

Getting Started with ADOBE® FLASH® LITE® for the Digital Home



© 2009 Adobe Systems Incorporated. All rights reserved.

Getting Started with Flash® Lite® for the Digital Home

Adobe, the Adobe logo, ActionScript, Flash, and Flash Lite are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

This work is licensed under the Creative Commons Attribution Non-Commercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/us/>

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

Chapter 1: Introducing Adobe Flash Lite for the digital home	
Developer documentation	1
About Flash Lite for the digital home	2
Flash Lite for the digital home platform development	2
Flash Lite for the digital home capabilities	3
System requirements	3
Chapter 2: Installing and building the source distribution	
Development environments	5
Quick start on Linux	5
Creating the source root directory	9
Platform-specific builds	9
Third-party libraries	10
Make utility command-line options	10
Executing unit tests	11
Chapter 3: Working with the stagecraft binary executable	
Running the stagecraft binary from the command line	13
Running multiple Flash Lite instances	14
Using the stagecraft binary command-line options	14
Chapter 4: Using Flash Lite for the digital home in Win32 environments	
Build Flash Lite for the digital home in Visual Studio 2008	21
Run the stagecraft binary in Win32 environments	22
Index	23

Chapter 1: Introducing Adobe Flash Lite for the digital home

This guide introduces C++ and Linux® developers to the Adobe® Flash® Lite® for the digital home platform development kit. It highlights Flash Lite for the digital home features. It also describes how to install and build the source distribution in Linux and Win32 environments, and how to perform tasks that are typical to the Flash Lite for the digital home platform development cycle.

Developer documentation

The following types of developers use Flash Lite for the digital home documentation:

System developers These developers customize Flash Lite for the digital home for their platform. They also provide a C++ application that loads and runs Flash Lite for the digital home. They are experienced C++/Linux developers.

Platform driver developers These developers provide optimized modules for Flash Lite for the digital home. These modules replace provided software implementations with implementations that utilize a platform's hardware for graphics, video, audio, and images. They are experienced C++/Linux developers.

SWF content developers These developers create SWF content that Flash Lite for the digital home runs.

The following documentation describes the different Flash Lite for the digital home documents:

Document	Audience	Purpose
<i>Getting Started with Adobe Flash Lite for the Digital Home</i>	System developers and platform driver developers	An introduction to Flash Lite for the digital home features, and to installing, building, and testing Flash Lite for the digital home on your platform
<i>Optimizing Adobe Flash Lite for the Digital Home</i>	Platform driver developers	<ul style="list-style-type: none"> Describes how to build platform-specific modules, including how to use the C++ APIs. Describes the process for building Flash Lite for the digital home with platform-specific modules
C++ API Doxygen reference provided with the source distribution	System developers and platform driver developers	Details the classes and methods of the C++ APIs, including return values and parameters. The Doxygen reference is generated from the code files.
<i>Developing Applications for Adobe Flash Lite for the digital home</i>	SWF content developers	Describes the ways in which SWF content development for Flash Lite for the digital home differs from development for Adobe® Flash® Lite® 3.1.

**Got Feedback?
Click Here**

About Flash Lite for the digital home

Adobe® Flash® Lite® for the digital home is Adobe® Flash® Lite® 3.1 optimized for hardware and software architectures of digital home electronics. These devices include, for example, television sets, Blu-ray players, game consoles, and set-top boxes. Adobe® Flash® developers can create applications for Flash Lite for the digital home that stream and play high-definition video from the Internet. These developers can also create rich Internet applications and graphical user interfaces for Flash Lite for the digital home.

- **Delivery and playback of HD video from the Internet**

Flash Lite for the digital home enables streaming video content from the Internet directly to TV sets and other Flash Lite for the digital home platforms, without the use of a web browser. Flash Lite for the digital home supports 1080p high definition video. The videos can use H.264, Sorenson H.263, and On2 VP6 codecs. Streaming can be from an HTTP server or from an Adobe® Flash® Media Server.

- **Rich Internet applications (RIAs)**

Flash Lite for the digital home is a vehicle for providing RIAs for use on Internet-connected home electronics devices. Flash Lite for the digital home enables the use of SWF movies as the interface to web services. These SWF movies are not browser-based.

- **High-performance graphical user interfaces**

The static, minimalistic user interfaces that are the current standard for most home-entertainment devices and programming sources are limited compared to the dynamic, feature-rich user experiences that Flash Lite for the digital home supports. Flash Lite for the digital home utilizes a platform's graphical hardware accelerators to scale user interfaces to provide a 10-foot user experience.

Flash Lite for the digital home platform development

Flash Lite for the digital home requires minimal effort to recompile on any Linux distribution that is based on version 2.6.x of the Linux kernel. Flash Lite for the digital home can be recompiled for most target platforms in a single working day, without requiring extensive porting of C++ code or knowledge of Flash Lite.

Flash Lite for the digital home provides a streamlined C++ Application Programming Interface (API) that does not require knowledge of Flash Lite details. You can develop and build a Flash Lite for the digital home platform for a target Linux operating system entirely in a Linux environment. You can also use the C++ debugging tools in Microsoft Visual Studio 2008 Professional on a Win32 environment to assist your development efforts.

The highly modular design of Flash Lite for the digital home facilitates the replacement of any module with a minimum of effort. To utilize hardware capabilities that are unique to your target platform, you replace a module provided by Flash Lite for the digital home with one that you create. For example, Flash Lite for the digital home provides a module that processes streaming video in software. You can replace this module with one you provide that accesses your platform's hardware accelerators for processing streaming video.

Flash Lite for the digital home capabilities

Flash Lite for the digital home is based on Flash Lite 3.1, but also provides the following capabilities:

- SWF content appears in an area of the screen known as the *Stage*. Flash Lite for the digital home provides a simple API to enable simultaneous playback of multiple SWF movies. Each SWF movie appears in its own Stage. Although FL3.1 also allows clients to run multiple simultaneous SWF movies, its interface is not optimized for this use case.
- The H.264 support in Flash Lite for the digital home provides full 1080p HD video. Additionally, Flash Lite for the digital home provides 720p 30-fps SWF content rendering. Flash Lite for the digital home supports SWF-embedded video, video content delivered with the `file://` or `http://` protocols, and video content from Adobe Flash Media Server.
- Flash Lite for the digital home is tailored for the system-on-a-chip (SOC) that controls an HD TV. Flash Lite for the digital home models its graphics and video planes in a way that is a natural fit with HD video. Flash Lite for the digital home supports the Sorenson, On2 VP6, and H.264 video codecs. It uses fixed-point math because many SOCs don't provide floating-point units. It supports vector-based graphics, but SWF content developers are encouraged to use bitmap graphics whenever possible for best performance on lower-end processors.
- You can create your own *driver* that enables your Flash Lite for the digital home platform to interact with hardware-based functionality of your target platform. Typically, you create drivers to replace functionality that Flash Lite for the digital home provides in software with a hardware-based equivalent. For example, to use hardware accelerators for decompressing and presenting video, you create a driver. The driver enables your Flash Lite for the digital home platform to interact with specific video-processing hardware.
- To supply custom functionality to SWF content developers, you can create an *extension*. An extension exposes your own C++ code as new Adobe® ActionScript® 2.0 classes that are built in your target platform. ActionScript 2.0 is the scripting language that SWF content developers use to create rich media content for Flash Lite for the digital home. ActionScript 2.0 is similar to JavaScript or ECMAScript. It is a typed, object-oriented language that allows creation of user data types (classes), implements data-hiding, and provides multiple-inheritance mechanisms.

A SWF content developer writes ActionScript code in a FLA file or a separate ActionScript .as file. The developer publishes the SWF content as an executable SWF file. Before you create an extension, be sure that a SWF content developer can't already implement your custom functionality in ActionScript.

System requirements

The system that runs your product that uses Flash Lite for the digital home is the *target system*. For example, a target system is the Linux-based system-on-a-chip that controls a digital TV. The system you use to write C++ code and build an executable file is the *development system*.

Target system requirements

The consumer electronics device that runs your Flash Lite for the digital home platform must meet these minimum hardware requirements:

Component	Minimum Requirement
Processor	300 mHz or equivalent dedicated to Flash operation
Screen	1280 x 720 ARGB8888 graphics plane with minimum 30-fps hardware compositing
Graphics	2-D hardware graphics engine that performs blit and fill graphics operations

Component	Minimum Requirement
Memory	128-MB RAM that makes 32 MB available to each application instance
Storage	32-MB persistent read-only local storage (ROM, flash RAM, or hard disk) for libraries and code 4 MB persistent read/write local storage (flash RAM or hard disk) for shared objects
Video	H.264 hardware decoder
Audio	Advanced Audio Coding (AAC) hardware decoder or software equivalent
Connectivity	Internet connection (100BaseT Ethernet, 802.11 wireless, or other connection)
Remote	Traditional directional or free-space

The consumer electronics device that runs your Flash Lite for the digital home platform must meet these minimum software requirements:

- Linux operating system running a version 2.6-based kernel
- `glibc` or `uclibc` runtime libraries

Development system requirements

The system you use to develop and build your Flash Lite for the digital home platform must meet these minimum software requirements:

- Linux operating system running a version 2.6-based kernel
The examples in this document are based on use of the Ubuntu 8.04 LTS Desktop distribution.
- `glibc` or `uclibc` runtime libraries
- GNU compiler collection, `binutils`, and the `make` utility

You can develop Flash Lite for the digital home platforms entirely in a Linux environment. You can also use Microsoft Visual Studio 2008 Professional to develop and debug your C++ code in a Win32 environment. You cannot build a non-debug version of a Flash Lite for the digital home platform in a Win32 environment.

Chapter 2: Installing and building the source distribution

You can develop Adobe® Flash® Lite® for the digital home platforms in Linux® and Windows® development environments. In Linux environments, you install the source distribution in a directory you choose. Then you build Flash Lite for the digital home with the make utility. The make utility uses Linux environment variables and command-line options to determine how to build Flash Lite for the digital home.

Development environments

You can develop *and* deploy Flash Lite for the digital home platforms on any Linux distribution that is based on a version 2.6 kernel. Adobe uses the bash shell to build and test Flash Lite for the digital home.

You can also develop Flash Lite for the digital home platforms on a Win32 platform. Microsoft Visual Studio 2008 is the only Win32 C++ environment that the source distribution supports. However, Flash Lite for the digital home supports this environment for testing and debugging purposes only. You cannot build a production version of a Flash Lite for the digital home platform in the Win32 environment.

In the early phases of development, you can run your Flash Lite for the digital home platform on the same Linux machine you use to develop it. Or, you can do all your development and debugging on a Windows® computer. Eventually, though, you'll need to test release builds of your platform on a Linux target that approximates your production deployment as closely as possible. This target is likely to be a stripped-down runtime environment rather than a full-featured Linux distribution. You use a full-featured Linux distribution to build your Flash Lite for the digital home platform. Therefore, make sure that you can easily transfer binary executables from the build machine to the production target machine. Your solution can be as simple as a USB drive that can connect to either machine. Alternatively, your solution can be as elaborate as a shared network file system. Plan ahead for the eventual need to transfer executables between distinct environments for building and testing your platform.

See also

[“Creating the source root directory”](#) on page 9

[“Using Flash Lite for the digital home in Win32 environments”](#) on page 21

Quick start on Linux

Getting started with Flash Lite for the digital home on an x86 desktop Linux development system consists of the following tasks:

- [“Install the source distribution”](#) on page 6
- [“Build on the Linux command line”](#) on page 6
- [“Test the build for linkage errors”](#) on page 7
- [“Display a SWF file”](#) on page 8

The example command-line instructions use the Bash shell.

**Got Feedback?
Click Here**

Install the source distribution

To install the source distribution, take the following steps:

- 1 Copy the source distribution .tgz file to your Linux development machine. Copy it to the directory into which you want to install the source distribution. For suggestions on choosing your source directory, see “[Creating the source root directory](#)” on page 9.
- 2 Change directories to the directory that contains the Flash Lite for the digital home .tgz file. In this example, the .tgz file is called stagecraft-source-distribution.tgz. Typically, the filename includes version information about the source distribution.
- 3 Untar the Flash Lite for the digital home source distribution.

```
tar xfz stagecraft-source-distribution.tgz
```

The tar command creates a subdirectory in the working directory. The name of the subdirectory depends on the .tgz file. However, as this example continues, it uses a subdirectory named stagecraft. The new stagecraft subdirectory contains the subdirectories and files that make up the source distribution.

- 4 If your distribution includes a thirdparty-private .tgz file, copy it to the same directory as the source distribution .tgz file. Use the tar command to untar the file.

Your next step is to build the source distribution.

Note: Do not modify the source distribution until you have built and run it successfully on at least one of the provided platforms.

Build on the Linux command line

To build the unpacked Flash Lite for the digital home sources for an x86 desktop platform such as Ubuntu, take the following steps:

- 1 Change directories to the stagecraft/build/linux/ directory.

Note: The stagecraft/build/ directory also contains a win32/ directory that is on the same level as the linux/ directory. For more information about building Win32 targets, see “[Build Flash Lite for the digital home in Visual Studio 2008](#)” on page 21.

- 2 Install the X11 and the ALSA development packages, available as shareware. For example, execute the following commands to install these development packages in Ubuntu:

```
## install X11
apt-get install libx11-dev
## install ALSA
apt-get install libasound2-dev
```

The apt-get command uses the packaging system for Debian distributions. Use the packaging system for your Linux distribution to install these development packages.

For information on building with other third-party packages, see “[Third-party libraries](#)” on page 10.

- 3 Execute the make command with the quiet option.

```
make quiet
```

The make utility prompts you to specify the platform to build:

```
Enter a valid SC_PLATFORM from the following choices:
--> x86Desktop <--
```

Because the `SC_PLATFORM` environment variable is not set, the make utility prompts you to enter a value. The prompt list includes the platforms provided with the source distribution, plus platforms you add. For more information, see “[Platform-specific builds](#)” on page 9.

Note: Execute the make utility from the `stagecraft/build/linux` directory only. Executing the make utility from any other directory results in the following message to standard output:

```
make: *** No targets specified and no makefile found. Stop.
```

4 Type `x86Desktop` and then press Enter.

```
Enter a valid SC_PLATFORM from the following choices:
--> x86Desktop <--
--> x86Desktop
```

The make utility prompts you to specify the build mode:

```
Enter a valid SC_BUILD_MODE from the following choices:
--> debug release <--
```

The first time you build Flash Lite for the digital home, create a debug build, as the next step specifies. The only other build mode is `release`, which builds a non-debug version of the specified target. To suppress the make utility prompt for debug or release mode, set the environment variable `SC_BUILD_MODE` to `debug` or `release`.

5 Type `debug` and then press Enter.

```
Enter a valid SC_BUILD_MODE from the following choices:
--> debug release <--
--> debug
```

The make utility reports progress as it compiles, builds, and links each module in the Flash Lite for the digital home distribution.

Test the build for linkage errors

Once you’ve built a debug version of Flash Lite for the digital home without error, execute the following commands. These commands determine whether all Flash Lite for the digital home modules linked properly:

```
## Add the working directory to $LD_LIBRARY_PATH.
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
## Set the working directory to the one that
## holds executable binaries for your platform.
## This cd command assumes that you installed the source distribution
## under your home directory.
cd ~/stagecraft/targets/linux/x86Desktop/debug/bin
## Run the kerneltest program.
./kerneltest
```

After it loads the Flash Lite for the digital home kernel, the `kerneltest` program loads and then unloads each Flash Lite for the digital home module. The `kerneltest` program displays output similar to the following listing:

```

kerneltest: Hello Kernel Test World
Loading Module IIO...                IIO = 0x806eaf0
Loading Module IShell...             IShell = 0x806dc10
Loading Module IStream...            IStream = 0x806ec18
Loading Module IURLOps...            IURLOps = 0x806eaf0
Loading Module IXMLReaderLib...      IXMLReaderLib = 0x806e5f8
Loading Module IFlashLib...          IFlashLib = 0x806e7b8
Loading Module IAudioDecoder...       IAudioDecoder = 0x806f8f8
Loading Module IVideoDecoder...       IVideoDecoder = 0x806eaf0
Loading Module IStreamPlayer...       IStreamPlayer = 0x80706f8
Loading Module IGraphicsDriver...     IGraphicsDriver = 0x8070760
Loading Module IImageDecoder...       IImageDecoder = 0x806f8f8
Loading Module IFL31NativeSoundOutput... IFL31NativeSoundOutput = 0x8072c70
Loading Module IEDKExtensions...      IEDKExtensions = 0x806e660
Loading Module IFileSystem...         IFileSystem = 0x807b580
Loading Module IProcess...            IProcess = 0x807b710
Loading Module ISocket...             ISocket = 0x8072b08
Loading Module IStagecraft...Adobe (R) Flash (R) Lite (R) 3 for the Digital Home - Version
9.10.122.0
(C) 2008-2009. Adobe Systems Incorporated. All rights reserved.
Patent and legal notices: http://www.adobe.com/go/digitalhome\_thirdpartynotice
IStagecraft = 0x8072f98
kerneltest: Goodbye Kernel Test World

```

Each hexadecimal value represents the handle to a module that is loaded into memory. Any module that displays 0x0 as its handle did not load successfully. For each module that does not load, stdout displays a message saying that Flash Lite for the digital home failed to load *moduleName.so*, as the following example output shows:

```

Failed to load libIStream.so:
libIStream.so: cannot open shared object file:
No such file or directory
IStream = 0x0

```

If the kerneltest program runs without error, you're ready to run Flash Lite for the digital home.

Display a SWF file

A C++ application loads and runs Flash Lite for the digital home by calling some of its interfaces. The source distribution provides such a C++ application. The binary executable file is called stagecraft, built by the make utility.

Flash Lite for the digital home displays content that a SWF file provides. This content can be video, audio, animated images, or user-interface controls, such as buttons that run Adobe® ActionScript® functions. To display the 1.swf sample file that the Flash Lite for the digital home distribution provides, take the following steps:

- 1 Add the dot directory (.) to the locations that your LD_LIBRARY_PATH environment variable specifies.

```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

Beyond this example, set LD_LIBRARY_PATH to include the full path of the bin directory for the platform you are working with.

- 2 Change directories to the one containing the stagecraft binary executable. The executable is in stagecraft/targets/linux/x86Desktop/debug/bin.
- 3 Copy the 1.swf sample SWF file to the directory containing the stagecraft binary executable. The 1.swf file is in stagecraft/source/executables/cppunittest/testfiles. Copying the file is a convenience for the next step. You can also use a full or relative pathname for 1.swf in the next step.
- 4 Execute the stagecraft binary, passing the 1.swf file as its argument:

```
./stagecraft 1.swf
```

The stagecraft binary loads and runs Flash Lite for the digital home. It passes the command-line argument `1.swf` to Flash Lite for the digital home to run. The `1.swf` movie displays random images at a high frame rate and displays performance statistics when it completes.

Note: If the value of `LD_LIBRARY_PATH` is incorrect, executing the stagecraft binary results in the following message to standard output:

```
Failed to load libIAEKernel.so: libIAEKernel.so: cannot open shared object file:  
No such file or directory
```

Make sure `LD_LIBRARY_PATH` includes the bin directory for your platform. In this example, the directory is `stagecraft/targets/linux/x86Desktop/debug/bin`.

Creating the source root directory

To begin working with Flash Lite for the digital home, create a writable directory. Make the directory available to all computers that you'll use to write and debug your code. You'll place your local copy of the Flash Lite for the digital home source distribution in this directory.

You can unpack the Flash Lite for the digital home source distribution into any location that provides read/write/execute access and sufficient disk space. For example, you can unpack the distribution into a repository on a source-control system. Alternatively, you can simply place it in your home directory on the file system of your Linux development machine. Your development goals and your corporate IT environment influence the location of your source code base. These factors also influence the location of the local copy of source code that you use for day-to-day development activities. Consider the following characteristics of your environment when determining where to place your source code.

- You can develop some parts of a platform in Windows and some parts in Linux. For example, you can use Visual Studio for debugging but create release builds in Linux. If you plan to use Visual Studio, placing your local sources in a directory that is available to both operating systems can be convenient. However, doing so is not a requirement.
- Embedded systems often provide resources sufficient for runtime execution but not for development purposes. When developing for such devices, you'll need to store executables on a file system that both the development environment and the target device can access.
- If you virtualize either of the Windows or Linux operating systems, placing your local copy of the source in a shared folder that is available to both environments can be convenient. However, doing so is not a requirement.

Platform-specific builds

Flash Lite for the digital home provides a mechanism for building multiple Flash Lite for the digital home platforms from a single source tree. A *platform* is an optimized deployment of Flash Lite for the digital home. The deployment is optimized for a specific combination of operating system, hardware platform, and product feature set. For example, you might implement entry-level and deluxe versions of a product as distinct Flash Lite for the digital home platforms. The deluxe platform might provide hardware acceleration of features that the entry-level platform provides in software. To implement the hardware acceleration, the deluxe platform substitutes its own modules for the software-based modules provided in the source distribution. Implementing the two products as distinct platforms allows them to coexist in the same source tree, sharing modules or providing their own as necessary. You can build either platform by setting the appropriate value for the `SC_PLATFORM` environment variable.

Put your platform-specific source and header files in a directory you create under the `stagecraft/thirdparty-private/<yourCompany>/stagecraft-platforms/<yourPlatform>` directory. For example:

```
stagecraft/thirdparty-private/yourCompanyName/stagecraft-platforms/yourPlatformA/ stagecraft/thirdparty-private/yourCompanyName/stagecraft-platforms/yourPlatformB/
```

For details about building your platform-specific version of Flash Lite for the digital home, see *Optimizing Adobe Flash Lite for the Digital Home*.

Third-party libraries

Flash Lite for the digital home depends on some third-party libraries to build and execute. In some cases, the source distribution provides the third-party library. In other cases, get the third-party library from another source. For example, use `apt-get` to get the ALSA and X11 libraries, as discussed in “[Build on the Linux command line](#)” on page 6.

The openssl library

Flash Lite for the digital home requires the openssl library, version 0.9.8i, for proper `https://` operation. To build the openssl library and have `https://` support enabled, do the following:

- 1 Retrieve the file `openssl-0.9.8i.tar.gz` from the Internet.
- 2 Place the file in the `stagecraft/thirdparty/openssl/` directory.
- 3 Build all modules of Flash Lite for the digital home, by executing the following:

```
make
```

The `make` utility automatically untars the openssl package, builds it, and statically links it into the `IFlashLib` module.

The FFMPEG library

Flash Lite for the digital home uses the FFMPEG library for the `FFMPEGStreamPlayer`. The source distribution provides the `FFMPEGStreamPlayer`. The `FFMPEGStreamPlayer` provides x86Desktop platforms overlay video processing for videos that use H.264 and AAC codecs.

Retrieve the FFMPEG library from the Internet. If you build Flash Lite for the digital home without first installing the FFMPEG library, Flash Lite for the digital home builds a dummy `StreamPlayer`. The dummy `StreamPlayer` uses the `SoftwareStreamPlayer` implementation that the source distribution provides. The `SoftwareStreamPlayer` dumps the audio and video streams to files. For more information on `StreamPlayers`, see *The overlay video driver* in *Optimizing Adobe Flash Lite for the Digital Home*.

Make utility command-line options

The `make` utility writes verbose output by default. This output allows you to observe progress as Flash Lite for the digital home compiles, builds, and links each module in the platform.

Once a platform builds reliably, you can reduce diagnostic output by passing the `quiet` option to the `make` utility:

```
make quiet
```

By default, the make utility builds all modules. It performs incremental builds based on comparisons of the modification dates of source files and their corresponding object files. However, you can also force the make utility to rebuild every component of a platform without reusing any object files generated by previous build. To rebuild everything, pass the clean option to remove the object files, then run the make utility again to build all targets:

```
## Remove old build output files
make clean
## Rebuild everything
make
```

By default, the make utility builds all modules of the platform that the `SC_PLATFORM` environment variable specifies. You can build an individual module by passing the name of the module to the make utility. This value is the name of a module-specific makefile (found in the `stagecraft/build/linux/modules/` directory) without the `.mk` filename extension. For example, the following command makes the `IStreamPlayer` module:

```
## These commands assume you installed the source distribution
## in your home directory.
cd ~/stagecraft/build/linux
## build IStreamPlayer
make IStreamPlayer
```

You can also use the clean option on individual modules. For example:

```
## Clean the stream player module
make clean-IStreamPlayer
```

Note: Always run the make utility from within the `stagecraft/build/linux/` directory, even when building individual modules.

You can also use the rebuild option to force rebuilding. This option first removes all output files (like the option clean), and then rebuilds all the modules. For example:

```
## rebuild everything ('all' is the default target)
make rebuild
## rebuild the stream player module
make rebuild-IStreamPlayer
```

You can pass a list of modules to build. The modules can appear in any order on the command line; the make utility automatically builds them and any dependent objects in the correct dependency order.

```
cd ~/stagecraft/build/linux
make libjpeg IStreamPlayer
```

You can combine options:

```
make quiet rebuild-IAEKernel IShell
```

Executing unit tests

The source distribution provides a suite of unit tests for testing your platform implementation of Flash Lite for the digital home. The suite uses the CppUnit library, which is a C++ framework for test driven development. You can run these tests to validate your platform implementation. However, because the test set is not extensive, passing all the tests does not guarantee your platform implementation is bug free.

You can also use the tests to regression test the functionality of your platform implementation. You can also add your own unit tests.

To build the executable that runs the unit tests, do the following:

```
## In this example, build for the x86Desktop platform.  
## Also, this example assumes that you installed Flash Lite for the digital home in  
## your home directory.  
## The test target builds the CppUnit library and the cppunittest executable.
```

```
make test  
cd ~/stagecraft/targets/linux/x86Desktop/debug/bin  
./cppunittest
```

Running the `cppunittest` executable with no arguments runs all the unit tests. Always run the `cppunittest` executable from the `bin` directory because `cppunittest` depends on the subdirectory `testfiles`. The `testfiles` directory contains files that some of the tests use.

To see the possible arguments for `cppunittest`, do the following:

```
./cppunittest -?
```

This command outputs the following:

```
cppunittest [options] [tests]:  
-l      List the names of all the registered Suites, and exit.  
-r      Repeat all the tests in a continuous loop.  
        (Loop can be terminated by sending a TERM signal to the thread.)  
-v      Verbose output. Prints the name of every test as they run.  
The remainder of the line lists the tests to be run. By default,  
all the tests are run. Tests can be specified by either the name of  
the test suite or the full name of the test.  
eg:  
cppunittest KernelTest -- runs all the KernelTest tests.  
cppunittest KernelTest::testMutex - runs just this test.
```

By default, `cppunittest` outputs only breadcrumbs (“...”) to show progress, followed by a final error report.

Note: *The `cppunittest` executable only builds and executes on Linux platforms. You can not use it on a Win32 platform.*

Chapter 3: Working with the stagecraft binary executable

A C++ application loads and runs Adobe® Flash® Lite® for the digital home by calling some of its interfaces. The source distribution provides such a C++ application. The binary executable file is called `stagecraft`, built by the `make` utility. The source file for the `stagecraft` binary is in `source/executables/stagecraft/stagecraft_main.cpp`.

You can use the `stagecraft` binary for testing your platform-specific drivers. You can also use the `stagecraft` binary as the application your product uses to load and run Flash Lite for the digital home. Depending on your product, you can develop the `stagecraft` binary into a more complex product application that loads and runs Flash Lite for the digital home. The application that loads and runs Flash Lite for the digital home is called the *host application*.

Running the stagecraft binary from the command line

After you have built Flash Lite for the digital home and determined that all its modules linked correctly, you can run it. To do so, execute the following commands on your Linux® command line:

```
## Change to the directory that contains the the stagecraft binary.
## This example assumes that you installed the source distribution
## in your home directory.
cd ~/stagecraft/targets/linux/x86Desktop/debug/bin
## play a specified Flash executable file
./stagecraft pathToSWF/SWFfilename
```

In the preceding command, `pathToSWF` is the path to a SWF file. The `stagecraft` binary loads and runs Flash Lite for the digital home. Then, `stagecraft` passes the information given by the command-line argument to Flash Lite for the digital home.

To try out this command, use the `1.swf` file in the `stagecraft/source/executables/cppunittest/testfiles` directory of the source distribution. For example:

```
## Change to the directory that contains the the stagecraft binary.
## This example assumes that you installed the source distribution
## in your home directory.
cd ~/stagecraft/targets/linux/x86Desktop/debug/bin
./stagecraft ~/stagecraft/source/executables/cppunittest/testfiles/1.swf
```

The `stagecraft` binary continues to run after the SWF content finishes playing. When you close the last window that `stagecraft` uses, the `stagecraft` process exits. Because only one window is associated with this example of using the `stagecraft` binary, closing that window causes the process to exit.

Next, try using a `file://` URL to specify the full path to the SWF movie to play. The `file://` value must specify a valid URL from the root of the local file system. When using the `file://` protocol, keep in mind that it does not interpret characters such as the tilde (`~`) and the double dot (`..`) as Linux file-navigation shortcuts. In the following example, `bob` is the username:

```
## play a SWF file from its original location in bob's home directory tree
./stagecraft file:///home/bob/stagecraft/source/executables/cppunittest/testfiles/1.swf
```

**Got Feedback?
Click Here**

Running multiple Flash Lite instances

Flash Lite for the digital home runs a *Flash Lite instance* to play SWF content. The Flash Lite instance contains Adobe® Flash® Player for Flash® Lite® 3.1. Multiple Flash Lite instances can run concurrently. Each Flash Lite instance runs in its own window, controlled by a *StageWindow instance*. Each Flash Lite instance plays its own SWF content.

To run multiple Flash Lite instances, pass multiple SWF files to the stagecraft binary. For example, the following command line runs two Flash Lite instances, each in its own StageWindow instance, each displaying the `1.swf` movie:

```
./stagecraft 1.swf 1.swf
```

If your desktop windowing system does not tile windows by default, you can use the `--outputrect` option to position them. The following example sets the size of the window of each StageWindow instance to 100 pixels by 100 pixels. The `--outputrect` option also positions the window of the second StageWindow instance 200 pixels below and to the right of the first window:

```
./stagecraft --outputrect 0,0,100,100 1.swf --outputrect 200,200,100,100 1.swf
```

Note: *In this release of Flash Lite for the digital home, X11 windowing systems do not observe window placement specified by the `--outputrect` option.*

For more information about the `--outputrect` option, see “[Display options](#)” on page 15.

Only one StageWindow instance at a time receives user input events and plays sound. This StageWindow instance has the focus. Designating which StageWindow instance has the focus is dependent on your platform implementation. By default, the StageWindow instance corresponding to the SWF file that appears first on the stagecraft command line gets the focus initially. Use a SWF file that plays sound to test how focus works. (The content in `1.swf` does not play sound). For more information about focus, see *Optimizing Adobe Flash Lite for the Digital Home*.

Using the stagecraft binary command-line options

You can use command-line options to change the default behavior of the stagecraft binary and its interface to Flash Lite for the digital home. If you do not specify an option, stagecraft passes default values to Flash Lite for the digital home.

Note: *All options that affect playback of a SWF file must appear before its filename on the command line. You can pass multiple SWF files to stagecraft, specifying different options for each, as long as the options for each SWF file precede its filename.*

Displaying all stagecraft options on the command line

Execute stagecraft with no arguments to display usage information (including options and default values) on the Linux command line:

```
## cd to bin/ directory of the platform target you built.  
## This example is for x86Desktop debug builds  
cd ~/stagecraft/targets/linux/x86Desktop/debug/bin  
./stagecraft
```

The output from the command is the following:

Adobe (R) Flash (R) Lite (R) 3 for the Digital Home - Version 9.10.122.0
(C) 2008-2009. Adobe Systems Incorporated. All rights reserved.
Patent and legal notices: http://www.adobe.com/go/digitalhome_thirdpartynotice

usage: stagecraft [--options] filename.swf [--options] filename.swf]

Up to 4 Flash files may be specified.

Valid filenames are local files (eg filename.swf) or fully qualified URLs (eg <http://myserver.com/myswf.swf>).

Valid options are:

--contentdims w,h
--outputdims w,h
--outputrect x,y,w,h
--bgalpha <0-255>
--keymap filename --extensionfilter <filter-string>
--flashvars <flashvars-string>
--sslclientcerttable <sslclientcerttable-string>
--memlimit <num[K|M]>
--tracefps
--tracefpsfull
--astrace
--noastrace
--nospeedlimit
--quality <low|medium|high>
--security <trusted|sandboxed>

default values are:

--contentdims <swf authored stage dims>
--outputdims <swf authored stage dims>
--bgalpha 255
--noastrace
--quality high
--security trusted

Other options for the stagecraft executable:

--useshell: enables the IShell
--noclearscreen: prevents the screen from being initially cleared

Note: To see the usage information in Win32 environments, run stagecraft from within the Microsoft Visual Studio 2008 development environment. Stagecraft never prints output to the Command Prompt window.

Display options

The following options modify default values that the Flash Lite instance uses to interpret and render SWF content:

- --contentdims *widthPixels,heightPixels*

Width and height (expressed in pixels) of the render plane. The render plane is the bitmap on which the Flash Lite instance renders each frame of Flash animation. By default, the dimensions of the render plane are equal to the Stage size specified when authoring the SWF file.

Depending on the SWF content, specifying a smaller render plane size can reduce system processor usage. For more information, see *Plane dimensions and scaling* in *Optimizing Adobe Flash Lite for the Digital Home*.

For hardware-based scaling, use the --outputdims and --outputrect options instead of the --contentdims option. Hardware scaling is a scaling of the final bitmap: it is faster than the vector-based scaling that the Flash engine performs, but produces poorer quality.

- --outputdims *widthPixels,heightPixels*

Width and height (expressed in pixels) of the output plane. The output plane is the bitmap used to display each completed frame of animation on the display device. On platforms that provide a windowing system, each output plane appears in its own window.

By default, the output plane is the same size as the render plane. You can use the `--outputdims` option to scale output. For more information, see *Plane dimensions and scaling in Optimizing Adobe Flash Lite for the Digital Home*.

Any scaling can produce artifacts. However, the Flash Lite instance minimizes aliasing of lines and colors automatically, and it performs special corrections that maintain sharp edges in scaled-up fonts.

- `--outputrect xPixels,yPixels,widthPixels,heightPixels`

Size and location of the output plane. The `widthPixels`, `heightPixels` values specify the width and height of the output plane in pixels, just as the `--outputdims` option does. By default, the output plane is the same size as the render plane. The `xPixels`, `yPixels` values specify the position of the upper-left corner of the window that displays the output plane.

Specifically, the `xPixels`, `yPixels` values specify the coordinates of the upper-left corner of the window in relation to the upper-left corner of the display device. Passing `0,0` as this coordinate pair places the window in the upper-left corner of the display device. Passing `10,10` places the window 10 pixels down and to the right of the upper-left corner of the display device.

However, your graphics driver implementation determines whether it can position the window containing the output plane according to `xPixels` and `yPixels`. In graphics drivers provided with the source implementation, X11 windowing systems do not observe the values of `xPixels` and `yPixels`. Instead, they tile multiple windows. Depending on your platform, the graphics driver may alter or ignore the `xPixels`, `yPixels` values you specify.

- `--balpha int 0 to 255`

The alpha channel (transparency) value of the background on which the Flash Lite instance renders the SWF file. Valid values are any integer 0 - 255. A value of 0 represents a fully transparent background and a value of 255 represents an opaque background. The default value is 255.

Use this option to blend the SWF content with other planes of graphics or video on your embedded system that provide a hardware-based compositor. That is, this value allows an underlying hardware-based graphics or video display to be visible through the SWF content.

For example, consider a SWF file that provides a video user interface. Use this option to overlay the user interface semi-transparently on a full-screen high definition video plane that is rendered and displayed in hardware. Setting the `--balpha` option to a semi-transparent value allows the video beneath the SWF content to “show through” the controls that the SWF content presents.

Note: *Video rendered and displayed in the hardware always obscures anything layered beneath it.*

- `--nospeedlimit`

Requests that the Flash Lite instance run the SWF content at the maximum frame rate possible. It also requests that Flash Player in the Flash Lite instance does not sleep in its main loop, and that it renders frames as fast as it can. Using this option can cause SWF content to play much faster than expected. Use this option to determine the maximum frame rate that a specific platform can deliver.

- `--quality [low|medium|high]`

Rendering quality of the Flash Lite instance. A value of `low` reduces the rendering quality, while a value of `high` increases it. The default is `high` quality.

Lower quality levels require less CPU usage to render each frame of Flash animation. However, they sometimes produce jagged lines, less-crisp text rendering, and blotchy color gradients.

ActionScript variables

```
--flashvars varstring
```

The `--flashvars` option passes the name-value pairs in *varstring* as root level Adobe® ActionScript® variables to the SWF content that the Flash® Lite® instance is playing.

The *varstring* string is a set of *name-value* pairs separated by ampersand (&) characters. To include a special character or non-printable character in the string, use a percent sign (%) and a two-digit hexadecimal value for the character. Use a plus sign (+) to represent a single blank space. The following example expresses the `username` and `password` parameters and their associated values:

```
--flashvars "username=DS&password=DanTube"
```

Security features

```
--security [trusted|sandboxed]
```

The `--security` option specifies the security mode that the Flash Lite instance runs in. Flash Lite for the digital home bases its security model on the Adobe® Flash® 8 security model. The value `trusted` specifies the local-trusted security mode. This value allows the SWF files to access both the local filesystem and the network. The value `sandboxed` means the Flash Lite instance runs in a security sandbox according to the origin of the SWF file passed to the stagecraft binary. The default value is `trusted`.

Use the `sandboxed` value for SWF files that originate from untrusted sources, such as the Internet. In `sandboxed` mode, the Flash Lite instance checks the local playback security setting of the SWF file. A SWF content developer specifies this setting at authoring time. The local playback security setting determines whether to allow the SWF content to access the network or the local filesystem; the SWF content cannot access both the network *and* the local filesystem.

For more information, see the [Flash Player 8 Security White Paper](#) on the Adobe website.

Key mapping

```
--keymap filename
```

The `--keymap` option specifies the *filename* of the key map XML file.

About key maps

A key map is an XML file that maps key codes used in Flash Lite for the digital home to key codes used in the ActionScript environment. The key codes used in Flash Lite for the digital home are defined in `include/ae/stagecraft/StagecraftKeyDefs.h`. Your platform implementation for handling user input uses these key codes for dispatching user input events into Flash Lite for the digital home. For example, when a user presses the Select key on a remote control device, your platform implementation dispatches the key code `AEKEY_ENTER` to Flash Lite for the digital home.

Flash Lite for the digital home then uses a default mapping to determine which ActionScript keycode to pass to the SWF content. For example, `AEKEY_ENTER` maps by default to the ActionScript keycode `Key.ENTER`.

A key map file allows you to replace the default mapping without rebuilding Flash Lite for the digital home. A key map file is useful, for example, when distributing SWF content authored for a desktop to a consumer electronics device. Typically, consumer electronics devices do not have a full keyboard for user input. For example, consider a desktop game in which the SWF content interprets pressing the spacebar to mean “shoot”. The key map file maps the Select key on the device’s remote control to the spacebar.

Key map XML file format

The following is an example of the XML format of a key map file:

```
<?xml version="1.0" encoding="UTF-8"?>
<keymap>
  <entry>
    <input type="aekey">AEKEY_INFO</input>
    <output type="char">i</output>
  </entry>
  <entry>
    <input type="char">'x'</input>
    <output type="flashkey">Key.ESCAPE</output>
  </entry>
  <entry>
    <input type="char">&#x26;</input>
    <output type="char">&#65;</output>
  </entry>
</keymap>
```

The XML format has one root element named `keymap`. The `keymap` element must contain one or more `entry` elements. Each `entry` element must contain an `input` and an `output` element. Each `input` and `output` element must have a `type` attribute. The `type` attribute identifies the type of value that the element contains. The type of an `input` element must be `aekey` or `char`. The type of an `output` element must be `aekey`, `char`, or `flashkey`.

The first entry in the XML example maps a `AEKEY_INFO` key event from Flash Lite for the digital home to the ASCII character `'i'` in the ActionScript environment. For this example, the remote control is configured to send an `AEKEY_INFO` key value to Flash Lite for the digital home when the user presses the “info” button. Then, this mapping directs Flash Lite for the digital home to behave as if the user pressed the `'i'` key on a computer keyboard. Therefore, Flash Lite for the digital home delivers the `'i'` key event to the SWF content. The second entry maps from the ASCII `'x'` character to the ActionScript `Key.ESCAPE` constant. For this example, the user input device has a physical keyboard but has no Esc key. This mapping causes the `'x'` key to behave like the Esc key to the SWF content. The last entry demonstrates using XML numeric character values for `char` values in either the `input` or the `output` elements.

Determining which key map to use

Flash Lite for the digital home uses the following steps to determine the key mapping:

- 1 Flash Lite for the digital home uses the key map file that you specify on the stagecraft command line.
- 2 If the stagecraft command line does not specify a key map file, Flash Lite for the digital home uses the key map file specified in a parameter in an interface call from the host application.
- 3 If the host application does not specify a key map file in an interface call, Flash Lite for the digital home looks for a `keymap.xml` file in the same directory as the SWF file.
- 4 If no `keymap.xml` file is in the same directory as the SWF file, Flash Lite for the digital home uses its default key mapping.

Filtering extensions

```
--extensionfilter filterString
```

The `--extensionfilter` option specifies a set of ActionScript extensions to register with a specific `StageWindow` instance or all `StageWindow` instances.

The `filterString` value is a comma-delimited string of ActionScript namespace specifiers. The last field of each namespace specifier can optionally be the wildcard character `*` (asterisk). For example, the `filterString` value `com.adobe.*, com.mycompany.*` specifies that the `StageWindow` instance can use all registered extensions which have a namespace beginning with `com.adobe` or `com.mycompany`.

The default `filterString` value is `*`. The default `*` value loads all extensions that the Extension Development Kit (EDK) registers as default extensions. In the source distribution, the only default extension is the `ProcessClass` extension. An empty `filterString` value specifies that Flash Lite for the digital home loads no extensions.

SSL certificates

```
--sslclientcerttable sslclientcerttable-string
```

Each Flash instance can have a table of Secure Sockets Layer (SSL) client certificates. The `--sslclientcerttable` option allows you to associate a target hostname with a certificate and private key. The client side uses the certificate and key in SSL mutual authentication with that target host.

The format of the `sslclientcerttable-string` is the following:

```
[hostname1.com^certfilename1.pem^keyfilename1.pem] [hostname2.com^certfilename2.pem^keyfilename2.pem]
```

Note: You can specify multiple associations. This example shows only two.

Brackets surround each association of a hostname, certificate file, and private key. Carets separate the three items. Each item is URL encoded. Therefore, if you have brackets or carets in your host name or filenames, URL encode them first. To URL encode a name, use the `%xx` URL character hexadecimal encoding syntax. Then, you can use brackets and carets as delimiters.

All client certificates and keys must be stored in `/etc/stagecraft-data/ssl/certs`. You can link this directory to another location in the filesystem.

For example, to authenticate with `http://www.myhost.mydomain`, using the certificate file `mycert.pem` and the private key file `mykey.pem`, use the following command-line argument:

```
stagecraft --sslclientcerttable [www.myhost.mydomain^mycert.pem^mykey.pem] myswf.swf
```

This command causes the Flash instance to use the files `/etc/stagecraft-data/ssl/certs/mycert.pem` and `/etc/stagecraft-data/ssl/certs/mykey.pem` for SSL mutual authentication with host `http://www.myhost.mydomain`.

Specify multiple target host entries as follows:

```
stagecraft --sslclientcerttable [www.myhost.mydomain^mycert.pem^mykey.pem] [myhost2.com^cert2.pem^key2.pem] myswf.swf
```

Generating performance statistics

You can use the `--tracefps` and `--tracefpsfull` options to generate performance statistics. For more information, see *Measuring performance* in *Optimizing Adobe Flash Lite for the Digital Home*.

Other options for debugging and performance tuning

Other options to the stagecraft binary control debugging and performance-tuning functionality.

```
--memlimit num [K|M]
```

Maximum system memory available to the Flash Lite instance executing the SWF file that follows this option on the command line. Set to zero for no limit. Use `k` to specify the number of kilobytes or `M` to specify the number of megabytes. This value is only for system memory, not other specialized memory such as memory allocated on your platform's graphics hardware.

`--astrace`

Prints ActionScript `trace()` output to the command line. This functionality occurs by default. Pass `--noastrace` to disable this functionality.

`--noastrace`

Disables command-line output of ActionScript `trace()` statements.

`--notrackmem`

Disables detection of memory leaks.

`--useshell`

Enables the IShell, which provides an interactive shell implementation that you can use to pass commands to Flash Lite for the digital home while it is running. For more information, see the `IShell.h` and `IShellCommand.h` files in the source distribution directory `include/ae/core/shell`.

`--noclearscreen`

Disables clearing of the display during initialization. The default behavior is to clear the display before executing the SWF content.

Chapter 4: Using Flash Lite for the digital home in Win32 environments

You can also build and debug Adobe® Flash® Lite® for the digital home in a Windows® environment. Flash Lite for the digital home was developed and tested using Microsoft Visual Studio 2008 Professional. No other Win32 development environment is officially supported. You do not have to build and run Flash Lite for the digital home in the Linux® environment to use it in Visual Studio.

Before you attempt to build and run Flash Lite for the digital home platforms in the Visual Studio 2008 Professional environment, make sure of the following:

- Visual Studio 2008 Professional is already installed and works correctly on the system you plan to use for Flash Lite for the digital home development.

Before using Visual Studio 2008 Professional to build and run Flash Lite for the digital home, make sure that you can use it to build and run a simple C++ program. For example, build and run one of the sample programs that Visual Studio provides. For more information, see Visual Studio 2008 Professional online Help.

- You have already installed the Flash Lite for the digital home source distribution in a location that is accessible to the Visual Studio development environment.

For more information, see “[Creating the source root directory](#)” on page 9.

Examples using Visual Studio 2008 use its default settings and layout.

Build Flash Lite for the digital home in Visual Studio 2008

- 1 Open the following file in Visual Studio 2008 Professional:

```
stagecraft/build/win32/vc9/AE_aWorkspaceMain.sln
```

The AE_aWorkspaceMain.sln solution opens and displays a list of all the Flash Lite for the digital home modules in the Solution Explorer panel. Examples of modules are AEKernel, AEModule, IFlashLib, and IGraphicsDriver.

- 2 Right-click on Solution AE_aWorkspaceMain in the Solution Explorer panel, and select Build Solution.

Visual Studio builds all the modules of Flash Lite for the digital home. If you have not yet built this source tree for a Win32 platform, Visual Studio builds all the Flash Lite for the digital home modules. Otherwise, it builds only modules that changed since they were last built.

To build an individual module, right-click on the module name in the Solution Explorer panel, and select Build Project.

Note: The Win32 version of Flash Lite for the digital home is for debugging purposes only. You cannot build non-debug versions of Flash Lite for the digital home platforms in Win32 environments.

**Got Feedback?
Click Here**

Run the stagecraft binary in Win32 environments

You can run Flash Lite for the digital home in Visual Studio 2008 or on the MS-DOS® command line. To run it, you execute the stagecraft binary executable. The stagecraft binary loads and runs Flash Lite for the digital home.

Run stagecraft in Visual Studio 2008

- 1 In the Solution Explorer panel, right-click the stagecraft project. Select Set as StartUp Project.
- 2 Right-click the stagecraft project again, and select Properties.
- 3 In the Stagecraft Property Pages window, select Configuration Properties > Debugging.
- 4 Verify that Command is set to `$(TargetPath)`. Visual Studio automatically sets `$(TargetPath)` to the stagecraft binary executable.
- 5 Verify that Working Directory has no value. By default, Visual Studio sets the working directory to the directory containing the project file. For example:

```
C:\Visual_Studio_2008\Projects\stagecraft\build\win32\vc9
```

- 6 Set Command Arguments to the command-line parameters for the stagecraft binary.

For example, to run the 1.swf movie that the Flash Lite for the digital home source distribution supplies, set Command Arguments to the following:

```
C:\Visual_Studio_2008\Projects\stagecraft\source\executables\cppunittest\testfiles\1.swf
```

- 7 To run stagecraft., select Debug > Start Debugging.

Stagecraft loads and runs Flash Lite for the digital home. Flash Lite for the digital home runs the specified SWF movie in a separate Window. You can set breakpoints in stagecraft_main.cpp and in the source files for Flash Lite for the digital home.

Run Flash Lite for the digital home from the DOS command line

- 1 Display the Command Prompt window.

For example, in your Windows toolbar, select Start > Run. Then enter `cmd` into the Run window, and press Enter.

- 2 Change to the directory that contains the stagecraft binary executable file.

For example, enter the following on the command line:

```
cd C:\Visual_Studio_2008\Projects\stagecraft\build\win32\vc9\debug\bin
```

- 3 To run a SWF file in Flash Lite for the digital home, type the following command and press Enter:

```
stagecraft pathToSWF
```

If the `pathToSWF` value contains spaces, enclose it in double-quotation (") marks.

Note: The DOS Command Prompt window does not show any output that would normally appear on the Linux command line. All output from a Win32 executable appears only in the Visual Studio debugging environment.

Index

Symbols

.as file 3

A

ActionScript
 about 3
 output trace 20
 alpha channel 16
 --astrace option 20

B

background transparency 16
 --bgalpha option 16
 binutils 4
 bitmap graphics 3
 Build command in Visual Studio 21
 building
 in Linux 6
 in Visual Studio 21

C

C++ 2
 codecs
 H.264 video 3
 On2 video 3
 Sorenson video 3
 codecs supported 3
 command line
 options, displaying 14
 compiler 4
 --contentdims option 15

D

debugging tools
 about 2
 development system, requirements 4
 directories
 required 9
 display options 15
 DOS instructions 22
 drivers, about 3

E

ECMAScript 3
 --extensionfilter option 18

extensions

ActionScript 3
 filtering of 18

F

files
 ActionScript 3
 as URLs 13
 FLA 3
 SWF 3
 FLA file 3
 Flash Lite, about 3
 Flash variables 17
 --flashvars option 17
 frame rate
 about 3
 setting 16

G

glibc library 4
 GNU compiler 4
 graphics, recommendations 3

H

H.264 video, support for 3

J

JavaScript 3

K

kerneltest program 7
 key mapping 17
 --keymap option 17

L

library
 glibc 4
 uclibc 4
 linkage errors, testing for 7

M

make utility
 options 10
 --memlimit option 19

modules

replacing 2

N

--noastrace option 20
 --noclearscreen option 20
 --nospeedlimit option 16
 --notrackmem option 20

O

options
 --astrace 20
 --bgalpha 16
 --contentdims 15
 display 15
 --extensionfilter 18
 --flashvars 17
 --keymap 17
 --noastrace 20
 --noclearscreen 20
 --nospeedlimit 16
 --notrackmem 20
 --outputdims 15
 --outputrect 14, 16
 --quality 16
 --security 17
 --sslclientcerttable 19
 --tracefps 19
 --tracefpsfull 19
 --useshell 20
 --outputdims option 15
 --outputrect option 16
 X11 windows and 14

P

performance, measuring 19

Q

--quality option 16
 quality, rendering 16

R

recompiling 2
 render plane, sizing 15
 rendering quality, setting 16

S

- security option 17
- security options 17
- source files
 - installing 6
 - location of 9
- source root directory 9
- Stage
 - location 16
 - size 15
- stagecraft
 - astrace option 20
 - balpha option 16
 - contentdims option 15
 - extensionfilter option 18
 - keymap option 17
 - memlimit option 19
 - nospeedlimit option 16
 - outputdims option 15
 - outputrect option 14, 16
 - quality option 16
 - security option 17
 - tracefps option 19
 - tracefpsfull option 19
- statistics, performance 19
- SWF files 3
 - running in Visual Studio 22
- system memory, setting available 19
- system requirements
 - development system 4
 - target system 3

T

- target system, requirements 3
- tracefps option 19
- tracefpsfull option 19
- transparency, background 16

U

- uclibc library 4
- user interface, high performance 2
- useshell option 20

V

- vector graphics 3
- video
 - H.264 3
 - Internet playback 2
 - sources 3

Visual Studio

- build command in 21
- building in 21
- non-debug builds and 4
- running in 22
- running SWF file in 22

W

- Win32
 - debug builds in 4
- Win32, debugging in 21
- Windows, building in 21

X

- X11 windows
 - outputrect option and 14