB_MASTER_LIB=y` are set.

If Nexus `nxClient` support is not available, for reference software URSR 12.4 or earlier DirectFB includes a sample Nexus server application called `nxsmaster`.



Note: For Nexus kernel mode builds, `DIRECTFB_MASTER_LIB` is automatically set. For user space Nexus builds, manually set this variable, as it requires a second complete build of Nexus, to reduce build time and target file system size, this option is not enabled by default.

Once successfully built, the resulting tarball is named in the same way as in [“Step 4A: Building DirectFB in Single-Application Mode”](#) on page 19.

Example:

```
DirectFB -1.7.1_debug_build.97425B2.tgz.
```

This tarball can be copied to the target platform in the same way as for step 4A.

Building DirectFB Tests

DirectFB test applications that reside in the `DirectFB-version/tests` directory are not built by default. To enable these unit tests ensure that the make build option `BUILD_TESTS` is set to `y`.

Example:

```
make BUILD_TESTS=y
```

Building DirectFB Examples

DirectFB examples are separate test/demo applications that are built by default and can be run on the Broadcom reference platforms. To disable these additional test/demo applications from being built, ensure that the make build option `BUILD_EXAMPLES` is set to `n`.

Example:

```
make BUILD_EXAMPLES=n
```

Building ++DFB

++DFB (a.k.a. ppDFB), is a library that a C++ application can call into to make DirectFB API calls (effectively a set of C++ bindings). ++DFB is a more advanced version of DFB++, and is compatible in the way applications can call methods/functions.

To build ++DFB, ensure that the make build option `BUILD_PPDFB` is set to `y`.

Example:

```
make BUILD_PPDFB=y
```



Note: ++DFB requires the standard C++ libraries be present on the target platform. The version of the ++DFB library that is built is set in `directfb_common.inc` as `PPDFB_VERSION += VERSION_STRING`.

Building Insignia Test Harness

The Insignia test harness checks that the Broadcom DirectFB graphics driver matches the software fallback implementation for over 1500 test cases. This package is built by setting the make build option `BUILD_INSIGNIA` to `y`.

Example:

```
make BUILD_INSIGNIA=y
```



Note: It is advisable to also set `DIRECTFB_HW_DITHERING=n`, since running without this option will cause failures in certain 16-bit pixel format bits due to added rounding differences.

Building the Tacho Test Harness

The Tacho test harness is only available to certain customers who have signed an SLA with YouView. This test harness will check the graphics performance of the Broadcom DirectFB graphics driver. If the Tacho tarball is present, then this package can be built by setting the make build option BUILD_TACHO to y.

Example:

```
make BUILD_TACHO=y
```

Building External Applications

Third party applications and/or external applications/tests can be built with the correct DirectFB compiler flags by using the directfb-config utility. The following example shows how to build a test application called my_test.c:

Example:

```
mipsel-linux-gcc `./DirectFB-1.7.1/directfb-config --cflags --libs` my_test.c -o my_test
```



Note: If the application to compile includes any Nexus headers and makes any Nexus calls, then the following additional flags can be passed to directfb-config for the application to be built more easily:

```
--extra-cflags --extra-libs
```

Additional Make Targets

The DirectFB build system allows for partial steps or targets to be chosen. These build targets are listed below along with a description:

Table 3: Make Targets

Make Target	Description
help	Lists the DirectFB make targets that can be called along with options.
default	This is the default make target. It attempts to install all DirectFB software modules if they have not already been installed. If a module has not been compiled, then it will first be compiled.
release	Creates a full release of the DirectFB software including both open-source and Broadcom SLA specific code. The result is a dated tarball.
all	Forces every DirectFB software module to be reconfigured, rebuilt and installed.
tarball	Creates a tarball of the target output directory that can be copied over to the target platform for unpacking and running.
install	This option is the same as the default target option and only installs software modules that need installing.

Table 3: Make Targets (Cont.)

Make Target	Description
compile	This attempts to compile all DirectFB software modules that need compiling. If a module has not already been configured, then it is configured before it is compiled.
config	This attempts to configure all DirectFB software modules that need configuring. If a module has not already been unpacked, then it will be unpacked before configuring.
uninstall	This causes all installed intermediate files to be uninstalled.
uninstall-target	This removes all target output directory installed files.
clean	Remove all generated object files, dependencies, binaries and temporary object directories.
distclean	Remove everything including the generated source code. The user is first prompted to remove any source code. This target should always be called prior to installing a new release of DirectFB.
mrproper	Remove everything including generated source code. No user prompts are displayed. This target should always be called prior to installing a new release of DirectFB.
check-tools	Check to ensure you have up-to-date tools to build DirectFB.
check-autogen-tools	Check to ensure you have up-to-date tools to auto generate the autoconf *.in files needed to build DirectFB.
directfb-defines	Updates the graphics and system defines files in the open-source DirectFB-version source tree.
xxx-all	Where xxx can be “,” “directfb,” “directfb-brcm,” “directfb-examples,” “sawman,” “fusion,” “ppdfb,” “divine,” “insignia,” “tacho,” “ffmpeg,” “freetype,” “jpeg,” “png” and “zlib.” This will re-build the chosen target software module including re-configuration and recompilation if necessary.
xxx-source	Like xxx-all above, but the source code for the chosen module xxx is created if not already present.
xxx-config	Like xxx-source above, but the configuration step is called.
xxx-compile	Like xxx-source above, but the compilation step is called.
xxx-install	Like xxx-source above, but the installation step is called.
xxx-uninstall	Like xxx-source above, but the uninstallation step is called.
xxx-clean	Only clean the required software module specified by xxx.
xxx-distclean	Perform a complete clean of the chosen software module specified by xxx with user prompt.
xxx-mrproper	Perform a complete clean of the chosen software module specified by xxx.
yyy-autogen	Where yyy can be “directfb,” “directfb-examples,” “sawman,” “FFmpeg” and “ppdfb.” This regenerates the “Makefile.in” and “configure” scripts from the *.am files. This option is only useful for developers who want to change the way the chosen software module is built (for example, build new test application).

Additional make Flags

Table 4 lists the complete set of flags that can be passed to the DirectFB build system to modify the default build behavior. The flags are normally passed on the make command line, but can also be set as environment variables.

Example:

```
make DIRECTFB_MULTI=y
export DIRECTFB_MULTI=y
```

Table 4: Make Flags, Part 1

Make Target	Description
DIRECTFB_VERSION	The version of DirectFB to be compiled can be overridden (for example, DIRECTFB_VERSION=1.7.1)
DIRECTFB_MULTI	The default is to build DirectFB in single application mode. Setting this flag to y builds DirectFB in multiapplication mode.
DIRECTFB_NSC_SUPPORT	The default is to build DirectFB in non-XS mode. Setting this flag to y builds in DirectFB-XS mode.
DIRECTFB_MASTER_LIB	The default is to build DirectFB with master Nexus libraries and to build DirectFB-XS with only client Nexus libraries. Setting this flag to y when DIRECTFB_NSC_SUPPORT=y is also set ensures that the DFB Nexus master server application, nxsmaster, is also built.
DIRECTFB_CLIENT_LIB	The default is to build single-application DirectFB with only master Nexus libraries and DirectFB-XS with client only Nexus libraries. This flag is not normally used by the end user.
DIRECTFB_SHARED	The default is to build the zlib, freetype, png and jpeg utility libraries as shared libraries that can then be dynamically linked with DirectFB at run time. Setting this flag to n builds these libraries as static (.a) and DirectFB will statically link with them at compile time. Setting this flag to n generally increases code size, but can be useful if applications use different versions of these utility libraries.
DIRECTFB_PREFIX	The default target prefix is /usr/local but this can be overridden using this flag (for example DIRECTFB_PREFIX=/usr). This specifies the path to the DirectFB installation on the target platform.
DIRECTFB_IR_PROTOCOL	The default IR protocol can be overridden using this flag. The IR protocol should be the name of the NEXUS IR protocol (for example, Generic, RemoteA, or CirNec).
DIRECTFB_IR_INPUT	The default is to enable the DirectFB IR input if the platform supports IR input. However, the user can specify n to disable it.
DIRECTFB_KEY_INPUT	The default is to disable the DirectFB front-panel keypad input. However, the user can specify y to enable it.
DIRECTFB_SID	The default is to build the DirectFB still image decoder (SID) image provider module/lib, if the platform supports a SID hardware decoder. However, the user can specify n to prevent this module from being built.

Table 4: Make Flags, Part 1 (Cont.)

Make Target	Description
DIRECTFB_SW_DITHERING	Enable software dithering (currently only supported with RGB16 and ARGB4444 formats). When set to <i>y</i> advanced software dithering for RGB16 and ARGB4444 formats are enabled. However, this increases the size of the data section by at least 64 KB. Default is <i>n</i> .
DIRECTFB_HW_DITHERING	Enable hardware dithering support for RGB16 and ARGB4444 pixel formats. The default is <i>y</i> . It is best to set this to <i>n</i> when running the Insignia test suite as some of the tests will fail due to the dithering.
DIRECTFB_SW_SMOOTH_SCALING	Enable software smooth scaling. When set to <i>y</i> software smooth scaling will be enabled and the size of the text section will increase by at least 100 KB.
DIRECTFB_GFX_PACKET_BUFFER	The default is to enable the packet buffer interface in the Broadcom DirectFB graphics driver. Setting this to <i>n</i> causes the legacy graphics driver to be used, which has lower graphics performance.
DIRECTFB_GFX_TRAPEZOID_SUPPORT	The default is to enable drawing of trapezoids in the graphics driver if the packet buffer interface is enabled too. Setting this option to <i>n</i> results in trapezoids being drawn using only software.
DIRECTFB_GFX_SOFT_MATRIX_SUPPORT	The default is to enable support for the SetMatrix() function in our graphics driver using the PX3D hardware to perform rotation and shearing. If the PX3D hardware is not present or this option is set to <i>n</i> , then a much limited feature set for SetMatrix() will be available.

Table 5: Make Flags, Part 2

Make Target	Description
BUILD_TESTS	The default is not to build the DirectFB unit test applications that are located in the tests directory. Setting this flag to <i>y</i> will instead build and install the additional DirectFB tests.
BUILD_EXAMPLES	The default is to build the additional DirectFB examples. Setting this flag to <i>n</i> disables the building and installing of the additional DirectFB examples.
BUILD_DIRECTFB_UNITTEST	The default is not to build unit tests. Setting this flag to <i>y</i> builds DirectFB unit tests in the Google® Test framework.
BUILD_SAWMAN	The default is to build SaWMan only when DIRECTFB_MULTI= <i>y</i> . Setting this flag to <i>n</i> disables building SaWMan and ensures that the default window manager of DirectFB is used. This option is only meaningful when building in multiapplication mode.
BUILD_PPDFB	The default is not to build the ++DFB library. Setting this flag to <i>y</i> builds and installs the library.
BUILD_FUSION	The default is to build the linux-fusion kernel module only when DirectFB is built in multiapplication mode. Setting this flag to <i>n</i> prevents this module from being built and installed and if used in conjunction with DIRECTFB_MULTI= <i>y</i> , the experimental multiapplication mode of DirectFB is built instead.
BUILD_FFMPEG	The default is <i>n</i> . To build the FFmpeg library that is used to decode MPEG-2 and H.264 I/IDR pictures set it to <i>y</i> . If it is not required to display MPEG-2/H.264 I/IDR still pictures, then leave it unset.

Table 5: Make Flags, Part 2 (Cont.)

Make Target	Description
BUILD_INSIGNIA	The default is not to build the Insignia library. Setting this flag to y builds and installs the library if the source tarball is present.
BUILD_TACHO	The default is not to build the Tacho library. Setting this flag to y builds and installs the library if the source tarball is present.
NEXUS_MODE	The default is to build NEXUS for user-space. Setting this option to proxy results in the NEXUS drivers being compiled for kernel-space.
B_REFSW_ARCH	The default for MIPS systems is to build DirectFB and the associated libraries in little endian mode. Setting this flag to mips-linux result in Nexus, DirectFB and its libraries being built in big endian mode instead. For ARM systems this should be set to arm-linux.
B_REFSW_DEBUG	The default is to build DirectFB in debugging mode. However, setting this flag to n builds DirectFB in release mode and no debugging information will be available. Setting this option to n also improves graphics performance.
B_REFSW_VERBOSE	The default is to build DirectFB with minimal information. Setting this flag to y will increase the amount of information available during the building stages.
TRACE	The default is to build DirectFB without any tracing information. Setting this flag to y allows tracing information to be enabled.
DIRECTFB_EXAMPLES_VERSION	The default is to build the DirectFB examples 1.6.0pre software package. If a different version is available, then seeing this flag results in that version being built (e.g., DIRECTFB_EXAMPLES_VERSION=1.2.1). The alternative software tarball should be placed in the directfb_tarballs directory before building the software.
FUSION_VERSION	Indicates the default build version of Linux fusion. If an alternative tarball version is available, it should be placed in the directfb_tarballs directory and this flag should be set accordingly (e.g., FUSION_VERSION=8.1.1).
SAWMAN_VERSION	Indicates the default build version of SaWMan library, if a different version of SaWMan is available, then setting this flag will result in that version being built (for example, SAWMAN_VERSION=1.4.15). The alternative software tarball should be placed in the directfb_tarballs directory before building the software. As of DirectFB 1.7.x, SaWMan is included as part of the DirectFB source tree.
FFMPEG_VERSION	Indicates the default build version of FFMPEG library. Setting this flag allows an alternative version of FFmpeg to be built (e.g., FFMPEG_VERSION=1.0.0). The alternative software tarball should be placed in the directfb_tarballs directory before building the software.
PPDFB_VERSION	Indicates the default build version of PPDFB. However, it can be overridden by specifying an alternative version (e.g., PPDFB_VERSION=1.4.0). The alternative software tarball should be placed in the directfb_tarballs directory before building the software. As of DirectFB 1.7.x, PPDFB is included as part of the DirectFB source tree.
INSIGNIA_VERSION	The default is to build Insignia version 1.0.1. However, this can be overridden by specifying an alternative version (e.g., INSIGNIA_VERSION=0.1.3). The alternative software tarball should be placed in the directfb_tarballs directory before building the software.

Table 5: Make Flags, Part 2 (Cont.)

Make Target	Description
TACHO_VERSION	The default is to build Insignia version 0.1.2. However, this can be overridden by specifying an alternative version (e.g., TACHO_VERSION=0.1.3). The alternative software tarball should be placed in the directfb_tarballs directory before building the software.
DFB_FREETYPE_VERSION	Indicates the default build version of freetype library from AppLibs/opensource/freetype. This can be overridden by setting this flag appropriately. For example, to build a different Freetype library version set "DFB_FREETYPE_VERSION=2.4.9."
DFB_JPEG_VERSION	Indicates the default build version of JPEG library from AppLibs/opensource/jpeg. This can be overridden by setting this flag appropriately. For example, to build a different JPEG library, set DFB_JPEG_VERSION=8d.
DFB_PNG_VERSION	Indicates the default build version of PNG library from AppLibs/opensource/libpng. This can be overridden by setting this flag appropriately. For example, to build a different PNG library, set DFB_PNG_VERSION=1.5.10."
DFB_ZLIB_VERSION	Indicates the default build version of ZLIB library from AppLibs/opensource/zlib. This can be overridden by setting this flag appropriately. For example, to build a different zlib library, set DFB_ZLIB_VERSION=1.2.6.
APPLIBS_INSTALL_PREFIX	The default is to install all files relative to /usr/local on the target platform. This option can be overridden to place the target files in a different directory structure.
APPLIBS_TARGET_TOP	This specifies the final output directory on the host build machine in which the DirectFB binaries and libraries are to be installed prior to being packed ready for transfer to the target platform. The default is AppLibs/target, but this can be overridden
APPLIBS_COMMON_INC	This specifies whether the AppLibs or DirectFB build process is used to compile the zlib, libpng, libjpeg, and freetype ancillary libraries. The default is n, for the DirectFB build process.
OPENSOURCE_TOP	This specifies the top-level directory where all open-source software components/libraries reside on the host machine. By default this is AppLibs/opensource.
NEXUS_TOP	This specifies the top-level directory where the Nexus reference software resides.
DIRECTFB_NEXUS_AV_SUPPORT	The default is to build DirectFB without video provider support. Setting this flag to y along with DIRECTFB_NXCLIENT_SUPPORT=y build option builds video provider support.
DIRECTFB_BUILD_COMPONENTS	The default is to build all the required open-source libraries such as freetype, zlib, libpng etc., and the Nexus reference software. Setting this flag to n disables building the open-source libraries and the Nexus reference software.
DIRECTFB_INET_IPC	Default is to use UNIX domain sockets in platform IPC. Setting this flag to "y" will enable INET socket type.
DIRECTFB_INET_IPC_PORT	Default is to use port number 6565 when using INET sockets in platform IPC.

Section 5: Running DirectFB on the Target Platform

This section explains how to run DirectFB and DirectFB-XS on different target platforms.

Standard DirectFB Single Application Mode

Once the tarball has been generated in [“Step 4A: Building DirectFB in Single-Application Mode” on page 19](#), copy it to the target platform. The most straightforward method is to run an NFS server on the build machine and mount the extracted file system onto the reference board.

```
mkdir -p /export/nfs/97425/  
tar -xzf DirectFB-version_debug_build.97425B2.tgz -C /export/nfs/97425/
```



Note: Ensure that the NFS server is running on the build machine and that the directory you have extracted the root file system into is exported in the `/etc/exports` configuration file.

Once the NFS server is configured, mount the file system on the reference platform and create a link to place the files in the correct location.

```
mount 192.168.0.1:/export/nfs/97425/usr /mnt/nfs;  
ln -s /mnt/nfs /usr
```

With the file system correctly mounted, run the installation script.

```
cd /usr/local/bin/directfb/1.7  
./rundfb.sh install
```

Any DirectFB application can now be run from this directory. For example, to run the `df_andi` (penguins) test, enter the following command:

```
./rundfb.sh df_andi
```

The output resolution default of the connected display type is 720p. To specify a different resolution use the following:

Example: To run with a 1080i output resolution:

```
./rundfb.sh df_andi --dfb:res=1080i
```

DirectFB Multiapplication Mode

With DirectFB running in multiapplication mode, more than one application can access the DirectFB APIs at the same time from different processes. The same steps can be taken as for single-application mode when it comes to running the first application. However, for any subsequent application, the `rundfb.sh` script needs to be used with the `join` option. An example of running both `df_andi` and `df_window` in multiapplication mode using different processes is given below:

Example:

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh install
./rundfb.sh df_window &
./rundfb.sh join df_andi --dfb:force-windowed,mode=640x480
```

The DirectFB Penguins application should now be seen on top of `df_window`. If a USB keyboard and mouse are connected to the platform, then you should be able to move the windows created by `df_window` around the screen. Pressing **Q** or **Esc** on the USB keyboard quits the application(s). Pressing **EXIT** on an infra-red handset also quits the application(s).

Users can also specify the size of the graphics surface/layer independent of the output resolution of the display. For example, if you would like to have a 640×480 graphics layer with a 1280×720p output resolution, you can use the mode DirectFB option.

Example:

```
./rundfb.sh join df_andi --dfb:force-windowed,mode=640x480
```

In some multiapplication usage modes, the graphics will be stretched horizontally and vertically to fill the display window. If the user would prefer not to have the graphics stretched to fill the display window, then the `scaled` option can be used in conjunction with `force-windowed`.

Example:

```
./rundfb.sh join df_andi --dfb:force-windowed,mode=640x480,scaled=640x480
```



Note: The first DirectFB application that runs is known as the master and subsequent DirectFB applications are known as slaves. The master application is normally a module that should not under normal circumstances be terminated. Internally within DirectFB it manages the Nexus display settings. If this application is terminated before the client applications are closed, then the system may be left in an unrecoverable state and may require a reboot.

DirectFB-XS (Nexus Surface Compositor or NxClient)

Before running any DirectFB applications in this mode, launch a Nexus server application. The recommended server application depends on the version of reference software being used. For Unified Reference Software Release (URSR) 13.1 and onwards, the newer NxClient API is the recommended solution while older releases use Nexus Surface Compositor (NSC).

The Nexus server application is the process that composites the different applications' frame buffers together. It is responsible for the size and position of the client applications' frame buffers and controls the settings of the display hardware.

It is possible to run other types of applications other than DirectFB and have them composited on the display at the same time, some of the Nexus graphics examples and OpenGL® test applications have been written to work this way.

Running with Nexus Surface Compositor (NSC)

If you have built DirectFB with the `DIRECTFB_NSC_SUPPORT=y` option and `DIRECTFB_MASTER_LIB=y` for userspace Nexus drivers, then the Nexus server application `nxsmaster` will have also been built. `nxsmaster` is a custom version of the reference software server application modified to provide all the resources a DirectFB application requires.

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh install
./rundfb.sh nxsmaster &
```

You can then launch any Nexus slave application. Specify the `join` option for all Nexus/DFB slave applications to ensure that the Nexus drivers are not reloaded.

Example: To run the `df_andi` (Penguins) test application as full screen, enter the following command:

```
./rundfb.sh join df_andi
```

There are some presets in the `nxsmaster` application that allow multiple applications to be positioned in different quadrants of the screen. You can specify the client/quadrant by using the `dfb_clientid` envvar before you launch an application.

Example: To launch `df_andi` in the top left-hand corner of the screen, enter the following command:

```
dfb_clientid=1 ./rundfb.sh join df_andi
```

To launch another instance of `df_andi` in the bottom right-hand corner of the screen, you may enter:

```
dfb_clientid=4 ./rundfb.sh join df_andi
```

Running with NxClient

If you have built DirectFB with `DIRECTFB_NSC_SUPPORT=y` and the reference software supports NxClient (URSR 13.1 or later), or if you have used the `DIRECTFB_NXCLIENT_SUPPORT=y` build option then the reference software server application `nxserver` should have been built and installed on the target file system. `nxserver` is provided as part of the reference software and may need to be modified to suit a particular use case.

Applications are run in a similar manner as to when using NSC, for example:

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh install
./rundfb.sh nxserver &
./rundfb.sh join df_andi
```

Multiple Nexus/DFB slave applications can also be run using `nxserver`. By default, applications will start in full screen mode. The `nxserver` application supports positioning of client applications on the screen. To enable positioning of the client applications on screen, start the server application with the option `./rundfb.sh nxserver -move on`. For more details on how the slave applications are positioned by the `nxserver`, see the file `nxserverlib.c` in the directory `nexus/nxclient/server`.

4K Graphics Support

This DirectFB release has added support for 4K display resolutions, supported by some Broadcom platforms like the BCM97445. It is recommended that the display framebuffer be set to 1920x1080 while setting the display resolution to 4K, as shown below. The framebuffer setting can be done by specifying “mode=1920x1080” option in target `/usr/local/etc/directfbrc` file or while running the application as follows:

Example:

```
df_andi --dfb:mode=1920x1080
```

```
{
    /*Example code to set 4K display resolution*/
    IDirectFB          *dfb;
    IDirectFBScreen    *primary_screen;
    DFBScreenEncoderConfig encoderCfg;

    DirectFBInit( &argc, &argv );
    DirectFBCreate( &dfb );

    dfb->GetScreen( dfb, DSCID_PRIMARY, &primary_screen );

    encoderCfg.resolution = DSOR_3840_2160;
    encoderCfg.scanmode = DSESM_PROGRESSIVE;
    encoderCfg.frequency = DSEF_24HZ;
    encoderCfg.flags      = (DFBScreenEncoderConfigFlags)(DSECONF_TV_STANDARD |
                                                         DSECONF_SCANMODE |
                                                         DSECONF_FREQUENCY |
                                                         DSECONF_RESOLUTION );

    encoderCfg.tv_standard = DSETV_DIGITAL;

    primary_screen->SetEncoderConfiguration(primary_screen, 0, &encoderCfg);
}
```

Running OpenGL ES 2.0 Graphics Applications

The DirectFB release does not contain any OpenGL ES 2.0 applications, but instead the `rockford/applications/opengles_v3d/v3d/directfb` directory contains a few example applications. These applications must be built after DirectFB has been built, they only run on chips that have a VC-4 3D graphics core, such as, the BCM7425.

To build any of these OpenGL ES 2.0 applications, simply type **make** in the appropriate application directory (ensuring that `V3D_SUPPORT=y` envvar is set first).

Example:

```
make -C rockford/applications/khronos/v3d/directfb/cube
```



Note: If you have built DirectFB in multiapplication mode, then ensure that you pass `DIRECTFB_MULTI=y` and `CLIENT=y` on the make command line.

For example:

```
make DIRECTFB_MULTI=y CLIENT=y -C rockford/applications/khronos/v3d/directfb/cube  
or
```

```
make DIRECTFB_MULTI=y CLIENT=y -C rockford/applications/khronos/v3d/directfb/earth_es2
```

After this step, type **make tarball** in the DirectFB build directory to create a tarball that contains the application(s) and the OpenGL driver and platform code.

This release supports running multiple OpenGL ES 2.0 applications simultaneously. For example, you can run `df_window`, `earth_es2`, and `cube` together when DFB is built in multiapplication mode with the `join` option, as follows:

```
./rundfb.sh df_window &  
./rundfb.sh join earth_es2 --dfb:force-windowed,mode=640x480 &  
./rundfb.sh join cube --dfb:force-windowed,mode=320x240
```

For platforms that support the Nexus Surface Compositor (NSC) or NxClient, consider this approach rather than using DirectFB to display OpenGL applications. This offers more flexibility and removes any dependency of the OpenGL application on DirectFB, reducing code size and increasing performance. This also means you can use DirectFB-XS, which uses DirectFB in single application mode that offers better performance and less CPU usage than multiapplication mode.

Running SaWMan (Multiapplication Mode)

SaWMan is the **Shared application and Window Manager** that overrides the default window manager of DirectFB. It can act as an application life-cycle manager, deciding what application/processes can be spawned or terminated and which application(s) receive input events. Many multiapplication environments use SaWMan to help fulfil their requirements for displaying and managing multiple applications simultaneously.

If DirectFB has been built in multiapplication mode, then the SaWMan becomes the default window manager. There are two specific applications that can be used to test SaWMan functionality. They are testman and testrun. In DirectFB 1.7.x, testman and testrun applications are built when DirectFB is built with option `BUILD_TESTS=y`. Testman is the main application manager and is used to register what applications can be spawned or terminated. It also has full control over the layout of multiple applications on the display. Testrun, on the other hand, is used to signal what preregistered application can be run. Testrun can be called from different processes multiple times, thus helping to simulate a real-world multiprocess/multiapplication environment. To test SaWMan, follow these steps:

1. Running with Kernel-Space (Proxy Mode) drivers

```
cd /usr/local/bin/directfb/1.7
./runsaw.sh testman &
./runsaw.sh join testrun Penguins
./runsaw.sh join testrun Penguins2
./runsaw.sh join testrun Penguins3
./runsaw.sh join testrun Penguins4
```

2. Running with User-Mode drivers

```
cd /usr/local/bin/directfb/1.7
./runsaw.sh testman &
cd /usr/local/bin/directfb/1.7
./rundfb.sh join df_andi --dfb:force-windowed &
./rundfb.sh join df_andi --dfb:force-windowed &
./rundfb.sh join df_andi --dfb:force-windowed &
./rundfb.sh join df_andi --dfb:force-windowed &
```

Four windows should be seen on the screen, each with their own df_andi (Penguins) moving around. Move the mouse over any of the windows and press <Q> to quit the application. Each of the applications is running in a separate process.

Running DirectFB Examples

DirectFB examples are additional example applications and tests that are built when the `BUILD_EXAMPLES=y` option is set. To test any of these additional example applications, follow the example steps below:

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh df_matrix
```

Running ++DFB

++DFB requires that the C++ standard libraries are installed on the target platform. These libraries are usually located in the `/lib` directory and are called `libstdc++.so`

If you have the C++ libraries installed, you should be able to run any of the ++DFB test applications. For example, you can run the `dfbshow` test application as follows:

```
cd /usr/local/bin/++dfb/1.7
./runppd.sh dfbshow /usr/local/share/directFB-version/biglogo.png
```

Running Audio/Video Tests

When building for multiapplication DirectFB (non-DirectFB-XS), there are three audio/video playback example tests:

- `playback_dfb`
- `decode_server_dfb`
- `decode_client_dfb`

playback_dfb

This is a DirectFB master and Nexus server application that is built when the Nexus server library (`libnexus.so`) is present.

This application can read in an MPEG-2 transport stream file and decode a particular packetized elementary stream within it specified by the audio and video PIDs. The audio and video codecs can also be specified on the command line.

It is possible to start this application running and then run any further DirectFB applications as slaves.

Example:

Run the playback example as a master DirectFB application (Nexus server):

```
./rundfb.sh playback_dfb --file <path to MPEG-2TS file> --vpid <video PID in decimal> --apid  
  <audio PID in decimal> --vcodec <video codec in decimal> --acodec <audio codec in decimal>.
```

Run `df_andi` as a DirectFB slave application (Nexus client):

```
./rundfb.sh join df_andi -dfb:force-windowed,mode=640x480
```



Note: You can find out what the audio and video codecs arguments are available by using the `--help` argument to `playback_dfb`. For example:

```
./rundfb.sh playback_dfb --help
```

decode_server_dfb

This is a Nexus server/DirectFB master application that is only built when the Nexus server library (libnexus.so) is present and DirectFB is compiled in multiapplication mode. This application initializes Nexus, opens up server-side Nexus modules/interfaces (for example, display or simple audio/video decoders), and starts the Nexus server. This application must be the first Nexus/DirectFB application in the system to be executed.

decode_client_dfb

This is a Nexus client application that is used to playback audio/video from an MPEG-2 TS file. It connects to the already-running decode_server_dfb application on the target platform. It accepts the same command line arguments as the playback_dfb application. The advantage this application has over playback_dfb is that it can be run as a Nexus client (slave) application. For example, to be able to play back audio/video in a Nexus client application while also running a DirectFB slave application (such as df_andi), follow the steps below:

From the main console:

```
./rundfb.sh decode_server_dfb
```

From a new virtual console (for example, telnet session):

```
./rundfb.sh join decode_client_dfb --file <path to MPEG-2 TS file> --vpid <video PID in decimal>  
--apid <audio PID in decimal> --vcodec <video codec in decimal> --acodec <audio codec in decimal>
```

From a new virtual console (for example, telnet session):

```
./rundfb.sh join df_andi --dfb:force-windowed,mode=640x480
```



Note: To find out which audio and video codecs are available use the --help argument to decode_client_dfb.

```
./rundfb.sh join decode_client_dfb --help
```

Run-Time Environment Variables

Table 6 lists the environment variables that affect the run-time behavior of DirectFB. These environment variables can be set using the export command.

Table 6: Run-time Environment Variables

Runtime Option	Description
dfb_slave	<p>This determines whether the DirectFB application should “join” Nexus (set to y) or initialize Nexus (set to n). This option can be set to y if another non-DirectFB application is the primary application in the system and has already initialized Nexus with NEXUS_Platform_Init().</p> <p>After initializing or joining Nexus, DirectFB can decide whether to open the display, graphics and picture decoder Nexus handles itself or use the handles provided to it in the DFB_Platform_Init() call.</p>
dfb_clientid	Used by client applications when running in DirectFB-XS mode to determine which client is attaching to the server. This envvar is not supported with the nxserver Nexus server application. It is only supported with the nxsmaster server application.
sw_picture_decode	This envvar only affects platforms that have a still image decoder (SID). Normally, JPEG, GIF, and PNG images are rendered using the SID. However, setting this envvar to any value will result in the software DirectFB picture decoding functions being used instead.
hdsd_mode	Set the HD/SD display mode. If this envvar is set to 0, then the composite/CVBS output is connected to the primary display 0 output. In this configuration, only SD display output resolutions are supported on both primary and secondary display outputs (for example, res=576i). If this envvar is not set or is a value other than 0, then the composite/CVBS output is connected to the secondary display 1 output and the primary display output can be configured to be either HD or SD. (Not available in DirectFB-XS)
DFBARGS	This is the standard DirectFB arguments envvar that can be used to specify the DirectFB run-time options (for example, export DFBARGS= “res=1080i”).

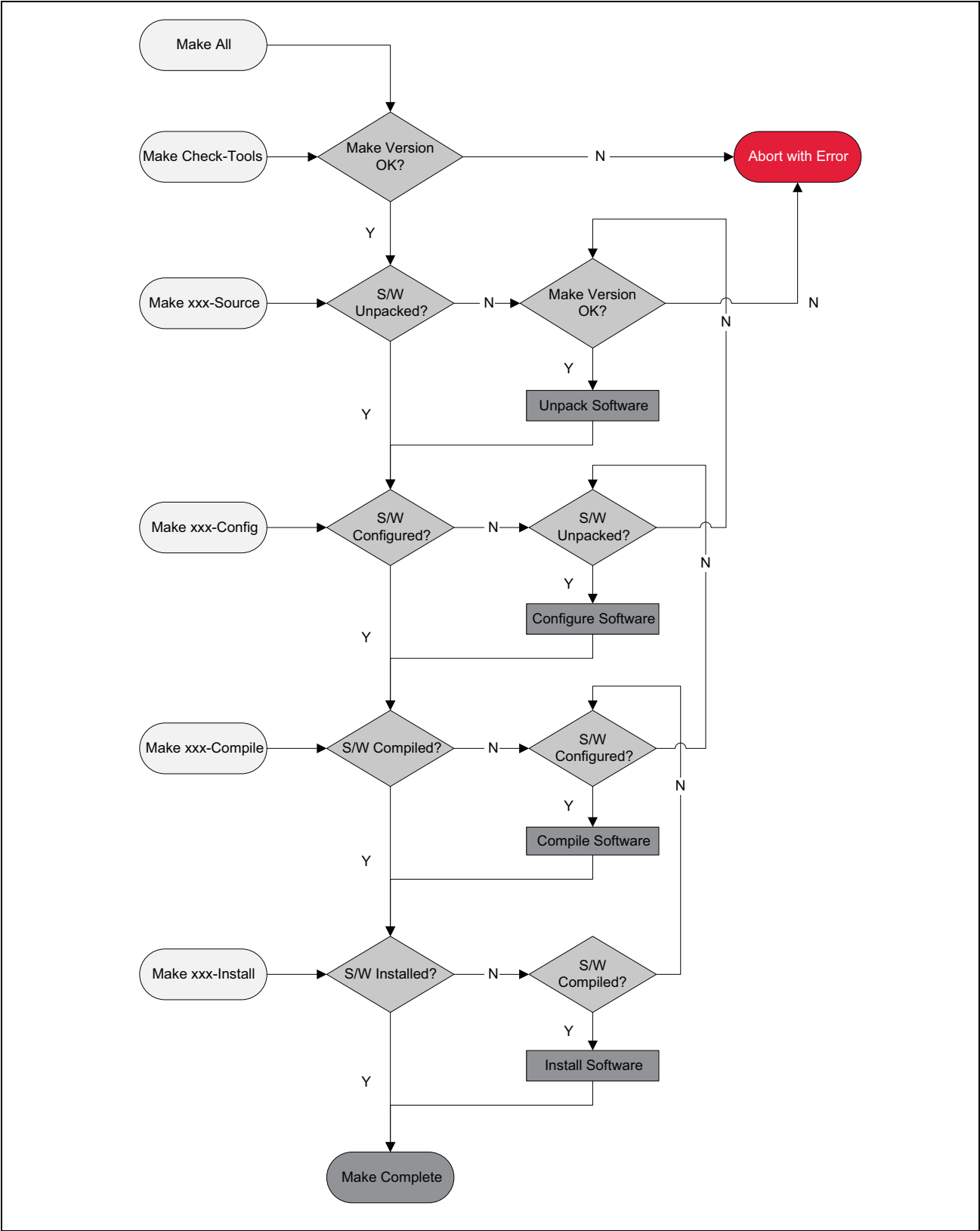
Section 6: Additional Information

Build System Information

The build system for DirectFB and its associated software components all resides in a top-level Makefile, include file (`directfb_common.inc`), and package includes in `packages/` in the `AppLibs/opensource/directfb/build` directory. The make process is broken down into different steps, each of which depends on a previous step.

[Figure 1 on page 39](#) shows how the user can enter any step directly, but the make system knows whether the previous step(s) have already been completed (dependencies).

Figure 1: Steps in the Make Process



It is worth noting that the build system does not track modified source code files between the stages shown in green. For example, if the user built and installed DirectFB by typing `make` and then modified a DirectFB open-source code file, the build system does NOT know that a source code file was modified if the user were to type `make` again. Instead, the user can type `make directfb-compile` and this will rebuild only the source files needed within DirectFB. The user can then type `make` and the build system is intelligent enough to know that the `directfb` installation phase needs to be completed next.

This approach saves time during the build process when making lots of source code modifications. If the makefile had to call each software modules compile stage, then it would also force an installation that would all consume valuable time. The recommended approach is to make source code modifications, type “`make xxx-compile`” (where `xxx` is the software module like “`directfb`”) and then type `make` for the build system to complete any further necessary steps (for example, installation). The same can be said if the user wants to reconfigure DirectFB or a software module. The user should type “`make xxx-config`” first to reconfigure the software module, and then type `make`. Usually this will involve the source code being recompiled and reinstalled.

Multiapplication Support with DirectFB

Standard DirectFB can be built in what is known as multiapplication mode. This mode allows multiple DirectFB and non-DirectFB applications to run in separate processes simultaneously. DirectFB multiprocess support is available when the Nexus drivers are built for proxy mode (kernel mode) and when the drivers are built for user-space.

The first DirectFB or non-DirectFB application that is executed on the target system is known as the master application and this application always needs to be running and must be the last application terminated on the target system. If this first application is a DirectFB application, then it will be responsible for receiving remote procedure calls (RPC) from client/slave DirectFB applications. This master DirectFB application is responsible for creating and destroying Nexus surfaces, allocating and freeing Nexus memory, graphics rendering using the underlying graphics hardware, and handling Nexus display settings (for example, setting the frame buffer). When a DirectFB slave application (Nexus client) tries to create a surface, to set the graphics frame-buffer or to use the DirectFB graphics operation APIs (for example, `Blit()`), the core DirectFB code will use secure-fusion to issue a RPC to the master DirectFB application to service the request. By default, secure-fusion is enabled so many of the DirectFB APIs are actually executed in a dispatch thread in the context of the master DirectFB process.

If the master application is terminated either intentionally or unintentionally, then the system will be in an unstable state, since client/slave applications won't be able to have their RPC requests serviced.

Running Non-DirectFB and DirectFB Applications

There are some situations where the system may have non-DirectFB applications and DirectFB applications running concurrently. In this scenario, the non-DirectFB application may have already initialized Nexus and opened Nexus modules that the DirectFB application(s) rely on (for example, Nexus display, graphics2d, graphics3d, picture decoder, etc.).



Note: For newer systems, consider using Nexus Surface Compositor to handle this situation.

To allow for this usage scenario, there is a `dfb_platform.h` file that contains a lightweight API that non-DirectFB and DirectFB applications can use.

Running a Non-DirectFB Master Application

There are two ways in which a non-DirectFB master application can initialize Nexus:

1. It can continue to use the existing `NEXUS_Platform_GetDefaultSettings()` and `NEXUS_Platform_Init()` API. It will also need to start the Nexus server using the `NEXUS_Platform_GetDefaultStartServerSettings()` and `NEXUS_Platform_StartServer()` APIs (see the example “[decode_server_dfb](#)” on page 36).

It can then inform the Broadcom DirectFB platform layer code that Nexus has already been initialized by making a call to “`DFB_Platform_GetDefaultSettings(DFB_PlatformClientType_eMasterNexusInitialized, pSettings)`” followed by a call to `DFB_Platform_Init(pSettings)`. The client type in the call to `DFB_Platform_GetDefaultSettings()` specifies that this is a master application and that Nexus has already been initialized.

By default, the `DFB_Platform_Init()` function will automatically attempt to open up the Nexus display, graphics2d, graphics3d, and picture decoder modules and will place these module handles in System V shared memory to allow other processes to access them. If the master non-DirectFB application has already opened up the Nexus modules, then the Nexus handles can be passed into the `DFB_Platform_Init()` function and the DFB platform code will know not to attempt to open the Nexus modules again.

The `DFB_Platform_Init()` function will also try to automatically connect up certain outputs to the display(s). Normally, the HDMI/component outputs will automatically be connected to the primary HD display and the composite output will be connected to the secondary SD display. If the non-DirectFB master application wants to override this default behavior, then it can indicate what outputs should be connected to what display by setting appropriate flags in the `DFB_PlatformSettings` structure.

2. It can replace the calls to `NEXUS_Platform_GetDefaultSettings()`, `NEXUS_Platform_Init()`, `NEXUS_Platform_GetDefaultStartServerSettings()`, and `NEXUS_Platform_StartServer()` with `DFB_Platform_GetDefaultSettings(DFB_PlatformClientType_eMasterNexusUninitialized, pSettings)` and `DFB_Platform_Init(pSettings)`. The application can still pass in certain Nexus platform settings through this API.

By default, the display, graphics2d, graphics3d, and picture decoder Nexus modules will automatically be initialized and outputs will be connected to the display (for example, HDMI, component). This is the simplest approach if the non-DirectFB application does not need to modify Nexus platform settings that are not exposed within the `DFB_PlatformSettings` structure.

Running a DirectFB Master Application

For a master DirectFB application, there are three ways in which Nexus can be initialized:

1. The simplest approach is not to do anything and rely on the internal Broadcom DirectFB system driver to automatically call into the DFB Platform code with default values to initialize Nexus and open up the display, graphics2d, graphics3d, and picture decoder Nexus modules. Outputs like HDMI and component will automatically be connected to the primary display. This is the preferred method, because it means the DirectFB application does not need to be modified to make any explicit `DFB_Platform_xxx()` calls and does not need to link with the DFB platform shared library (`libinit.so`).
2. Use the `DFB_Platform_GetDefaultSettings` (`DFB_PlatformClientType_eMasterNexusUninitialized`, `pSettings`) and `DFB_Platform_Init`(`pSettings`) APIs to explicitly initialize Nexus. Again, by default the Nexus display, graphics2d, graphics3d, and picture decoder modules will automatically be opened and outputs will be connected to the display. This option is useful if the default platform values are not suitable and need modifying.
3. Use the `NEXUS_Platform_GetDefaultSettings()`, `NEXUS_Platform_Init()`, `NEXUS_Platform_GetDefaultStartServerSettings()` and `NEXUS_Platform_StartServer()` APIs to explicitly initialize Nexus with non-default values.
Then use `DFB_Platform_GetDefaultSettingsExtended`(`DFB_PlatformClientType_eNexusMaster`, `&platformSettings`, `initOrJoinNexus`) with boolean variable `initOrJoinNexus` set to false and `DFB_Platform_Init`(`pSettings`) to inform DirectFB that Nexus as already been initialized and that the Nexus server has been started. This option is useful if non-default Nexus platform settings need to be provided. This case is rarely used for DirectFB master applications. Refer to DirectFB example applications `decode_server_dfb.c` and `decode_client_dfb.c` for details.
4. When DirectFB is built in XS mode with Nexus `nxClient` support and the application has explicitly joined Nexus by calling `NEXUS_Platform_Join()` then use `DFB_Platform_GetDefaultSettingsExtended`(`DFB_PlatformClientType_eNxClient`, `&platformSettings`, `initOrJoinNexus`) with boolean variable `initOrJoinNexus` set to false.

Running a Non-DirectFB Slave Application

A non-DirectFB slave application (Nexus client) is one in which another DirectFB or non-DirectFB application is the master and initialized Nexus. A slave application cannot initialize Nexus, but can join it and can open up additional Nexus modules for its own use. Any Nexus handles that were opened in either the DirectFB master application or another Nexus application cannot be shared and used in future Nexus calls in a different process if the Nexus client process(es) are run in either “untrusted” or “protected” Nexus security modes. Refer to the Nexus Multiapplication document ([Reference \[6\] on page 11](#)) for more information.

There are two ways in which a non-DirectFB slave application can be run:

1. Simply call `DFB_Platform_GetDefaultSettings` (`DFB_PlatformClientType_eSlaveNexusUninitialized`, `pSettings`), followed by `DFB_Platform_Init`(`pSettings`).
2. Make an explicit call to `NEXUS_Platform_Join()` to join Nexus. Then call `DFB_Platform_GetDefaultSettingsExtended`(`DFB_PlatformClientType_eNexusMaster`, `&dfbPlatformSettings`, `initOrJoinNexus`); followed by `DFB_Platform_Init`(`pSettings`). Where boolean variable `initOrJoinNexus` is set to false to indicate DirectFB not to join Nexus again. This case would rarely be used, as it requires an additional call to Nexus.

Running a DirectFB Slave Application

For slave DirectFB applications, Nexus will have already been initialized. However, slave DirectFB applications can still have control over whether a particular Nexus module that DirectFB depends on should be opened, as long as only that process consumes the Nexus module (in untrusted and protected Nexus security modes.)

There are two ways in which a DirectFB slave application can be run:

If the slave DirectFB application(s) doesn't need to perform any additional platform initialization or configuration, then they can simply ignore the Broadcom DirectFB platform API entirely. Internally, the DirectFB system driver will call into the platform code to join Nexus and to obtain already opened Nexus module handles.

If the slave DirectFB application(s) would like to perform any additional initialization or configuration, then they can do so by using `DFB_Platform_GetDefaultSettings(DFB_PlatformClientType_eSlaveNexusUninitialized, pSettings)`, followed by `DFB_Platform_Init(pSettings)` API calls. The application would have to explicitly link against the DirectFB platform library (`libinit.so`).

Terminating a Non-DirectFB application

Non-DirectFB applications that have called `DFB_Platform_Init()` must explicitly call `DFB_Platform_Uninit()` to ensure the system is in a consistent state. This function will release resources that were previously acquired by the `DFB_Platform_Init()` call, such as Nexus modules and outputs. It will only release resources that were opened/created in the call to `DFB_Platform_Init()` and in the context of that process.

Terminating a DirectFB application

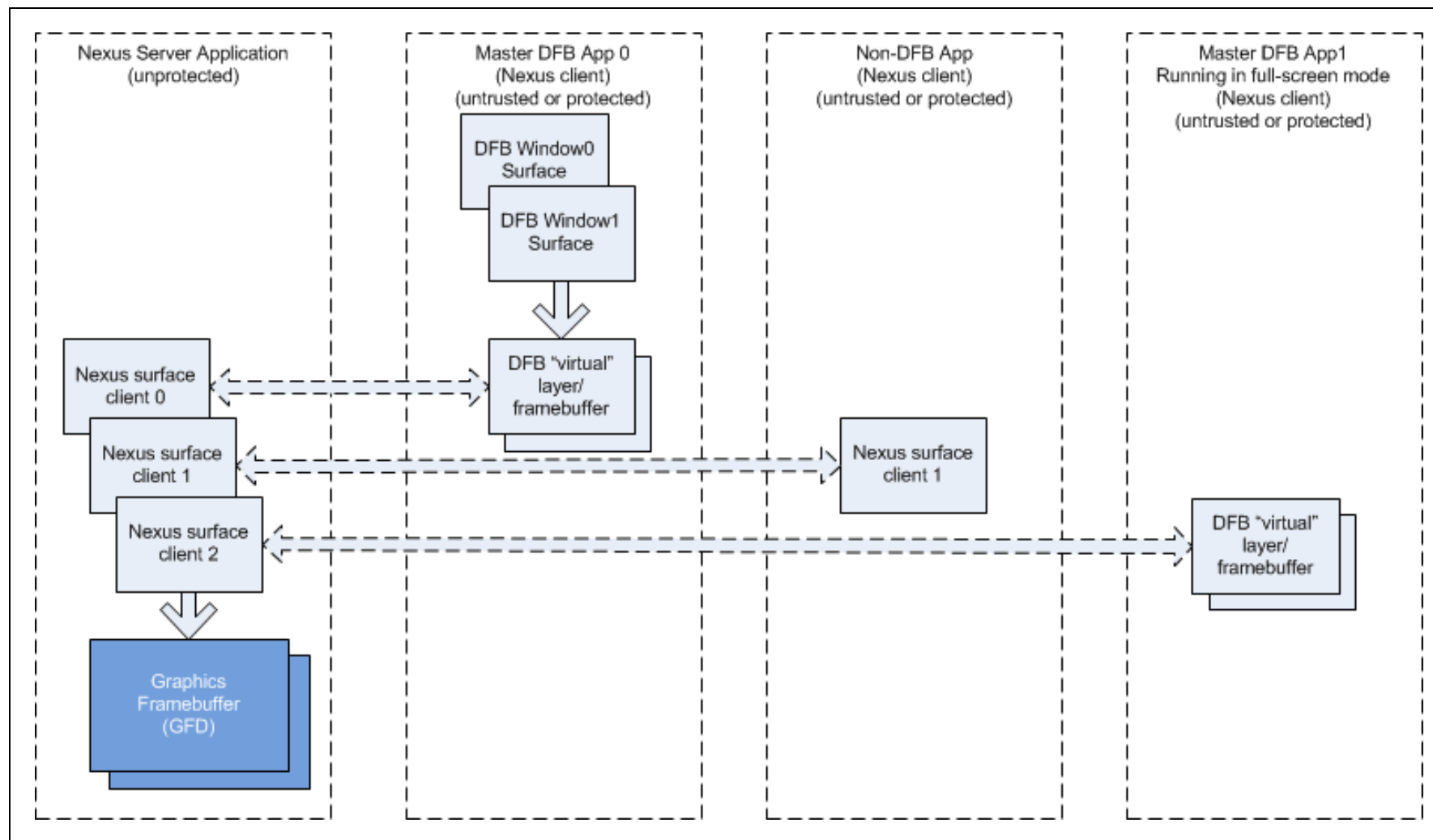
When a DirectFB application is to be terminated, it can either explicitly call `DFB_Platform_Uninit()` to ensure the system is in a consistent state or can let the DirectFB system driver automatically call this function upon shutdown. This function will release resources that were previously acquired by the implicit or explicit call to `DFB_Platform_Init()` call, such as Nexus modules and outputs.

Multiapplication Support with DirectFB-XS

DirectFB-XS is a Broadcom terminology for running DirectFB with an eXternal Surface compositor (Nexus surface compositor). In this usage scenario, multiple DirectFB and non-DirectFB applications can run concurrently and their graphics outputs will be positioned, sized, and blended together by the Nexus surface compositor before being provided to the graphics feeder for output on the associated display. DirectFB can only be built in single-application mode and multiple instances of single-app DirectFB applications can be run simultaneously with non-DirectFB applications.

Instead of DirectFB accessing a physical frame buffer when it composites graphics on the DirectFB layer, a virtual layer/frame buffer is provided. This virtual frame buffer is, in fact, just a Nexus surface that is blended with other Nexus surfaces by the Nexus surface compositor. [Figure 2 on page 44](#) helps to illustrate this behavior.

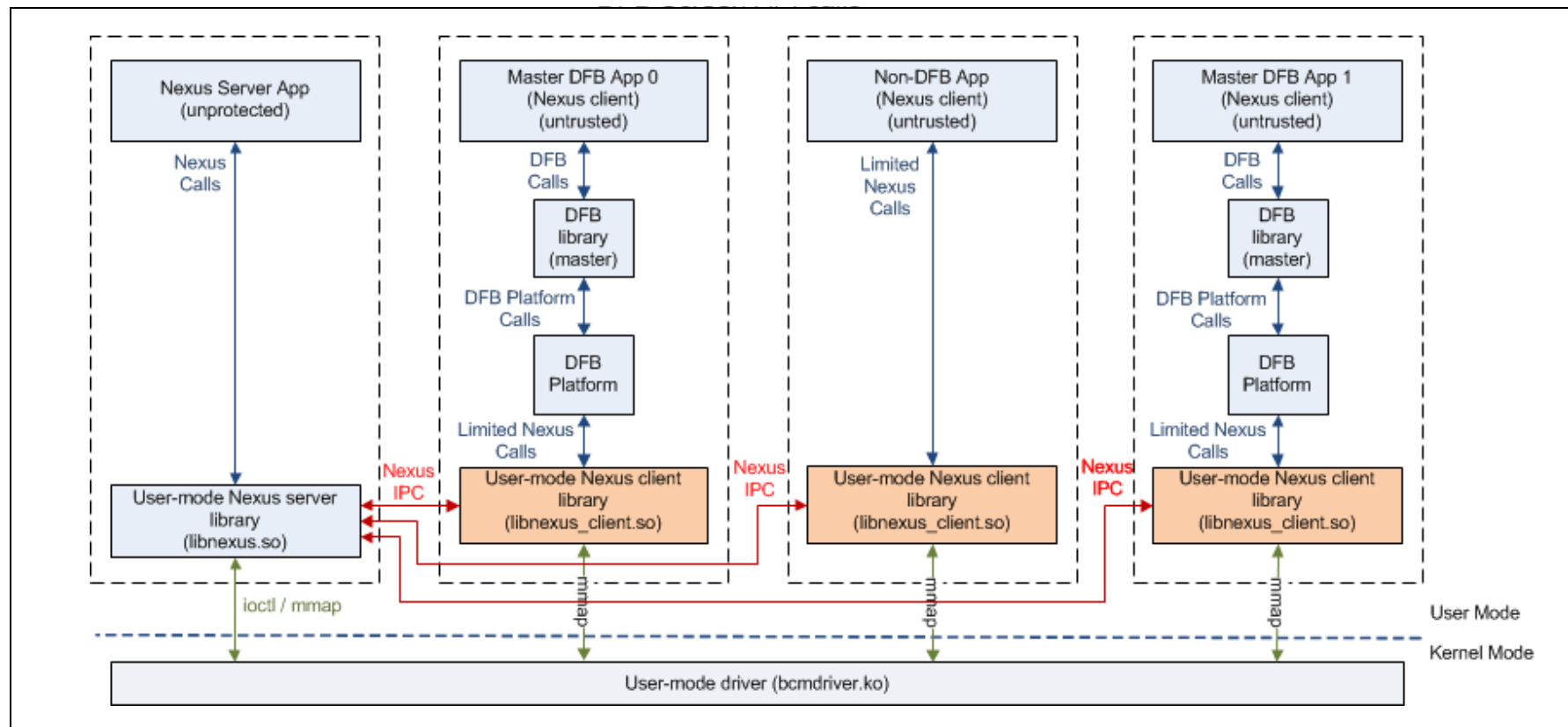
Figure 2: A Virtual Layer/Frame Buffer Used in DirectFB Access



In [Figure 2 on page 44](#), a Nexus server application contains the Nexus surface compositor. This is used to composite Nexus surfaces from different process application spaces. The Master DFB App 0 and Slave DFB App processes help to illustrate what occurs when DirectFB is built and run in multiapplication mode. Here there is a conventional DFB master application that uses the DirectFB window compositor to composite graphics from its application and a DFB slave application on to a virtual frame buffer. This virtual frame buffer is then pushed to the Nexus surface compositor and blended with other virtual frame buffers from “non-DFB App” and other “Master DFB App” applications. The “Master DFB App” application shown on the far right of the diagram can be another instance of DirectFB that has been built in single-application mode.

[Figure 3](#) helps to illustrate what software modules each type of application can call.

Figure 3: DirectFB Multiapplication Architecture Using the Nexus Surface Compositor



What can be seen is that neither the DirectFB nor the non-DirectFB applications directly call the Broadcom DFB platform library. Instead, the DFB platform layer is only called internally by the DirectFB application through the core DFB code (during the system initialize/join call). This means that non-DirectFB applications no longer need to call the DFB platform APIs in order to inform DFB whether the system has been initialized or not.

Each of the DirectFB and non-DirectFB applications are treated as Nexus clients. They will all join Nexus, rather than initializing Nexus. Only the Nexus server application is used to initialize Nexus and it is typically the application that also opens any audio/video outputs. It is the process that will instantiate the Nexus surface compositor and will typically be the process that instantiates the Nexus simple audio/video decoders (for playing back of audio/video from a Nexus client application(s)).

[Figure 3 on page 45](#) also helps to illustrate how Nexus client applications communicate with a Nexus server. In [Figure 3 on page 45](#), Nexus has been built to run in user-space. Each of the DFB and non-DFB applications will inherently link with the client Nexus user space-shared library. Nexus calls that affect the underlying hardware will automatically be “marshalled” across to the “Nexus server” application, using the auto-generated thunk (Nexus IPC in the diagram). Here, a master server instance of the Nexus library will actually make the call to the underlying hardware.

For more information about the Nexus multiapplication architecture, refer to [Reference \[6\] on page 11](#), which is bundled as part of the reference software release.

DirectFB Memory Management

This is a brief overview of some of the principles of how Broadcom's DirectFB implementation uses memory.

DirectFB uses three sources of memory:

- Linux memory (ignored for this discussion)
- Primary on screen display frame buffers
- Off-screen graphics memory

Memory for cases 2 and 3 are available from the Nexus heaps and are requested using the `NEXUS_Platform_GetFramebufferHeap()` function in the Nexus platform code.

For case 2, the call used is `NEXUS_Platform_GetFramebufferHeap(0)` for the main display and this needs to return a handle to a heap that is of type `NEXUS_MemoryType_eFull`.

For case 3, the memory is available from the call to `NEXUS_Platform_GetFramebufferHeap(NEXUS_OFFSCREEN_SURFACE)` and this should return a heap handle of type `NEXUS_MemoryType_eApplication`. If DirectFB fails to get a valid heap for case 3, the off-screen memory, we can set the off-screen heap to be the same as for case 2, the primary heap.

In a DirectFB multiapplication system, when we set the client application permissions, we only pass across the heap settings for the off-screen heap (case 3). All the allocations the slave application makes should be from this heap. If the heap is full, it returns an out-of-memory error.

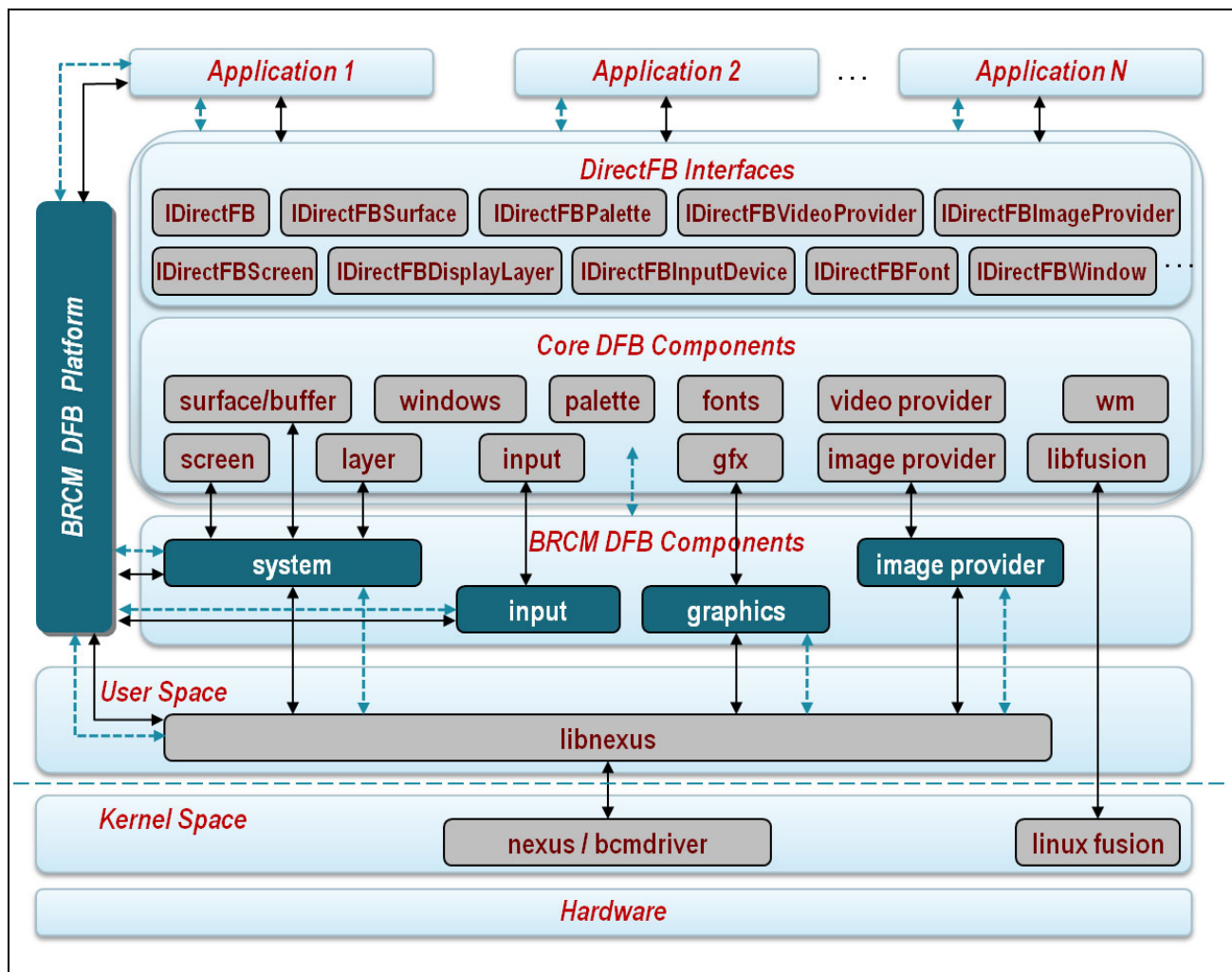
For a DirectFB master application, if the off-screen heap is full, we are able attempt to allocate from the primary heap (2), but not for a slave as there is no virtual memory mapping in the slave application's address space. For security reasons, we try and make sure that the slave applications have no access to hardware or display buffers to reduce the chance of a malicious slave app causing the system any damage or hijacking the display. If you want to sandbox the slave applications memory usage, you could create a separate heap inside Nexus and return this handle via `NEXUS_Platform_GetFramebufferHeap(NEXUS_OFFSCREEN_SURFACE)`. This way the slave applications would have the most limited access to any memory from which they could affect other system components.

Section 7: Broadcom DirectFB Software Architecture

Figure 4 helps to show the overall DirectFB software architecture with the Broadcom proprietary DFB software components highlighted in dark blue. These components are all shared libraries that are dynamically loaded into the system at run time. They are dynamically linked with the rest of DirectFB to provide additional features and performance improvements. Only the Broadcom DirectFB platform shared library is accessible to applications; all the remaining Broadcom DirectFB components are not directly accessible by applications, but instead are accessible through the DirectFB API (interfaces).

All these Broadcom proprietary software components rely on Nexus and are accessible through a shared library (libnexus). Nexus provides access to the underlying hardware through a well-defined API.

Figure 4: Overall DirectFB Software Architecture

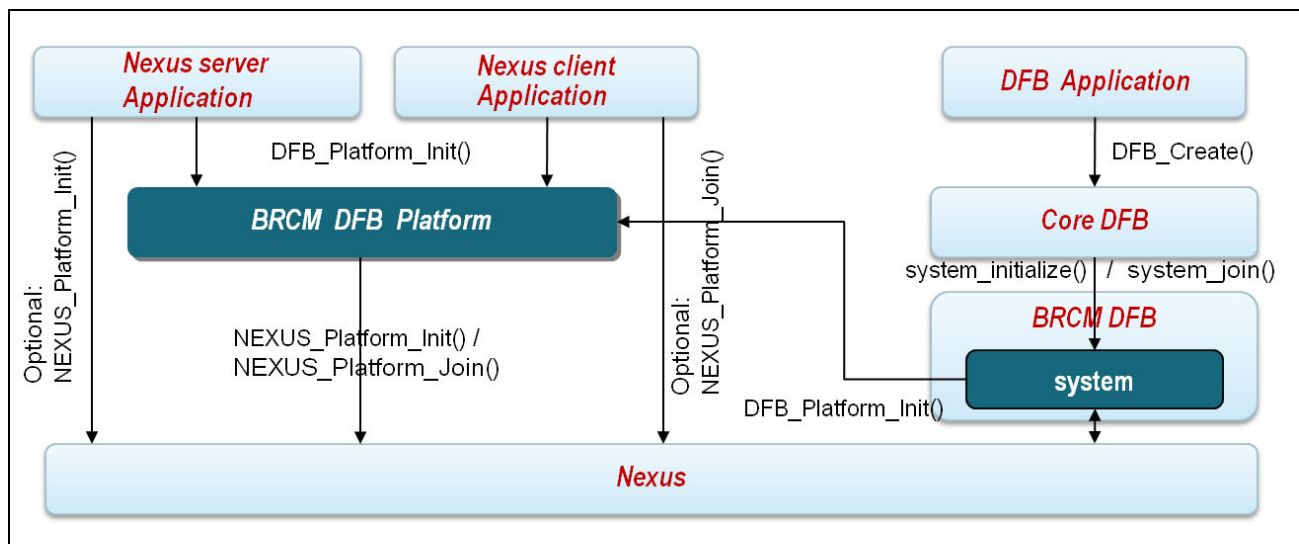


Platform Library Usage for Standard DirectFB

Broadcom provides a DirectFB platform software library that is used to help initialize/join Nexus (and its required modules) and to support sharing of resources between DirectFB and non-DirectFB applications. This platform library has its own lightweight API that is not part of the original DirectFB API. DirectFB applications can run without modification, but non-DirectFB applications (for example, Nexus applications) must use this API to keep Nexus and DirectFB synchronized with each other.

Non-DirectFB applications must link against this shared library (libinit.so or libinit_client.so) and must call `DFB_Platform_GetDefaultSettings()`, followed by `DFB_Platform_Init()`. DirectFB applications themselves need not be aware of this platform library, as the Broadcom system driver will automatically call these functions during the system initialization stage. [Figure 5](#) helps to illustrate these concepts.

Figure 5: Platform Library Usage



Platform Library Usage for DirectFB-XS

The Broadcom DirectFB platform software library is also used when DirectFB is built to use the Nexus surface compositor, also known as DirectFB-XS in Broadcom terminology. In this case, the platform library is used to join Nexus, as all DirectFB applications must be run as Nexus client applications. The platform library is also used to open certain Nexus modules (for example, graphics2d, graphics3d, picture decoder), but the handle for each of these modules is not shared between different processes as they are for standard DirectFB. DirectFB-XS also does not open the Nexus display, but instead acquires an instance of a Nexus surface compositor client interface. This interface is used to provide a virtual frame buffer for DirectFB-XS to render in to. The platform library also does not open any Nexus display windows or outputs. These interfaces are opened and configured in the Nexus server application.

Standard DirectFB applications can run unmodified when DirectFB-XS is built. These applications will all run as Nexus clients. Non-DirectFB applications no longer need to call the DirectFB platform layer API, as no sharing of handles or resources is permitted (or enabled). The DirectFB platform layer adds no additional functionality.

Graphics Driver

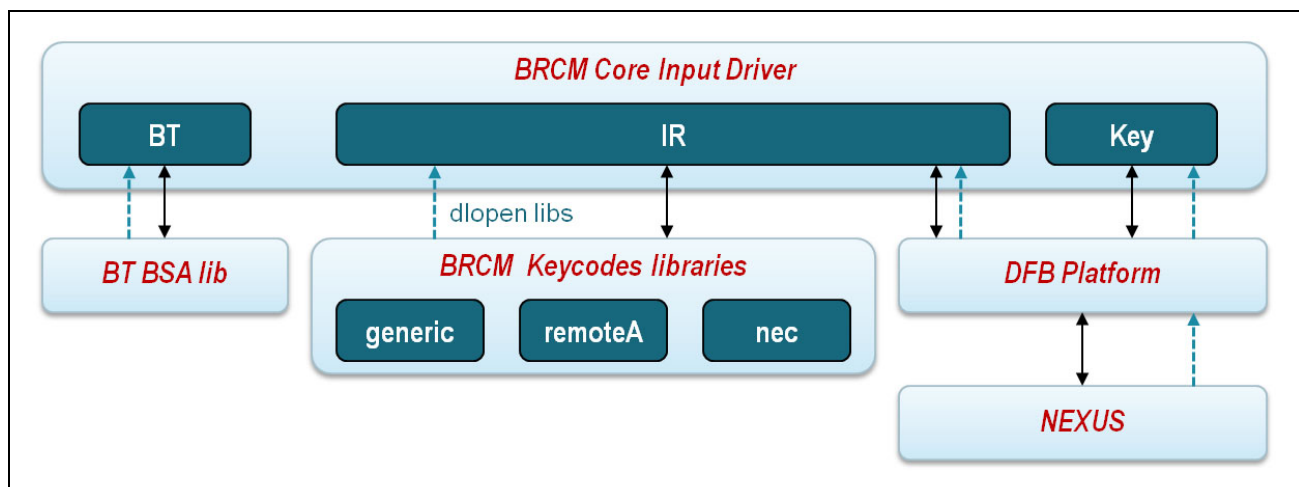
Broadcom provides a highly optimized 2D and 3D graphics driver that accelerates most drawing and blitting operations. This graphics driver accelerates 2D operations when the underlying hardware is using the M2MC core (blitter). When using this core, the driver defaults to using a packet buffer-based API to gain maximum performance. The legacy NEXUS_Graphics2D_xxx() API can still be enabled, but the graphics performance will not be as high as for the packet buffer implementation.

For chips that have a VC-4 graphics core, such as the BCM7425 or BCM7435, 3D transformations like texture mapping, triangle fills, rotation of primitives, etc., are hardware-accelerated. For chips that do not have this 3D graphics card, software fallback operations are provided. For chips that have the VC-4 graphics core, customers are encouraged to use the OpenGL ES 2.0 or OpenGL ES 1.1 APIs to provide these types of operations.

IR and Front Panel Drivers

Broadcom provides infrared (IR), and keypad (front-panel) drivers to allow interaction with DirectFB applications. These input device drivers can be optionally compiled in during the build process. [Figure 6](#) helps to illustrate the software architecture.

Figure 6: IR and Front Panel Drivers Software Architecture



The IR input driver allows run-time selection of the IR protocol and IR keycodes mapping file. The keycodes mapping file is used to convert from native IR command codes to DirectFB input event codes. The default IR protocol and keycodes mapping file are defined in the build system, but the user can override this default at compile time and/or run time. This means that it is possible to change the protocol or keycodes mapping file used at run time without having to recompile DirectFB.

The run-time selection is available with the following DirectFB config options:

```
bcmnexus-ir-protocol  
bcmnexus-ir-keycodes
```

By default, the “CirNec” keycodes and “CirNec” IR protocol are used; these support the Broadcom silver handset. Previous DirectFB releases supported the RemoteA One-For-All handset or black slim handset by default.

If the user prefers to use this older remote control handset to control STBs, then the RemoteA IR protocol and keycodes file can be specified at run-time as follows:

```
./rundfb.sh df_input --dfb:bcmnexus-ir-protocol=RemoteA,bcmnexus-ir-keycodes=RemoteA
```

If DirectFB-XS is being used, then the run time option cannot be used. In this case, keyboard and IR protocols can be specified in the `directfb_common.inc` file found in the `opensource/directfb/build` directory or as a compile-time option (see below).

The choice of the IR protocol name can be found in the `DirectFB-Broadcom/inputdrivers/bcmnexus/core/bcmnexus_ir_inputmode.h` header file. This is an autogenerated header file that extracts the name of the input modes from Nexus. The choice of IR keycode mapping file can be found in what keycodes modules are built and present in the `/usr/local/lib/directfb-version/inputdrivers/bcmnexus` target directory. For example, if `libdirectfb_bcmnexus_ir_keycodes_cirnec.so` is present, then “cirnec” can be specified as the keycodes mapping file at run time.

If the user wishes to support a different IR protocol or handset, then a new keycodes mapping file will have to be created and added to the DirectFB build system. If the user would like to override the default IR protocol and keycodes file at build time, then the following environment variables can be set to override the defaults:

```
DIRECTFB_IR_PROTOCOL=xxxxx  
DIRECTFB_IR_KEYCODES=yyyyy
```

The front panel and IR receiver drivers have been designed to mimic the behavior of a keyboard by default. This means that if the user presses a key, a single `DIET_KEYPRESSED` event is generated and when the key is released a single `DIET_KEYRELEASED` event is generated. If the key is held down for longer than the “skip” count, single `DIET_KEYPRESSED` events are generated with the `DIET_REPEAT` flag set.

If the user would like to revert to using the original mechanism, whereby both `DIET_KEYPRESSED` and `DIET_KEYRELEASED` events are generated together, then the following runtime DirectFB options can be specified in the `directfbrc` file (or set in the `DFBARGS` envvar):

```
bcmnexus-ir-timeout=0  
bcmnexus-key-timeout=0
```

The list of all IR and KEYPAD options can be found at run time by using the help option. For example:

```
./rundfb.sh <app> --dfb-help
```

The IR repeat filter time can be specified with the following run-time option:

```
bcmnexus-ir-repeat-time=xxx
```

DirectFB Nexus Input Router

DirectFB 1.4.17 version 1.1 and higher features the Nexus Input Router (NIR) as an additional input driver. NIR allows a server process to capture various input events like IR, keyboard, and mouse and send them via IPC to client applications. This allows multiple clients to share system inputs without contention over the actual hardware devices.

This feature can be used when DirectFB Nexus Surface Compositor (NSC) support is enabled. To enable DirectFB NIR support, the following build flags need to be set.

```
DIRECTFB_NSC_SUPPORT=y
DIRECTFB_NIR_SUPPORT=y
```



Note: If Nexus reference software supports nxClient API, then `DIRECTFB_NIR_SUPPORT=y` is set automatically when `DIRECTFB_NSC_SUPPORT=y` is set during DirectFB build. For mouse and keyboard events, DirectFB NIR support is not available. DirectFB uses open-source `linux_input` driver to handle mouse and keyboard input events.

Enabling DirectFB NIR support suppresses run-time loading of the `linux_input` input driver by adding an entry during build in the `directfbrc` file as “`disable-module=linux_input.`” This is done so as not to have input events being sent by both `linux_input` and Nexus Input.

The DirectFB NIR feature requires the Nexus server input router application running in the background. The server NIR application captures various input events and then sends them via IPC to client applications. The client NIR application then registers for desired input events and receives them via IPC using Nexus APIs. DirectFB NIR client-side code can be found in `platform_nexus_input_router.c` in the DirectFB Broadcom package in the `platform` directory.

For reference software releases before URSR 13.1 DirectFB provides a sample NIR server called “`nxsmaster.`” For v13.1 and later reference software releases, the Nexus “`nxserver.`” application is used.

To test the DirectFB NIR input driver, the DirectFB example application `df_input.c` can be used. Start the Nexus server application `nxserver` (or `nxsmaster` for older systems) first:

```
ifconfig (Note the IP address of the box.)
./rundfb.sh nxserver
```

Then launch the `df_input` DirectFB example application from a separate window/terminal like this:

```
./rundfb.sh join df_input
```

For more details on the Nexus Input Router server and client applications, refer to the Nexus applications `input_router.c` and `input_client.c` in `Nexus/examples/multiprocess`. Details on how to build these applications can be found in [Reference \[3\] on page 11](#) (*Nexus_Usage.pdf* in the Reference Software Release).

Nexus nxClient support is available from reference software URSR 13.1 onwards. The Nexus server application “`nxserver.`” is part of the Nexus reference software and resides at `nexus/nxclient/server` directory. For more details on nxClient, please refer to the *NxClient.pdf* from the directory `nexus/nxclient`.



Note: The Nexus server application “nxserver” does not currently support keyboard and mouse events. Only IR remote input are supported. The “nxsmaster” application supports all three types of input. The “nxsmaster” server application is only available with older RefSW releases. From URSR Phase 13.1 onwards, nxserver is available. For builds using the “nxserver” application, the linux-input module is used to receive keyboard and mouse events.

DirectFB Google Test Framework

This release of DirectFB also features DirectFB unit tests which can be automated using the Google Test framework. When DirectFB is built with the flag `BUILD_DIRECTFB_UNITTEST` set to `y`, an extra executable “df_unittest” will be built. df_unittest can be launched in the same way as any other DirectFB test application with the following commands:

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh install
./rundfb.sh df_unittest
```

This will start the individual DirectFB unit tests automatically. The console log shows the test results. After all the tests have been completed, the final result showing number of tests passed is also shown on the console.

Currently, the following unit tests are covered by the df_unittest program:

- df_andi
- df_dok
- df_brcmTest
- dfbtest_blit
- dfbtest_layer
- Insignia tests (Not available to customers without an applicable SLA.)

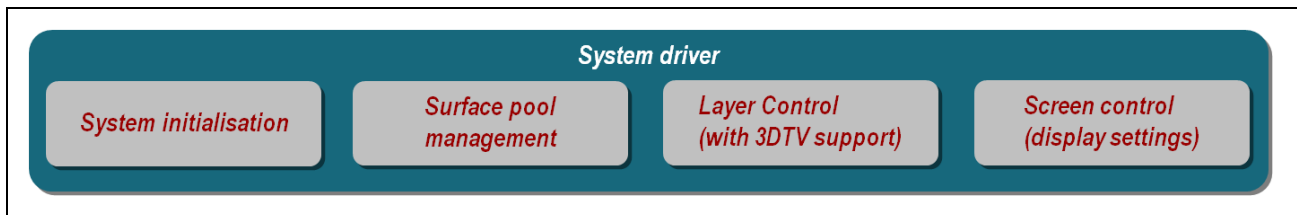
The Google Test framework implementation and associated unit tests reside in the directory `AppLibs/opensource/directfb/unittest`.

System Driver

Broadcom provides what is known as a system driver (shared library) to help initialize the system, control the layer(s) (frame buffers), manage memory, and surface buffers, to set up the display pipeline (such as setting correct video format), and to allow different outputs to be added or removed from a given screen/display.

Figure 7 depicts these key responsibilities.

Figure 7: Key System Driver Responsibilities



The system driver is multiprocess-aware and has the responsibility to marshal memory allocation/deallocation requests, surface buffer creation/destruction, setting of the graphics frame buffer(s), and setting of display/screen settings (for standard DirectFB mode) from client/slave DirectFB applications to the master DirectFB application using fusion IPC. The system driver also parses all Broadcom-specific application options passed in from the command line, DFBARGS envvar, or directfbrc file.

For standard DirectFB (non-DirectFB-XS) builds, Broadcom supports the DirectFB Screen Encoder API and Mixer API to allow a master DirectFB application to set up the video output resolution, frequency, scan-mode, background color, etc. The default start-up video output resolution is specified in the `/usr/local/etc/directfbrc` file with the “`res=`” run-time option. The default is 720p/60 Hz, but the user can override this in a number of ways:

- Modify the `directfbrc` file.
- Specify `res=xxx` on the command line; for example:
`./rundfb.sh df_andi --dfb:res=720p50).`
- Specify the `res=xxx` option in the DFBARGS envvar; for example:
`export DFBARGS="res=720p50").`



Note: The complete list of resolutions is available if you type:

```
./rundfb.sh df_andi --dfb:help
```

For standard DirectFB mode, the layer-handling code supports both mono and stereo surface layer buffers. The latter is required when 3DTV stereoscopic graphics are enabled. For chipsets that don’t have 3DTV-capable video processing, the left/right buffers are provided to the display in “top/bottom half” or “left/right half” orientation. For newer STB chipsets that have 3DTV-capable video processing, the left/right buffers can be provided to the display hardware in additional formats like frame-packed. There are new 3DTV DirectFB APIs that Broadcom has provided to `directfb.org` and have been incorporated into the DirectFB-1.5.x series. Broadcom has back-ported these changes to DirectFB-1.4.17. Refer to DirectFB-1.6.x APIs on the DirectFB website (http://directfb.org/docs/DirectFB_Reference_1_6/index.html) for more information. 3DTV DirectFB APIs are available in this DirectFB release.

ImageProvider Driver

Broadcom provides a hardware-accelerated ImageProvider based on the Nexus Picture Decoder. This can decode GIF, JPEG, and PNG images using the underlying Still Image Decoder (SID) hardware.

The ImageProvider() driver for Nexus uses the “Picture Decoder” API to decode and render an image into a temporary buffer before being blit (or stretch blit) to the final destination surface. This last blitting operation is done using the internal DirectFB APIs and does not directly use the Nexus graphics2d or packet-buffer API directly. This final blit also provides any color format conversion between the typical YUV output from the hardware and the final destination surface’s format.

If the image cannot be decoded and rendered (for example, SID hardware doesn’t support the image), then the software image providers will render the image to the final destination surface.

The ImageProvider driver for Nexus now supports segmented and streaming decoding. This is useful when trying to decode and display large sized images that would otherwise require large amounts of memory. The segmented image decode using hardware Still Image Decoder is available only for the non-multiscan mode jpeg images. For other image formats Still Image Decoder is used without using the segmented decode method.

Additions to the Stock DirectFB

Broadcom has made additions to the input drivers and improvements to the DirectFB build systems as listed in this section. For more details on features and improvements in Core open-source DirectFB refer to the NEWS and ChangeLog files in the corresponding download directories on directfb.org.

Broadcom Specific DirectFB Unit Tests

The following additional test applications are provided:

- `df_brcmTest`—Graphics conformance test comparing hardware vs. software blits and fill operations.
- `df_window_prealloc`—Used to test system and video memory pre-allocated surfaces.
- `decode_server_dfb`—This is a Nexus server/DirectFB master-based application available for standard DirectFB builds that allows a video/audio playback client to be attached.
- `decode_client_dfb`—This is a Nexus client-based application that allows video/audio content to be played back from a transport stream file. It connects to the already running `decode_server_dfb` application and is only available for standard DirectFB builds.
- `playback_dfb`—This is a Nexus master/DFB master based video/audio playback application (standalone) that is only available for standard non-XS mode DirectFB builds.
- `dfb_input_hotplug`—Application used to support hot-plugging of USB keyboard/mice input devices. The application is copied into the `/usr/local/etc/hotplug.d` directory during DirectFB build.
- `nxsmaster`—This is a Nexus master server application only available for DirectFB-XS master library builds. It is the first application that should be launched before any further DirectFB-XS or non-DirectFB-XS applications are started. This application is not built with Nexus reference software 13.1 onwards. Use the “nxserver” application provided with Nexus instead.

- `df_screen_encoder`—This example shows how to connect and disconnect various output connectors from the primary and secondary displays.
- `nxserver`—This is a Nexus master server application only available for DirectFB-XS master library builds. It is the first application that should be launched before any further DirectFB-XS or non-DirectFB applications are started. The application is only available when using Nexus reference software 13.1 onwards. Use the “`nxsmaster`” server application for older Nexus reference software releases.

Build System

- Many compiler warnings have been removed from the code.
- The build system now uses “silent make” and shows a “CC” or “CCLD” when a file is being compiled or linked rather than the verbose compilation information. This makes it much easier to spot errors or warnings.
- Added the ability to link DirectFB with external shared libraries (for example, `zlib`, `libpng`, `libjpeg`, `freetype`) rather than statically compile them in (saves space).
- Added support for setting the vendor version string as part of the DirectFB version information.

Input Devices

Support hot-plugging of USB keyboard/mice properly (hot-plug thread now exits).

Section 8: Testing DirectFB

Testing the IR Input

The current DirectFB IR driver supports the Broadcom silver remote control by default (NEC protocol).

The best application to test the IR remote control is `df_input`. You can run this test application by entering the following command:

```
cd /usr/local/bin/directfb/1.7
./rundfb.sh df_input
```

Then you can press any key on the handset and you should see the name of the key and key code displayed on the display. If the key is held down on the remote control, the repeat event should be set.

Testing Different Blitting and Drawing Modes

There is a specific test called `df_brcmTest` that runs through many different blitting/blending and drawing operations with the hardware acceleration output displayed in a window on the left-hand side the screen and the software fallback mechanism displayed in its own window on the right-hand side the screen (side-by-side for comparison). The user simply needs to press the **<OK>** or **<SELECT>** key on the IR handset to progress through the different tests.

The test also accepts setting the `blittingflags` and `drawingflags` environment variables to test additional blitting and drawing scenarios (such as Destination color keying). For example, to test source color keying on all blitting/blending test cases, you need to set the `blittingflags` environment variable as follows (prior to running the test):

```
export blittingflags=0x08
```

To test destination color keying for all blitting test cases, you need to set the environment variable as follows:

```
export blittingflags=0x10
```



Note: These values are determined by looking at the `DirectFB-version/include/directfb.h` header file and reviewing the `DFBSurfaceBlittingFlags` typedef.

The test defaults to an ARGB pixel format for the graphics layer/plane, but any valid pixel format can be set by modifying the `df_brcmTest.c` file in the `DirectFB-version/tools` directory and setting the following define to the required format:

```
#define PRIMARY_PIXELFORMAT  DSPF_XXXX
```

where XXXX is one of the pixel formats as defined in `DirectFB-version/include/directfb.h`.

Performance Tests

Both `df_andi` (Penguins) and `df_dok` are good benchmarking tests to check the overall graphics performance of the target platform.

The `df_andi` test measures real-world blitting performance by seeing how many penguin blits can be sustained at a given number of frames per second. The graphics performance is proportional to the number of penguins and fps (frames per second) displayed at the top left-hand corner of the screen. The number of penguins can be increased by pressing the `<S>` key on the target platform's USB keyboard. Also, the number of penguins can be decreased by pressing the `<D>` key on the keyboard. Pressing the `<SPACE>` bar will cause the penguins to form a logo and pressing `<R>` will cause them to revert to moving around the screen. Pressing `<P>` will power up/down the screen. Pressing `<O>` will cause the output resolution to change. Pressing `<M>` will toggle mirroring of the primary graphics frame-buffer to the secondary display. Pressing `<Q>` will quit the application.

The `df_dok` test is a true benchmarking test that measures CPU load and graphics blitting/drawing performance. It shows the CPU load in square brackets along with a print out of the number of graphics operations per second of each test (for example, Mpixels/s, Kchars/s).



Note:

- There will be a significant difference in the CPU load when running DirectFB in release mode vs. debug mode. To obtain the best results, always build DirectFB with `"B_REFSW_DEBUG=n"` (release mode).
- Building the Nexus and magnum drivers in release mode (`B_REFSW_DEBUG=n`) will also provide a significant increase in some hardware-accelerated operations.

Supported Platforms

[Table 7](#) lists the set-top box platforms that this release of DirectFB supports.

Table 7: Supported Platforms

Platform	Comments
BCM97145	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97231	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97241	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97358	No OpenGL 3D support/Stereoscopic hardware support
BCM97425	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97429	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97435	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support
BCM97445	OpenGL ES2.0 support (VC-4)/Stereoscopic hardware support

Section 9: Frequently Asked Questions

How Do I Enable Debugging on a Per Module Basis in DirectFB?

Enabling debugging for module_identifier can be done in the following three different ways:

- On the target platform export DFBARGS as follows:
 - export DFBARGS="debug=module_identifier,debug=module2_identifier"
- Edit /usr/local/etc/directfbrc and add the lines:
 - debug=module_identifier debug=module2_identifier ...
- Pass the debug options on the command line:
 - ./rundfb.sh df_dok --dfb:debug=module_identifier,debug=module2_identifier

For the DirectFB platform layer, the Nexus/Magnum debug system is used. The exact syntax used varies, depending on whether kernel- or user-space Nexus drivers are used.

An example of how to set some variables in Linux user mode is as follows:

```
export msg_modules=platform_nexus_init
```

In the kernel mode configuration, variables can be passed in using the configuration environment variable, using the following syntax:

```
export config="msg_modules=platform_nexus_init"
```

The config variable should then be passed to the nexus module when it is being inserted. For example:

```
insmod nexus.ko config=$config
```

How Do I Enable Back-Tracing in DirectFB?

One other useful debug tool is to enable a back-trace whenever DirectFB receives a signal. Add the option TRACE=y to your make command. This is also the option that should be set when using GDB to help debug DirectFB issues.

How Can I Disable Hardware Acceleration and Use the Generic DirectFB Software Graphics Functions Instead?

Users can pass the no-hardware option to DirectFB like this.

Example:

```
export DFBARGS="no-hardware"
```

This is useful if you are unsure whether the hardware-accelerated version is doing the correct graphics operation, or if you want to measure the performance with and without hardware acceleration.

How Can I Tell What Size Surfaces Are Being Created? Why Can't I See Memory for my Surface Being Allocated on Creation?

DirectFB implements a lazy surface allocation scheme. The Broadcom DirectFB system driver uses NEXUS_Surface_Create() to create a buffer that is associated with each DirectFB surface. This buffer is normally only allocated at the very first Lock() call on the surface. This Lock() call can be made either by the CPU explicitly or internally by the GPU/blitter. When a surface is first locked, a buffer will be allocated for it and the code to do this is in bcmnexus_pool.c. To see the size of the buffer (Nexus surface), you can enable debugging for the bcmNexus/Pool module identifier.

Example:

```
export DFBARGS="debug=bcmNexus/Pool"
```

Blending Multiple Windows Together Does Not Look Right — Why?

Refer to http://directfb.org/wiki/index.php/Blending_HOWTO twiki page to better understand how your application should use Porter-Duff and Blitting flag calls to blend multiple surfaces together.

How Do I Change the Cursor in DirectFB?

The best method is to load in an image and then set the cursor using the layer or window `SetCursorShape()` function.

The sample code below presumes you want to use a PNG file called `cursor.png` from `/usr/local/share/directfb-version/images`, which is 32x32 pixels.

```
IDirectFBDisplayLayer *layer;
IDirectFBSurface      *cursurface;
IDirectFBImageProvider *provider;
DFBSurfaceDescription desc;

DFBCHECK(dfb->CreateImageProvider( dfb,
                                   DATADIR"/cursor.png",
                                   &provider ));

desc.flags = DSDESC_WIDTH | DSDESC_HEIGHT | DSDESC_CAPS;
desc.width = 32;
desc.height = 32;
desc.caps   = DSCAPS_NONE;

DFBCHECK(dfb->CreateSurface( dfb, &desc, &cursurface ) );

DFBCHECK(provider->RenderTo( provider, cursurface, NULL ) );
provider->Release( provider );

DFBCHECK(layer->SetCursorShape(layer,cursurface,0,0));
```

When using the `SaWMan` window manager, you can have different cursors for different windows; see `df_window.c` for an example.

Broadcom® Corporation reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design.

Information furnished by Broadcom Corporation is believed to be accurate and reliable. However, Broadcom Corporation does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.

Connecting
everything®



BROADCOM CORPORATION

5300 California Avenue

Irvine, CA 92617

© 2014 by BROADCOM CORPORATION. All rights reserved.

Phone: 949-926-5000

Fax: 949-926-5203

E-mail: info@broadcom.com

Web: www.broadcom.com