Protecting Content

Adobe® Flash® Access™

June 2010

Version 2.0

Adobe® Flash® Access™ 2.0 Protecting Content

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

Contents

About this document

This document explains how to use the Java™ APIs and reference implementations that are provided with Adobe® Flash® Access™ SDK to help secure content by performing the following tasks:

- Creating and managing policies
- Encrypting video and audio files
- Implementing a license server

Who should read this document?

This document is intended for publishers of video and audio files who use Flash Access SDK to protect their content and manage policies.

Conventions used in this document

This document uses the following conventions.

Name	Description
brackets [ ]	Indicates an optional item.
code italic	Indicates that you should replace the text in code italic with an actual value.

Additional information

The resources in this table provide additional information about Flash Access SDK.

For information about	See
Flash Access key terms and concepts, development environment, and an overview of the steps needed to deploy the SDK	*Overview*
The Java™ interfaces and classes used to create custom packaging tools and license servers	*Adobe Flash Access API Reference* *Downloadable zip file*
Best practices on how to securely deploy Flash Access	*Secure Deployment Guidelines*
Installing Flash Access SDK, securing video content using the SDK Java APIs, and information on the tools provided in the reference implementation.	*Protecting Content*

For information about	See
Describes how to use the Flash Access Certificate Enrollment website to request digital security certificates from Adobe.	*Certificate Enrollment Guide*
Outlines the usage rules or controls available to content owners and content distributors to limit usage of copyrighted content.	*Usage Rules*
The Java™ interfaces and classes for the Adobe® Flash® Access™ Server for Protected Streaming license server.	*Adobe® Flash® Access™ Server for Protected Streaming API Reference* *Downloadable zip file*

1 Introduction

Adobe® Flash® Access™ is an advanced digital rights management and content protection solution for high-value audiovisual content. Using tools that you create using Java APIs, you can create policies, apply policies to files containing audio and video content, and encrypt those files. The high-level steps for performing these tasks are as follows:

1. Use the Java APIs to set the policy properties and encryption parameters.
2. Create a policy describing the usage roles for the content. (See Working with policies.)

   You can create any number of policies. Most users create a small number of policies and apply them to many files.
3. Package a media file.

   In this context, *packaging a file* means to encrypt it and apply a policy to it. (See Packaging media files.)
4. Implement the license server to issue a license to the user.

The encrypted content is now ready for deployment, and the client can request the license from the server.

The SDK provides a Java API to accomplish these tasks, and includes reference implementations of the license server, and command line tools that are based on the Java APIs. For information, see Using the reference implementations.

Note: The architecture allows for usage policies to be specified and bound to content when the content is packaged. Before a client can play back content, the client must acquire a license for that computer. The license specifies the usage rules that are enforced and provides the key used to decrypt the content. The policy is a template for generating the license, but the license server may choose to override the usage rules when issuing the license. Note that the license may be rendered invalid by such constraints as expiration times or playback windows.

Usage rules

The following section describes the usage rules that you can specify in a policy.

Authentication

User authentication

Specifies whether a credential, such as username and password, is required to acquire a license. If authenticated (identity-based) licensing is specified, the server authenticates the user before issuing a license.

Example use case: A subscription service might require a username/password to be entered before issuing a content license. A DVD or Blu-ray disc with Digital Copy might provide a code or other token as proof of payment, which can be redeemed for an electronic download.

Time-based rules

Start date

Specifies the date after which a license is valid.

Example use case: Use an absolute date to issue content licenses ahead of the availability date of an asset, or to enforce an "embargo" period.

End date

Specifies the date after which a licenses expires.

Example use case: Use an absolute expiration date to reflect the end of distribution rights.

Relative end date

Specifies the license expiration date, expressed relative to the date of packaging.

Example use case: In an automated packaging process, use a single policy with this option for a series of videos to set the expiration date to 30 days relative to the date of packaging.

License caching duration

Specifies the duration a license can be cached in the client's local License Store without requiring reacquisition from the license server. You can alternatively specify an absolute date/time after which a license can no longer be cached.

Example use case: Use the license caching duration to specify a fixed amount of time valid for a particular license, such as in a rental use case. A 30-day rental can be specified (with license caching) to indicate the total license duration within which to consume the content.

Playback window

Specifies the duration a license is valid after the first time it is used to play protected content.

Example use case: Some business models allow a rental period of 30 days but, once playback begins, it must be completed in 48 hours. This 48-hour longevity of the license is defined as the playback window.

Runtime and application restrictions

White-list for Adobe® AIR® applications allowed to play protected content

Specifies the AIR applications that can play content. Specify the AIR application ID, min version, max version, and AIR publisher ID.

Example use case: Use this rule to limit playback to a given application (a particular AIR application), or to control which version can access the content.

White-list for Adobe® Flash® Player SWFs allowed to play protected content

New in 2.0: A white list for Adobe® Flash® Player SWFs allowed to play protected content

Specifies the SWF files that can play content. Specify the SWF file by a SWF URL or a SHA-256 digest computed using the contents of the SWF. If you use the SHA-256 digest, this usage rule also specifies the maximum amount of time to allow for the client to download and verify the SWF.

Example use case: Conceptually equivalent to SWF Verification in the case of Flash Media Server, but enforced on the client side to limit which video players can play the content.

Blacklist of DRM Clients restricted from accessing protected content

New in 2.0: Flash Access DRM module versions restricted from accessing protected content.

Specifies the DRM client that cannot access content. Specified by DRM client version and platform.

Example use case: In the event of a security breach, a newer version of the DRM client can be specified as the minimum version required for license acquisition and content playback. The license server checks that the DRM client making the license request meets the version requirements before issuing a license. The DRM client also checks the DRM version in the license before playing content. This client check is required in the case of domains where a license may be transferred to another machine.

Blacklist of application runtimes restricted from accessing protected content

New in 2.0: Application Runtimes restricted from accessing protected content.

Specifies the version of the AIR application or SWF file that cannot access content. Specify the restricted runtime (Flash Player or AIR), platform, and version.

Example use case: Similar to the DRM Client blacklist, a higher version of the Flash Player or AIR runtimes can be specified as the minimum version required for license acquisition and content playback.

Minimum security level for DRM and runtimes

Specifies the security level required to access content. Specified by individual security levels for each component.

Example use case: Certain types of content (for example HD video) might require a higher security level than other types.

Other policy options

Custom usage rules

Specifies custom usage rules. Custom data can be passed to the server during license acquisition. The interpretation/handling of this data is completely up to the implementation of the client application and license server.

Example use case: Enables extensibility of usage rules by allowing other business rules to be conveyed securely as part of the policy and/or content license. For security reasons, because these usage rules are enforced in custom client application code, this option should be used in conjunction with the AIR

application or Flash Player SWF white-list options. For more information, see "Runtime and application restrictions" on page 3.

License chaining

New in 2.0: License chaining

Allows a license to be updated using a parent root license for batch updating of licenses.

Example use case: Use this option to update any linked licenses by downloading a single root license. For example, implement subscription models where content can be played back as long as the user renews the subscription on a monthly basis. The benefit of this approach is that users only have to do a single license acquisition to update all of their subscription licenses.

Multiple play rights

Allows different usage rules to be specified for different platforms or applications.

Example use case: Using multiple play rights, you can create a policy to specify that Output Protection is required on the Microsoft® Windows® platform, and is optional on the Apple® Macintosh® and Linux® platforms.

Packaging Options

The following encryption options are selected at packaging time and cannot be modified during license acquisition.

Encrypting tracks

Specifies which parts of the content are encrypted: audio, video, or both.

Example use case: Permits encrypting just the tracks of the content that require protection, thus reducing the decryption overhead on the client and improving playback performance.

Encrypting script data

Specifies whether script data embedded in the content is encrypted.

Note: This rule only applies for FLV file format. Script data is always left in the clear for files in the F4V format.

Example use case: Use this option to leave script data unencrypted, which allows for metadata aggregation tools to read the metadata of protected content.

Partial encryption level

New in 2.0: Partial encryption level

Specifies whether all frames, or only a subset of frames, should be encrypted. There are three levels of encryption: low, medium and high.

Note: For video track in F4V/H.264 files only.

Partial encryption is designed to give content providers granularity over the percentage of their H.264 encoded content is encrypted. Encryption of content adds CPU overhead to the device that is decrypting and viewing the content. Use partial encryption to reduce CPU overhead while maintaining very strong protection of the content. A motivating case for using this feature is a single piece of content that is intended to be playable across low, medium, and high powered devices.

Due to the nature of video encoding, it is not necessary to encrypt 100% of video to make it unplayable if it is stolen. Partial encryption has three setting, low, medium, and high, and the associated percentages of encryption are dependent upon how the video is encoded. Because of this encoding dependency, the percentage of your content that is encrypted falls within the following ranges:

- High: Encrypts all samples.
- Medium: Encrypts a target 50% of the data.
- Low: Encrypts a target 20 to 30% of the data.

These settings were designed with the following rule: Any content that is encrypted at the low setting is also encrypted at the medium setting. This ensures that the same piece of content distributed at low encryption by one party and distributed at medium encryption by another party does not compromise the protection of the content.

Example use case: Reducing the encryption level decreases the decryption overhead on the client and improves playback performance on low-end machines.

Initial portion of content in the clear

New in 2.0: Initial portion of content in the clear

Specifies an optional amount of time, in seconds, that the beginning of the content is left in the clear (meaning it is not encrypted).

Example use case: Allows for faster time to playback while the Flash Access client downloads the license in the background. This speed improvement is because the unencrypted portion of the video begins playback immediately, while the Flash Access initialization and license acquisition occur behind the scenes. With this feature turned off, users may notice a delay in playback experience, as the client machine is performing all of the licensing steps before any video playback occurs.

Custom metadata

New in 2.0: Specify custom key/value to add to content metadata that can be interpreted by the server application.

The Flash Access content metadata format allows for inclusion of custom key/value pairs at packaging time to be processed by the license server during license issuance. This metadata is separate from the policy and can be unique for each piece of content.

Example use case: During a Beta phase, you include the custom property "Release:BETA" at packaging time. License servers can vend licenses to this content during the Beta period, but after the Beta period expires, the license servers disallow access to the content.

Multiple policies

Specify multiple policies to be associated with a single piece of content. The specific policy to be used is determined by the license server.

Example use case: If the same asset is used for both electronic sell-through and rental models, this option allows you to specify different sets of usage rules for each business model. The license server can choose which policy to use based on whether the customer purchased or rented the content.

Output protection

Output protection controls

New in 2.0: Control whether output to external rendering devices is protected. Specify analog and digital outputs independently.

Controls whether output to external rendering devices should be restricted. An external device is defined as any video or audio device that is not embedded in the computer. The list of external devices excludes integrated displays, such as in notebook computers. Analog and digital output restrictions can be specified independently.

The following options/levels of enforcement are available:

- Required — must be enabled in order to play content to an external device
- Use if available — attempt to enable, but allow playback if not available
- No protection — no output protection enablement is enforced
- No playback — don't allow playback to an external device

Note: While these rules are consistently enforced across all platforms, currently it is only possible to securely turn on output protection on Windows platforms. On other platforms (such as Macintosh and Linux) there are no supporting operating system functions available to third party applications.

Example use case: Some content might enforce output protection controls, and the level of protection can be set by the content distributor. If "Required" is specified and playback is attempted on a Macintosh, the client does not play back content on external devices. The content will, however, play back on internal monitors.

If "Required" is specified and playback is attempted on Linux, the client does not play back content on any devices because it is not possible to differentiate between internal and external devices.

If you specify "Use if available", output protection is turned on where possible. For example, on Windows machines that support the Certified Output Protection Protocol (COPP), the content is passed with output protection to an external display. This example is sometimes known as "selectable output control".

2 Setting up the SDK

To set up the Adobe® Flash® Access™ for use, copy files from the DVD. These files include JAR files containing code, certificates, and third-party classes. In addition, request a certificate from Adobe Systems Incorporated. You will be issued multiple credentials used to protect the integrity of packaged content, licenses, and communication between the client and server.

Setting up the development environment

From the DVD, copy the following SDK files for use in your development environment and your Java classpath:

- adobe-flashaccess-certs.jar (contains Adobe root certificates)
- adobe-flashaccess-sdk.jar (contains Adobe code)

You need the following third party JAR files also located on the DVD in the SDK's "thirdparty" folder:

- bcmail-jdk15-141.jar
- bcprov-jdk15-141.jar
- commons-discovery-0.4.jar
- commons-logging-1.1.1.jar
- jaxb-api.jar
- jaxb-impl.jar
- jaxb-libs.jar
- relaxngDatatype.jar
- rm-pdrl.jar
- xsdlib.jar

You also need one of the JAR files in the "thirdparty/jsafe" folder. jsafe.jar is the pure-Java version. Alternatively, you can use jsafeWithNative.jar, which has better performance, but also requires that platform specific libraries (for example, jsafe.dll for Windows or libjsafe.so for Linux) be added to the path.

Additionally, an optional part of the SDK is adobe-flashaccess-lcrm.jar. This file is only needed for functionality related to 1.x compatibility. If you previously deployed Adobe Flash Media Rights Management Server 1.x, you must add support to your license server so it will be able to handle old content and clients.

Requesting certificates

There are three types of credentials required to use Flash Access:

- Packager: used during packaging to sign the metadata added to the encrypted content
- License Server: used to protect the content encryption key in the metadata, and used by the license server to sign licenses
- Transport: used to protect requests/responses exchanged between the client and the license server

Enrollment is the process of requesting a certificate from Adobe. You can generate your keys and create a request that is sent to Adobe. Adobe then generates the certificate and sends it back to you. Adobe will not know the contents of the private key. Therefore, you must have a way to back up the key so that you can recover it in case of hardware failure.

You can keep a private key on a Hardware Security Module (HSM) and use the SDK to pass in the credential you obtain from the HSM. To use a credential stored on an HSM, use a JCE provider that can communicate with an HSM to get a handle to the private key. Then, pass the private key handle, provider name, and certificate containing the public key to `ServerCredentialFactory.getServerCredential()`.

The SunPKCS11 provider is one example of a JCE provider which can be used to access a private key on an HSM (see the Sun Java documentation for instructions on using this provider). Some HSMs also come with a Java SDK which includes a JCE provider.

The SDK supports multiple ways of storing credentials, including on an HSM or as a PKCS12 file. PKCS12 is a standard format for a file containing a credential (a public key certificate and its associated private key) encrypted using a password. PFX is the file extension commonly used for files of this format.

Credentials are used when the private key is required (for example, for the packager to sign the metadata, or for the license server to decrypt data encrypted with the license server or transport public key). Private keys must be closely guarded to ensure the security of your content and license server.

Adobe recommends using an HSM for maximum security. For more information, see *Secure Deployment Guidelines*.

PEM and DER are two ways of encoding a public key certificate. PEM is a base-64 encoding and DER is a binary encoding. Certificate files typically use the extension .cer, .pem. or .der. Certificates are used in places where only the public key is required. If a component requires only the public key to operate, it is better to provide that component with the certificate instead of a credential or PKCS12 file.

For instructions on how to obtain the Flash Access credentials, see the Certificate Enrollment Guide.

3 Working with policies

Using Adobe® Flash® Access™, content providers can apply policies to FLV files and F4V files. Using the policy management APIs, administrators can create, view details of, and update policies.

A *policy* defines how users can view content; it is a collection of information that includes security settings, authentication requirements, and usage rights. When policies are applied, encryption and signing allow content providers to maintain control of their content no matter how widely it is distributed. Protected files can be delivered by using Adobe® Flash® Media Server or an HTTP server. They can be downloaded and played in custom players built with Adobe® AIR® and Adobe® Flash® Player. The policy is a template for the license server to use when it generates a license. The client may also refer to the policy before requesting a license to determine whether it needs to prompt the user to authenticate before issuing a license request to the server.

A policy specifies one or more rights that are granted to the client. Typically, a policy includes, at a minimum, the "Play Right". It is also possible to specify multiple Play Rights, each with different restrictions. When the client encounters a license with multiple Play Rights, it uses the first one for which it meets all the restrictions. For example, this feature can be used to enforce different output protection settings on different platforms. For sample code illustrating this example, see CreatePolicyWithOutputProtection.java in the Reference Implementation Command Line Tools "samples" directory.

You can accomplish the following tasks by using the policy management APIs:

- Create and update policies
- View policy details

In addition to these operations, it is possible to create a policy update list that does not require that content be repackaged after a policy is updated. For more information, see Packaging media files.

For details about the Java API discussed in this chapter, see *Adobe Flash Access API Reference*.

For information about the Policy Manager reference implementation, see Using the reference implementations.

Creating and updating policies

You can create and update policies using the Java API.

Creating a policy using the Java API

Updating a policy using the Java API

Creating a policy using the Java API

To create a policy by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.

2. Create a `com.adobe.flashaccess.sdk.policy.Policy` object and specify its properties, such as the rights, license caching duration, and policy end date.

3. Serialize the `Policy` object and store it in a file or database.

The following Java file shows how to create a policy.

Example: *Creating a policy using the Java API*

```
/**************************************************************************
 *
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 **************************************************************************/
package com.adobe.flashaccess.samples.policy;

import com.adobe.flashaccess.sdk.policy.LicenseServerInfo;
import com.adobe.flashaccess.sdk.policy.Policy;
import com.adobe.flashaccess.sdk.policy.PolicyException;
import com.adobe.flashaccess.sdk.policy.ServerInfo.AuthenticationType;
import com.adobe.flashaccess.sdk.rights.PlayRight;
import com.adobe.flashaccess.sdk.rights.Right;

import java.io.FileOutputStream;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;

/**
 * Demonstrates policy creation.
 */
public class CreatePolicy
{
   public static void main(String[] args) {
      // Create a new Policy object.
      // False indicates the policy does not use license chaining.
      Policy policy = new Policy(false);

      policy.setName("DemoPolicy");

      // Specify that the policy requires authentication to obtain a license.
      policy.setLicenseServerInfo(new
LicenseServerInfo(AuthenticationType.UsernamePassword));

      // A policy must have at least one Right, typically the play right
      PlayRight play = new PlayRight();
```

```
        // Users may only view content for 24 hours.
        play.setPlaybackWindow(24L * 60 * 60);

        // Add the play right to the policy.
        List<Right> rightsList = new ArrayList<Right>();
        rightsList.add(play);
        policy.setRights(rightsList);

        // Licenses may be stored on the client for 7 days after downloading
        policy.setLicenseCachingDuration(7L * 24 * 60 * 60);
        try {
           // Content will expire December 31, 2010
           SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
           policy.setPolicyEndDate(dateFormat.parse("2010-12-31"));
        } catch (ParseException e) {
           // Invalid date specified in dateFormat.parse()
           e.printStackTrace();
        }

        try
        {
           // Serialize the policy
           byte[] policyBytes = policy.getBytes();
           System.out.println("Created policy with ID: " + policy.getId());

           // Write the policy to a file.
           // Alternatively, the policy may be stored in a database.
           FileOutputStream out = new FileOutputStream("demopolicy.pol");
           out.write(policyBytes);
           out.close();
        } catch (PolicyException e) {
           // The policy could not be created.
           e.printStackTrace();
        } catch (IOException e) {
           // There was an error writing the policy to the file.
           e.printStackTrace();
        }
     }
}
```

Updating a policy using the Java API

If you update policies after packaging content, the license server needs to have the latest version in order to issue licenses using the updated policy. One way to achieve this is through a policy update list. For more information, see Working with Policy Update Lists. To update a policy by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.

2. Create a `Policy` instance and read in the policy from a file or database.

3. Update the `Policy` object by setting its properties, such as its name.

4. Serialize the updated `Policy` object and store it in a file or database.

The following Java file shows how to update a policy.

Example: *Updating a policy using the Java API*

```
/**************************************************************************
 *
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 **************************************************************************/
package com.adobe.flashaccess.samples.policy;

import com.adobe.flashaccess.sdk.policy.Policy;
import com.adobe.flashaccess.sdk.policy.PolicyException;
import com.adobe.flashaccess.sdk.rights.ModuleRequirements;
import com.adobe.flashaccess.sdk.rights.PlayRight;
import com.adobe.flashaccess.sdk.rights.Right;
import com.adobe.flashaccess.sdk.util.VersionInfo;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;

/**
 * Demonstrates updating an existing policy.
 */
public class UpdatePolicy
{
   public static void main(String[] args) {
      // Specify the name of the file containing the policy.
      String policyFile = "demopolicy.pol";

      // Read the policy from the file.
      Policy policy = null;
      try {
         policy = readPolicyFromFile(new File(policyFile));
      } catch (PolicyException e) {
         // File contains an invalid policy
         e.printStackTrace();
      } catch (IOException e) {
         // Error reading policy from file
         e.printStackTrace();
```

```
      }
      if (policy == null)
        return;

      System.out.println("Updating policy with ID: " + policy.getId() +
                      " and revision: " + policy.getRevision());

      // Change the policy name.
      policy.setName("UpdatedDemoPolicy");

      // Add DRM module restrictions to the play right
      for (Right r: policy.getRights()) {
        if (r instanceof PlayRight) {
          PlayRight pr = (PlayRight) r;
          // Disallow Linux versions up to and including 1.9.  Allow
          // all other OSes and Linux versions above 1.9
          VersionInfo toExclude = new VersionInfo();
          toExclude.setOS("Linux");
          toExclude.setReleaseVersion("1.9");
          Collection<VersionInfo> exclusions = new ArrayList<VersionInfo>();
          exclusions.add(toExclude);
          ModuleRequirements drmRestrictions = new ModuleRequirements();
          drmRestrictions.setExcludedVersions(exclusions);
          pr.setDRMModuleRequirements(drmRestrictions);
          break;
        }
      }

      try
      {
        // Serialize the policy.
        byte[] policyBytes = policy.getBytes();
        System.out.println("New policy revision number: " +
policy.getRevision());

        // Write the policy to a file.
        // Alternatively, the policy may be stored in a database.
        FileOutputStream out = new FileOutputStream("demopolicy-updated.pol");
        out.write(policyBytes);
        out.close();
      } catch (PolicyException e) {
        // The policy could not be created, because required fields were not
set.
        e.printStackTrace();
      } catch (IOException e) {
        // Error writing policy to file
        e.printStackTrace();
      }
    }

    private static Policy readPolicyFromFile(File policyFile) throws
IOException, PolicyException {
      FileInputStream in = null;
      try {
        in = new FileInputStream(policyFile);
```

```
            byte[] polBuf = new byte[ (int) policyFile.length() ];
            in.read( polBuf );
            return new Policy(polBuf);
         } finally {
            if (in != null)
               in.close();
         }
      }
   }
```

4 Packaging media files

Packaging refers to the process of encrypting and applying a policy to FLV or F4V files. Use the media packaging APIs to package files.

Packaging is decoupled from the license server. There is no need for the packager to connect to the license server to exchange any information about the content. Everything the license server needs to know to issue the license is included in the content metadata.

When a file is encrypted, its contents cannot be parsed without the appropriate license. Flash Access allows you to select which parts of the file to encrypt. Because Adobe® Flash® Access™ can parse the file format of the FLV and F4V content, it can intelligently encrypt selective parts of the file instead of the entire file as a whole. Data such as metadata and cue points can remain unencrypted so that search engines can still search the file.

It is possible for a given piece of content to have multiple policies. This could be useful, for example, if you would like to license content under different business models without having to package the content multiple times. For example, you could allow anonymous access for a short period of time, and after that allow the customer to buy the content and have unlimited access. If content is packaged using multiple policies, the license server must implement logic for selecting which policy to use to issue a license.

There are numerous options available when packaging content. These are specified in the `DRMParameters` interface and the classes implementing that interface, which are the `F4VDRMParameters` and `FLVDRMParameters`. With these classes you can set signature and key parameters, as well as indicate whether to encrypt audio content, video content, or script data. To see how these are implemented in the reference implementation, see the descriptions of the command line options discussed in Policy Manager. These options are based on the Java API and are therefore available for programmatic usage.

The packaging options include:

- Encryption options (audio, video, partial encryption). For more information on encryption options, see "Packaging options" on page 9.
- License server URL (the client uses this as the base URL for all requests sent to the license server)
- License server transport certificate
- License sever certificate, used to encrypt the CEK.
- Packager credential for signing metadata

Flash Access provides an API for passing in the CEK. If no CEK is specified, the SDK randomly generates it. Typically you need a different CEK for each piece of content. However, in Dynamic Streaming, you would likely use the same CEK for all the files for that content, so the user only needs a single license and can seamlessly transition from one bit rate to another. To use the same key and license for multiple pieces of content, pass the same `DRMParameters` object to `MediaEncrypter.encryptContent()`, or pass in the CEK using `V2KeyParameters.setContentEncryptionKey`. To use a different key and license for each piece of content, create a new `DRMParameters` instance for each file.

In some cases you may need to store the content metadata as a separate file and make it available to the client separate from the content. To do this, invoke `MediaEncrypter.encryptContent()`, which returns a `MediaEncrypterResult` object. Call `MediaEncrypterResult.getKeyInfo()` and cast the result to `V2KeyStatus`. Then retrieve the content metadata and store it in a file.

All of these tasks can be accomplished using the Java API. For details about the Java API discussed in this chapter, see *Adobe Flash Access API Reference*.

For information about the Media Packager reference implementation, see Using the reference implementations.

Encrypting and examining content

You can encrypt and examine content using the Java API.

Encrypting content

Encrypting FLV and F4V content involves the use of a `MediaEncrypter` object. You can also package FLV and F4V files that contain only audio tracks. When encrypting H.264 content for lower-end devices, it may be practical to apply only partial encryption to improve performance. In such cases, an F4V file may be partially encrypted using the `F4VDRMParameters.setVideoEncryptionLevel` method.

To encrypt an FLV or an F4V file by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.
2. Create a `ServerCredential` instance to load the credentials needed for signing.
3. Create a `MediaEncrypter` instance. Use a `MediaEncryperFactory` if you do not know what type of file you have. Otherwise you can create an `FLVEncrypter` or `F4VEncrypter` directly.
4. Specify the encryption options by using a `DRMParameters` object.
5. Set the signature options using a `SignatureParameters` object and pass the `ServerCredential` instance to its `setServerCredentials` method.
6. Set the key and license information using an `V2KeyParameters` object. Set the policies using the `setPolicies` method. Set the information needed by the client to contact the license server by calling the `setLicenseServerUrl` and `setLicenseServerTransportCertificate` methods. Set the CEK encryption options using the `setKeyProtectionOptions` method, and its custom properties using the `setCustomProperties` method. Finally, depending on the type of encryption used, cast the `FMRMSKeyParameters` object to one of `VideoDRMParameters`, `AudioDRMParameters`, `FLVDRMParameters`, or `F4VDRMParameters`, and set the encryption options.
7. Encrypt the content by passing the input and output files and encryption options to the `MediaEncrypter.encryptContent` method.

The following Java file shows how to encrypt a FLV file.

Example: *Encrypting a file using the Java API*

```
/**************************************************************************
 *
```

```
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 ********************************************************************/
package com.adobe.flashaccess.samples.mediapackager;

import com.adobe.flashaccess.sdk.cert.CertificateFactory;
import com.adobe.flashaccess.sdk.cert.ServerCredential;
import com.adobe.flashaccess.sdk.cert.ServerCredentialException;
import com.adobe.flashaccess.sdk.cert.ServerCredentialFactory;
import com.adobe.flashaccess.sdk.media.drm.DRMParameters;
import com.adobe.flashaccess.sdk.media.drm.MediaEncrypter;
import com.adobe.flashaccess.sdk.media.drm.MediaEncrypterFactory;
import com.adobe.flashaccess.sdk.media.drm.MediaEncrypterResult;
import com.adobe.flashaccess.sdk.media.drm.UnknownMediaException;
import com.adobe.flashaccess.sdk.media.drm.WarningMessage;
import com.adobe.flashaccess.sdk.media.drm.format.AudioDRMParameters;
import com.adobe.flashaccess.sdk.media.drm.format.VideoDRMParameters;
import com.adobe.flashaccess.sdk.media.drm.format.f4v.F4VDRMParameters;
import com.adobe.flashaccess.sdk.media.drm.format.flv.FLVDRMParameters;
import com.adobe.flashaccess.sdk.media.drm.keys.KeyRetrievalException;
import
com.adobe.flashaccess.sdk.media.drm.keys.v2.AsymmetricKeyProtectionOptions;
import com.adobe.flashaccess.sdk.media.drm.keys.v2.V2KeyParameters;
import com.adobe.flashaccess.sdk.media.drm.sig.SignatureParameters;
import com.adobe.flashaccess.sdk.policy.Policy;
import com.adobe.flashaccess.sdk.policy.PolicyException;
import com.adobe.flashaccess.sdk.util.ApplicationProperties;

import java.io.*;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

/**
 * Demonstrates packaging an FLV or F4V file.
 */
public class EncryptContent {
   public static void main(String[] args) {
      // Specify the original file.
      File inputFile = new File("C:/samplevideo.flv");

      // Specify the location of the encrypted file.
      File outputFile = new File("C:/encryptedsamplevideo.flv");

      // Specify the file containing packaging credentials issued by Adobe.
      // These credentials are used to sign the encryption metadata in the
content.
      File packagerCredentialFile = new File("C:/packagerCredentials.pfx");
```

```
     // Specify the password needed to open the certificate file.
     String packagerCredentialPassword = "password";

     // Specify the file containing the license server's certificate.
     // This certificate is used to encrypt the content key.
     File licenseServerCertificateFile = new
File("C:/licenseServerCert.cer");

     // Specify the file containing the license server's transport
certificate.
     // This certificate is used to protect request messages sent by the client
to the server.
     File tranportServerCertificateFile = new
File("C:/licenseServerTransportCert.cer");

     // Specify the file containing the policy created using Policy Manager.
     File policyFile = new File("C:/policy1.pol");
     String licenseServerUrl = "http://localhost:8080";

     // Assign an ID for this content, which will be passed to
     // the External Authorizer when users download a license.
     String contentID = inputFile.getName();

     // Indicate which parts of the video should be encrypted.
     boolean encryptVideo = true;
     boolean encryptAudio = true;
     boolean encryptScript = true;

     try {
       // Load the credentials for signing.
       ServerCredential signingCred =
ServerCredentialFactory.getServerCredential(
          packagerCredentialFile,
           packagerCredentialPassword
       );

       // Get a MediaEncrypter. Use a MediaEncrypterFactory if
       // you do not know what type of file you have.
       // Otherwise, you can create an FLVEncrypter or F4VEncrypter directly.
       MediaEncrypter encrypter = MediaEncrypterFactory.getMediaEncrypter(
         new BufferedInputStream(new FileInputStream(inputFile))
       );

       // Fill in the encryption options.
       DRMParameters params = encrypter.createEncrypterParameters();

       // Specify the signature options.
       SignatureParameters sigParams =
MediaEncrypter.createSignatureParameters();
       sigParams.setServerCredentials(signingCred);
       params.setSignatureParameters(sigParams);

       // Set the key and license information.
       V2KeyParameters keyParams =
MediaEncrypter.createKeyParameters(V2KeyParameters.class);
```

```
        keyParams.setLicenseServerUrl(licenseServerUrl);
        keyParams.setContentId(contentID);

        // Set the policies.
        Policy[] policies = new Policy[1];
        policies[0] = loadPolicy(policyFile);
        keyParams.setPolicies(policies);

        // Set the transport certificate for the license server

keyParams.setLicenseServerTransportCertificate(loadCert(tranportServerCertif
icateFile));

        // Set the key options. An encrypted key means the content key is
encrypted
        // using the license server's public key.
        AsymmetricKeyProtectionOptions  keyOpts = new
AsymmetricKeyProtectionOptions (
          loadCert(licenseServerCertificateFile)
        );
        keyParams.setKeyProtectionOptions(keyOpts);

        // Add custom properties.
        ApplicationProperties customProps = new ApplicationProperties();
        customProps.addUTF8String("My Custom Property", "Property Value");
        keyParams.setCustomProperties(customProps);
        params.setKeyParameters(keyParams);

        // Specify the type of encryption to be used.
        if (params instanceof VideoDRMParameters) {
          ((VideoDRMParameters) params).setEncryptVideo(encryptVideo);
        }
        if (params instanceof AudioDRMParameters) {
          ((AudioDRMParameters) params).setEncryptAudio(encryptAudio);
        }
        if (params instanceof FLVDRMParameters) {
          ((FLVDRMParameters) params).setEncryptScript(encryptScript);
        }
        if (params instanceof F4VDRMParameters) {
          // Change to Medium or Low to enable partial encryption of H.264
content
          ((F4VDRMParameters)
params).setVideoEncryptionLevel(F4VDRMParameters.EncryptionLevel.High);
        }

        // Encrypt the content,
        MediaEncrypterResult result = encrypter.encryptContent(inputFile,
outputFile, params);

        // Display the results,
        if (result.hasWarnings()) {
          System.out.println("Media Packaging Finished with warnings");
          for (WarningMessage warn : result.getWarnings()) {
            System.out.println("WARNING: " + warn.getMessage());
          }
```

```
            } else {
               System.out.println("Media Packaging Successful");
            }
         } catch (CredentialException e) {
            e.printStackTrace();
            System.err.println("Failed to load signing credential");
         } catch (UnknownMediaException e) {
            e.printStackTrace();
            System.err.println("Unable to determine file type");
         } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Error opening file");
         } catch (KeyRetrievalException e) {
            e.printStackTrace();
            System.err.println("Unable to get key from server");
         } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Error: " + e.getMessage());
         }
      }

      private static Policy loadPolicy(File policyFile) throws IOException,
   PolicyException {
         FileInputStream in = null;
         try {
            in = new FileInputStream(policyFile);
            byte[] polBuf = new byte[ (int) policyFile.length() ];
            in.read( polBuf );
            return new Policy(polBuf);
         } finally {
            if (in != null)
               in.close();
         }
      }

      public static X509Certificate loadCert(File certFile) throws IOException {
         FileInputStream in = null;
         try {
            in = new FileInputStream(certFile);
            return CertificateFactory.loadCert(in);
         } catch (CertificateException e) {
            throw new IOException("Unable to load license server certificate");
         } finally {
            if (in != null)
               in.close();
         }
      }
   }
```

Examining encrypted file content

To examine the contents of an FLV or an F4V file by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.
2. Create a `MediaEncrypter` instance.
3. Pass the encrypted file to the `MediaEncrypter.examineEncryptedContent` method, which returns a `KeyMetaData` object.
4. Inspect the information within the `KeyMetaData` object.

The following Java file shows how to examine the content of an encrypted FLV file.

Example: *Examining encrypted file content using the Java API*

```
/**************************************************************************
 *
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 **************************************************************************/
package com.adobe.flashaccess.samples.mediapackager;

import com.adobe.flashaccess.sdk.media.drm.MediaEncrypter;
import com.adobe.flashaccess.sdk.media.drm.format.flv.FLVEncrypter;
import com.adobe.flashaccess.sdk.media.drm.keys.KeyMetaData;
import com.adobe.flashaccess.sdk.media.drm.keys.v2.V2KeyMetaData;

import java.io.*;

/**
 * Demonstrates examining an FLV or F4V that was previously packaged.
 */
public class ExamineContent {
   public static void main(String[] args) {
      // Location of encrypted file
      File encryptedFile = new File("C:/encryptedsamplevideo.flv");

      // Get a MediaEncrypter.
      MediaEncrypter encrypter = new FLVEncrypter();

      try {
         // Get information about the key
         KeyMetaData keyInfo =
encrypter.examineEncryptedContent(encryptedFile);
         if (keyInfo == null) {
            System.out.println("Unable to find encryption metadata in file");
         } else if (keyInfo instanceof FMRMSKeyMetaData) {
            V2KeyMetaData metadata = (V2KeyMetaData)keyInfo;
            System.out.println("License ID: " +
metadata.getContentMetadata().getLicenseId());
```

```
            }
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Error reading file");
        }
    }
}
```

5 Implementing the license server

Your license server will handle authentication and license acquisition requests. The license server uses the Adobe® Flash® Access™ SDK to perform these tasks:

- Process authentication requests
- Process license requests
- Upgrade 1.x content to 2.0 (optional)

In addition, the server needs to provide business logic for authenticating users, determining if users are authorized to view content, and optionally track license usage.

For details about the Java API discussed in this chapter, see *Adobe Flash Access API Reference*.

Handling general requests and responses

The general approach to handling requests is to create a handler, parse the request, set the response data or error code, and close.

The base class used to handle single request/response interaction is `com.adobe.flashaccess.sdk.protocol.MessageHandlerBase`. The handler reads the request data and parses the request into an instance of `RequestMessageBase`. The caller can examine the information in the request and decide whether to return an error or a successful response (subclasses of `RequestMessageBase` provide a method for setting response data).

If the request is successful, set the response data; otherwise invoke `RequestMessageBase.setErrorData()` on failure. Always end the implementation by invoking the `close()` method (it is recommended that `close()` be called in the `finally` block of a `try` statement). See the `MessageHandlerBase` API reference documentation for an example of how to invoke the handler.

Note: HTTP status code 200 (OK) should be sent in response to all requests processed by the handler. If the handler could not be created due to a server error, the server may respond with another status code, such as 500 (Internal Server Error).

For replay protection, it may be prudent to check whether the message identifier has been seen recently by calling `RequestMessageBase.getMessageId()`. If so, an attacker may be trying to replay the request, which should be denied. To detect replay attempts, the server can store a list of recently seen message ids and check each incoming request against the cached list. To limit the amount of time the message identifiers need to be stored, call `HandlerConfiguration.setTimestampTolerance()`. If this property is set, the SDK will deny any request that carries a timestamp more than the specified number of seconds off the server time.

For rollback detection, some usage rules require the client to maintain state information for enforcement of the rights. For example, to enforce the playback window usage rule, the client stores the date and time when the user first began viewing the content. This event triggers the start of the playback window. To securely enforce the playback window, the server needs to ensure that the user is not backing up and restoring the client state in order to remove the playback window start time stored on the client. The server does this by tracking the value of the client's rollback counter. For each request, the server gets the

value of the counter by calling `RequestMessageBase.getClientState()` to obtain the `ClientState` object, then calling `ClientState.getCounter()` to obtain the current value of the client state counter. The server should store this value for each client (use `MachineId.getUniqueId()` to identify the client associated with the rollback counter value), and then call `ClientState.incrementCounter()` to increase the counter value by one. If the server detects that the counter value is less than the last value seen by the server, the client state may have been rolled back. For more information on client state tamper detection, see the `ClientState` API reference documentation.

The `HandlerConfiguration` class stores server configuration information, including revocation list information, timestamp tolerance, policy update lists, and revocation lists. It is used to initialize concrete `AbstractHandler` instances. In addition to configuration used by the license server, `HandlerConfiguration` stores configuration information that can be sent to the client to control how licenses are enforced. This is done by creating a `ServerConfigData` class and calling `HandlerConfiguration.setServerConfigData()` (these settings apply only to licenses issued by this license server). The clock windback tolerance is one property that can be set by the license server to control how the client enforces licenses. By default, users may set their machine clock back 4 hours without invalidating licenses. If a license server operator wishes to use a different setting, the new value can be set in the `ServerConfigData` class. When you change the value of any of these settings, be sure to increment the version number by calling `setVersion()`. The new values will only be sent to the client if the version on the client is less than the current `ServerConfigData` version.

The client uses the License Server URL specified at packaging time as the base URL for all requests sent to the license server. For example, if the server URL is specified as "http://licenseserver.com/path", the client will send requests to "http://licenseserver.com/path/*flashaccesspath*". See the following sections for details on the specific path used for each type of request. When implementing your license server, be sure the server responds to the paths required for each type of request.

Crossdomain policy file

If the license server is hosted on a different domain than the video playback SWF, then a cross-domain policy file (crossdomain.xml) might be necessary to allow the SWF to request licenses from the license server. A cross-domain policy file is an XML file that provides a way for the server to indicate that its data and documents are available to SWF files served from other domains. Any SWF file that is served from a domain that the server's cross-domain policy file specifies is permitted to access data or assets from that server.

Adobe recommends that developers follow best practices when deploying the cross-domain policy file by only allowing trusted domains to access the license server and limiting the access to the license sub-directory on the web server. For more information on cross-domain policy files, please see the following locations:

- Website controls (policy files)
- Cross-domain policy file specification

Handling authentication requests

The `AuthenticationHandler` class is used to process authentication requests. It is used only for username/password authentication.

When generating the authentication token, the token expiration date must be specified. Custom properties may also be included in the token. If set, those properties will be visible to the server when the

authentication token is sent in subsequent requests. See Handling license requests below for information on handling custom authentication tokens.

The handler reads an authentication request and parses the request message when `parseRequest()` is called. The server implementation examines the user credentials in the request, and if the credentials are valid, generates an `AuthenticationToken` object by calling `getRequest().generateAuthToken()`. If `AuthenticationRequestMessage.generateAuthToken()` is not called before `close()`, an authentication failure error code is sent.

The request URL is constructed as follows: "*License Server URL in the metadata*" + "/flashaccess/authn/v1". See the `AuthenticationHandler` API reference documentation for an example of how to invoke the handler.

Handling license requests

To request a license, the client sends the metadata that was embedded in the content during packaging. The license server uses the information in the content metadata to generate a license.

Note: License requests are different from other types of requests in that the protocol between the client and server supports batching of license requests (multiple licenses may be requested at one time). For 2.0, the client sends one license request at a time, but batch support may be added in the future.

To accommodate batch requests, `LicenseHandler` extends `BatchHandlerBase`, which provides the caller with a `List` of `LicenseRequestMessage` objects.

The `LicenseHandler` reads a license request and parses the request. Since the request from the client may contain multiple license requests, the server should iterate through the `LicenseRequestMessages` returned by `getRequests()`. For each request either generate a license or set an error code (see the `LicenseRequestMessage` API reference documentation for details).

For each license request, the server determines whether it will issue a license. Call `LicenseRequestMessage.getContentInfo()` to obtain information extracted from the content metadata, including the content ID, license ID, and policies.

Call `Policy.getRootLicenseId()` to determine if the policy has a root license. If the policy has a root license, the server decides whether to issue the user a root license, leaf license, or both. Typically, the server issues a leaf license the first time the user requests a license for a particular machine and a root license thereafter. To determine if the machine already has a leaf license for the specified policy, call `LicenseRequestMessage.clientHasLeafForPolicy()`.

To give the user a leaf license, the SDK must decrypt the CEK contained in the content metadata and re-encrypt it for the machine requesting a license. To decrypt the CEK, the server must provide information required to decrypt the key. Call `ContentInfo.setKeyRetrievalInfo()` and provide an `AsymmetricKeyRetrieval` object. Since it is possible for a piece of content to have multiple policies, you must determine which policy to use and call `LicenseRequestMessage.setSelectedPolicy()`. Then call `LicenseRequestMessage.generateLicense()` to generate the license. Using the `License` object that is returned, you may modify the expiration or rights in the license.

If an error occurs while parsing the request, a `HandlerParsingException` is thrown. This exception contains error information to be returned to the client. To retrieve the error information, call `HandlerParsingException.getErrorData()`. If an error occurs while generating a license because the policy requirements have not been satisfied, a `PolicyEvaluationException` is thrown. This

exception also includes `ErrorData` to be returned to the client. See the API documentation for `LicenseRequestMessage.generateLicense()` for details on how policies are evaluated during license generation.

The client can send a license preview request, meaning that the application can carry out a preview operation before asking the user to buy the content in order to determine whether the user's machine actually meets all the criteria required for playback. *License preview* refers to the client's ability to preview the license (to see what rights the license allows) as opposed to previewing the content (viewing a small portion of the content before deciding to buy). Some of the parameters that are unique to each machine are: outputs available and their protection status, the runtime/DRM version available, and the DRM client security level. The license preview mode allows the runtime/DRM client to test the license server business logic and provide information back to the user so he can make an informed decision. Thus the client can see what a valid license looks like but would not actually receive the key to decrypt the content. Support for license preview is optional, and only necessary if you implement a custom client that uses this functionality.

The request URL is constructed as follows: "*License Server URL in the metadata*" + "/flashaccess/license/v1".

If identity-based licensing is used, the server checks for a valid authentication token before issuing a license.

Note: To preview a license for identity-based content, a client must authenticate.

To determine whether the client sent a preview request or license acquisition request, call `LicenseRequestMessage.getRequestPhase()` and compare it to `LicenseRequestMessage.RequestPhase.Acquire`.

Licenses and errors are sent at one time when `LicenseHandler.close()` is called.

Authentication

A license request can contain an authentication token. If username/password authentication was used, the request may contain an `AuthenticationToken` generated by the `AuthenticationHandler`, and the SDK will check if the token is valid.

If the client and server are using a custom authentication mechanism, the client obtains an authentication token through some other channel and passes the token into the client APIs. Use `RequestMessageBase.getRawAuthenticationToken()` to get the custom authentication token and implement the logic to check that the token is valid.

Updating policies

If policies are updated after the content is packaged, provide the updated policies to the license server so the updated version can be used when issuing a license. If your license server has access to a database for storing policies, you can retrieve the updated policy from the database and call `LicenseRequestMessage.setSelectedPolicy()` to provide the new version of the policy.

For license servers that do not rely on a central database, the SDK provides support for Policy Update Lists. A policy update list is a file containing a list of updated or revoked policies. When a policy is updated, generate a new Policy Update List and periodically push the list out to all the license servers. Pass the list to the SDK by setting `HandlerConfiguration.setPolicyUpdateList()`. If an update list is provided, the SDK consults this list when parsing the content metadata. `ContentInfo.getUpdatedPolicies()` contains the updated versions of policies specified in the metadata.

Using machine identifiers

Authentication and License requests contain information about the machine token issued to the client during individualization. The machine token contains a Machine Id, an identifier assigned during individualization. Use this identifier to count the number of machines from which a user has requested a license.

There are two ways of using the identifier. The `getUniqueId()` method returns a string. You can store the strings in a database and search by identifier. However, if the user wipes out the machine and individualizes again, he will get a different identifier and the server counts it as a second machine. This identifier is also different between Adobe® AIR®, Adobe® Flash® Player in Internet Explorer®, and Flash Player in FireFox on the same machine.

To more accurately count machines, you can use `getBytes()` to store the whole identifier. To determine if the machine has been seen before, get all the identifiers for a user name and call `matches()` to check if any match. Because the `matches()` method must be used to compare the values returned by `MachineId.getBytes`, this option is only practical when there are a small number of values to compare (for example, the machines associated with a particular user).

Handling FMRMS compatibility

There are two types of requests related to Flash Media Rights Management Server 1.x compatibility. One type of request is used to prompt 1.x clients to upgrade to 2.0. Another is used to update 1.x metadata to the 2.0 format before a license can be requested. Support for these requests is only needed if you previously deployed content using FMRMS 1.0 or 1.5.

Upgrading clients

If an FMRMS 1.x client contacts a Flash Access 2.0 server, the server needs to prompt the client to upgrade. This is achieved using `com.adobe.flashaccess.sdk.protocol.compatibility.FMRMSv1RequestHandler`. Unlike other Flash Access request handlers, this handler does not provide access to any request information or require any response data to be set. Create an instance of the `FMRMSv1RequestHandler`, and then call `close()`.

The request URL is constructed as follows: "*Base URL from 1.x content*" + "/edcws/services/urn:EDCLicenseService".

Upgrading metadata

If a 2.0 client encounters content packaged with Flash Media Rights Management Server 1.x, it will extract the encryption metadata from the content and send it to the server. The server will convert the 1.x metadata into the 2.0 format and send it back to the client. The client then sends the updated metadata in a standard 2.0 license request. The `FMRMSv1MetadataHandler` class is involved in processing these requests.

The request URL is constructed as follows: "*Base URL from 1.x content*" + "/flashaccess/headerconversion/v1".

This conversion could be done on the fly when the server receives the old metadata from the client. Alternatively, the server could preprocess the old content and store the converted metadata; in this case,

when the client requests new metadata, the server just needs to fetch the new metadata matching the license identifier of the old metadata.

To convert metadata, the server must perform the following steps:

- Get `LiveCycleKeyMetaData`. To pre-convert the metadata, `LiveCycleKeyMetaData` can be obtained from a 1.x packaged file using `MediaEncrypter.examineEncryptedContent()`. The metadata is also included in the metadata conversion request (`FMRMSv1MetadataHandler.getOriginalMetadata()`).
- Get the license identifier from the old metadata, and find the encryption key and policies (this information was originally in the Adobe LiveCycle ES database. The LiveCycle ES policies must be converted to Flash Access 2.0 policies.) The Reference Implementation includes scripts and sample code for converting the policies and exporting license information from LiveCycle ES.
- Fill in the `V2KeyParameters` object (which you retrieve by calling `MediaEncrypter.getKeyParameters()`).
- Load the `SigningCredential`, which is the packager credential issued by Adobe used to sign encryption metadata. Get the `SignatureParameters` object by calling `MediaEncrypter.getSignatureParameters()` and fill in the signing credential.
- Call `MetaDataConverter.convertMetadata()` to obtain the `V2ContentMetaData`.
- Call `V2ContentMetaData.getBytes()` and store for future use, or call `FMRMSv1MetadataHandler.setUpdatedMetadata()`.

Handling certificate updates

There might be times when you have to get a new certificate from Adobe. For example, when a production certificate expires, an evaluation certificate expires, or when you switch from an evaluation to a production certificate. When a certificate expires, you do not want to repackage the content that used the old certificate. Instead, you make the license server aware of both the old and new certificates.

Use the following procedure to update your server with the new certificates:

1. Use the Java API to sign any policy update list or revocation list using the new certificate. The following code can be used to re-sign a revocation list using a new certificate. The steps are similar for the policy update list:

```
// Load existing RevocationList
RevocationList oldList =  RevocationListFactory.loadRevocationList(in);
// Verify signature
oldList.verifySignature(oldLicenseServerCertificate);
// Generate a new list that contains all the old entries
RevocationListFactory factory = new
   RevocationListFactory(newLicenseServerCredential, crlNumber);
factory.addRevocationEntries(oldList.getRevocationEntries());
RevocationList newList = factory.generateRevocationList();
```

2. (Optional) Once the policy update list or revocation list has been re-signed, you can add new entries to the list using the command line tools or API, making sure to sign with the new credentials.

   For example, use the following command line to add an entry to an existing policy update list:

```
java -jar AdobePolicyUpdateListManager.jar newList -f oldList -u pol 0 "" ""
```

3. Use the Java API to update the license server with the new policy update list or revocation list:

`HandlerConfiguration.setRevocationList`

or:

`HandlerConfiguration.setPolicyUpdateList`

In the reference implementation, the properties you use are `HandlerConfiguration.RevocationList` and `HandlerConfiguration.PolicyUpdateList`. Also update the certificate used to verify the signatures: `RevocationList.verifySignature.X509Certificate`.

4. To consume content that was packaged using the old certificates, the license server requires the old and new license server credentials and transport credentials. Update the license server with the new and old certificates.

   For the license server credentials:

   - Ensure that the current credential is passed into the LicenseHandler constructor:
     - In the reference implementation, set it through the `LicenseHandler.ServerCredential` property.
     - In the Flash Access Server for protected streaming, the current credential must be the first credential specified in the `LicenseServerCredential` element in the flashaccess-tenant.xml file.
   - Ensure that the current and old credentials are provided to AsymmetricKeyRetrieval
     - In the reference implementation, set it through the `LicenseHandler.ServerCredential` and `AsymmetricKeyRetrieval.ServerCredential.n` properties.
     - In the Flash Access Server for protected streaming, the old credentials are specified after the first credential in the `LicenseServerCredential` element in the flashaccess-tenant.xml file.

   For the transport credentials:

   - Ensure that the current credential is passed into the `HandlerConfiguration.setServerTransportCredential()` method:
     - In the reference implementation, set it through the `HandlerConfiguration.ServerTransportCredential` property.
     - In the Flash Access Server for protected streaming, the current credential must be the first credential specified in the `TransportCredential` element in the flashaccess-tenant.xml file.
   - Ensure that the old credentials are provided to `HandlerConfiguration.setAdditionalServerTransportCredentials`:
     - In the reference implementation, set it through the `HandlerConfiguration.AdditionalServerTransportCredential.n` properties.
     - In the Flash Access Server for protected streaming, this is specified after the first credential in the `TransportCredential` element in the flashaccess-tenant.xml file.

5. Update packaging tools to make sure they are packaging content with the current credentials. Ensure that the latest license server certificate, transport certificate, and packager credential are used for packaging.

Performance tuning

Use the following tips to help to increase performance:

- Using a network HSM can be significantly slower than using a directly-connected HSM.

- You can one of the JAR files in the "thirdparty/jsafe" folder. jsafe.jar is the pure-Java version. Alternatively, you can use jsafeWithNative.jar, which has better performance, but also requires that platform specific libraries (for example, jsafe.dll for Microsoft® Windows® or libjsafe.so for Linux®) be added to the path.
- A 64-bit operating system, such as the 64-bit version of Red Hat® or Windows, provides much better performance over a 32-bit operating system.

  Note: If you use a 64-bit version of Windows, HSM is currently not supported.

6 Revoking credentials and policies

Under certain conditions it is necessary to revoke a client's credentials or check whether a given set of credentials have already been revoked. Credentials may be revoked if the credentials are compromised. When this happens, licenses will no longer be issued to compromised clients.

Adobe maintains Certificate Revocation Lists (CRLs) for revoking compromised clients. These CRLs are automatically enforced by the SDK. License servers may further restrict clients by disallowing particular machine credentials or particular versions of DRM and runtime credentials. A `RevocationList` may be created and passed into the SDK to revoke machine credentials. Particular DRM/runtime versions can be revoked either at the policy level (by setting module restrictions in the play right) or globally (by setting module restrictions in the `HandlerConfiguration`).

Policies may also be revoked if content owners or distributors want to discontinue issuing licenses under a particular policy. A policy update list may be used to enforce policy revocation in the SDK. Policy update lists may also be used to provide a list of updated policies to the SDK. Note that revoking a policy does not revoke licenses that have already been issued. It only prevents additional licenses from being issued under that policy.

The discussion in this chapter will be centered on revoking client credentials and working with policy update lists.

All of these tasks can be accomplished using the Java API. For details about the Java API discussed in this chapter, see *Adobe Flash Access API Reference*.

Certificate Revocation Lists published by Adobe

The SDK automatically enforces the CRLs published by Adobe. The SDK fetches the CRLs from Adobe and caches them locally. There are some system properties that control how the CRLs are cached. To modify these properties, use the -D option when starting Java to specify the property name and value.

By default, the CRLs will be saved to disk in the directory specified by "java.io.tmpdir" (if the temp dir property is not set, the working directory is used). The server must have read/write access to this directory. To use a directory other than the temp directory, specify a system property called "flashaccess.crl.dir" whose value is the directory where the CRLs should be stored.

By default, if a CRL cannot be retrieved, this is treated as an error. To ignore missing CRLs in a development environment, set the Java System property "flashaccess.crl.error=ignore". This property must not be set for production environments.

If an error occurs fetching the CRL, by default the SDK will not try to fetch the CRL again for 5 minutes. The duration can be configured by setting the number of minutes in the "flashaccess.crl.retry" System property. If the value is less than 1, the server will not wait before retrying (not recommended).

If a cached CRL is expiring soon, the SDK will attempt to fetch it again from the CRL server. By default, the SDK will try to fetch the CRL 15 days before its expiration. This duration is configurable by setting the number of minutes in the "flashaccess.crl.prefetch" system property. If the value is less than 1, the server will not try to pre-fetch the CRL (not recommended for production environments).

Revoking DRM client and runtime credentials

DRM/Runtime versions are identified by security level, version number, and other attributes including OS and runtime. To restrict the DRM/Runtime versions allowed, set the module restrictions in a policy or in a `HandlerConfiguration`. Module restrictions may include a minimum security level and list of module versions that are not permitted to be issued a license.

If the minimum security level is set, the version on the client (specified in the machine token), must be greater than or equal to the specified value.

If a list of excluded versions is specified and the client's version matches any of the version identifiers in the list, the client will not be allowed to use a right containing this ModuleRequirements instance. For a module to match the version information, all parameters specified in the version information, except for the release version, must exactly match the modules's values. The release version matches if the client module's value is less than or equal to the value in the version information.

In the event a breach is reported with a particular DRM client or runtime version, the content owner and content distributer (who runs the license server) can configure the server to refuse to issue licenses during a period in which Adobe does not have a fix available. This can be configured through the `HandlerConfiguration` as described above, or by making changes to all the policies. In order to stop distributing licenses to those clients. In the latter case, you can maintain a policy update list and use it to check whether a policy has been updated or revoked.

If you require a newer version of the Adobe® Flash® Player/Adobe® AIR® Runtime or the Adobe Content Protection library, update your policies as shown in Updating a policy using the Java API and create a Policy Update List, or set restrictions in HandlerConfiguration by invoking `HandlerConfiguration.setRuntimeModuleRequirements()` or `HandlerConfiguration.setDRMModuleRequirements()`. When a user requests a new license with these blacklists enabled, the newer runtimes and libraries must be installed before a license can be issued. For an example on blacklisting DRM and runtime versions, see the sample code in Updating a policy using the Java API.

Revoking machine credentials

Adobe maintains a CRL for revoking machine credentials that are known to be compromised. This CRL is automatically enforced by the SDK. If there are additional machines to which you do not want your license server to issue licenses, you may create a machine revocation list and add the Issuer name and serial number of the machine tokens you want to exclude (use `MachineToken.getMachineTokenId()` to retrieve the issuer name and serial number of the machine certificate).

Revoking machine credentials involves the usage of a `RevocationListFactory` object. To create a revocation list, load an existing revocation list, and check whether a machine token has been revoked by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.

2. Create a `ServerCredentialFactory` instance to load the credentials needed for signing. The license server credential is used to sign the revocation list.

3. Create a `RevocationListFactory` instance.

4. Specify the issuer and serial number of the machine token to be revoked by using a `IssuerAndSerialNumber` object. All authentication and license requests contain a machine token.

5. Create a `RevocationList` object using the `IssuerAndSerialNumber` object you just created, and add it to the revocation list by passing it into `RevocationListFactory.addRevocationEntry()`. Generate the new revocation list by calling `RevocationListFactory.generateRevocationList()`.

6. To save the revocation list, you can serialize it by calling `RevocationList.getBytes()`. To load the list, call `RevocationListFactory.loadRevocationList()` and pass in the serialized list.

7. Verify that the signature is valid and the list was signed by the correct license server by calling `RevocationList.verifySignature()`.

8. To check whether an entry was revoked, pass the `IssuerAndSerialNumber` object into `RevocationList.isRevoked()`. The revocation list may also be passed into `HandlerConfiguration` to have the SDK enforce the revocation list for all authentication and license requests.

To add additional entries to an existing RevocationList, load an existing revocation list. Create a new RevocationListFactory instance, and be sure to increment the CRL number. Call `RevocationListFactioryEntries.addRevocationEntries` to add all the entries from the old list to the new list. Call `RevocationListFactory.addRevocationEntry` to add any new revocation entries to the RevocationList.

The following Java file shows how to create a revocation list, load an existing revocation list, and check whether a machine token has been revoked.

Example: *Creating and parsing a revocation list using the Java API*

```
/**************************************************************************
 *
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 **************************************************************************/
package com.adobe.fmrms.samples.revocation;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.math.BigInteger;
import java.util.Date;

import com.adobe.flashaccess.sdk.cert.IssuerAndSerialNumber;
import com.adobe.flashaccess.sdk.cert.ServerCredential;
import com.adobe.flashaccess.sdk.cert.ServerCredentialException;
import com.adobe.flashaccess.sdk.cert.ServerCredentialFactory;
import com.adobe.flashaccess.sdk.revocation.RevocationEntry;
import com.adobe.flashaccess.sdk.revocation.RevocationException;
```

```
import com.adobe.flashaccess.sdk.revocation.RevocationList;
import com.adobe.flashaccess.sdk.revocation.RevocationListFactory;

/**
 * Demonstrates how to create a revocation list,
 * how to load an existing revocation list,
 * and how to check if a machine token has been revoked.
 */
public class CreateRevocationList
{
  public static void main( String args[] )
  {
    // Specify the file containing the license server credentials issued by
Adobe.
    // These credentials are used to sign the revocation list.
    File licSvrCredentialFile = new File("C:/licenseServerCredentials.pfx");

    // Set the password needed to open the certificate file.
    String licSvrCredentialPassword = "password";

    try {
      // Load the credential for signing.
      ServerCredential signingCred =
ServerCredentialFactory.getServerCredential(
        licSvrCredentialFile,
         licSvrCredentialPassword
      );
      RevocationListFactory rlFactory = new RevocationListFactory(
        signingCred,
        1
      );

      // Set the issuer and serial number of the machine token to be revoked.
      IssuerAndSerialNumber toRevoke = new
IssuerAndSerialNumber("E=*.adobe.com", BigInteger.valueOf(1));

      // Create the revocation entry and add it to the revocation list that
will be generated.
      RevocationEntry crlEntry = new RevocationEntry(toRevoke, new Date());
      rlFactory.addRevocationEntry(crlEntry);

      // Generate the new revocation list.
      RevocationList newRevList = rlFactory.generateRevocationList();

      // Serialize the revocation list.
      byte[] revListBytes = newRevList.getBytes();

      // Given a revocation list, verify its signature.
      RevocationList revList = RevocationListFactory.loadRevocationList(new
ByteArrayInputStream(revListBytes));
      revList.verifySignature(signingCred.getCertificate());

      // Check if the entry was revoked.
      boolean isRevoked = revList.isRevoked(toRevoke);
```

```
        } catch (ServerCredentialException e) {
            // There was a problem loading the license server credential.
            e.printStackTrace();

        } catch (RevocationException e) {
            // There was a problem creating or reading the revocation list.
            e.printStackTrace();
        }
    }
}
```

Working with Policy Update Lists

For license servers that do not have access to a database for storing information about policies, you may wish to use a policy update list to notify the license server of updated policies. Policy Update Lists may contain updated versions of policies or a list of policy IDs that have been revoked. If a policy update list is supplied in `HandlerConfiguration`, the SDK will enforce this list when issuing a license.

Working with policy update lists involves the use of a `PolicyUpdateListFactory` object. To create a policy update list, load an existing policy update list, and check whether a policy has been updated or revoked by using the Java API, perform the following steps:

1. Set up your development environment and include all of the JAR files mentioned in Setting up the development environment within your project.
2. Create a `ServerCredentialFactory` instance to load the credentials needed for signing.
3. Create a `PolicyUpdateListFactory` instance using the `ServerCredential` you created.
4. Specify the policy ID to be revoked.
5. Create a `PolicyRevocationEntry` object using the policy ID `String` you just created, and add it to the policy update list by passing it into `PolicyUpdateListFactory.addRevocationEntry()`. Generate the new policy update list by calling `PolicyUpdateListFactory.generatePolicyUpdateList()`. Similarly, updated policies can be added to the list using `PolicyUpdateEntry`.
6. If a policy update list already exists, you can serialize it for loading by calling `PolicyUpdateList.getBytes()`. To load the list, call `PolicyUpdateListFactory.loadPolicyUpdateList()` and pass in the serialized list.
7. Verify the signature is valid and the list was signed by the correct license server certificate by calling `PolicyUpdateList.verifySignature()`.
8. To check whether an entry was revoked, pass the policy ID `String` into `PolicyUpdateList.isRevoked()`. Alternatively, the list can be passed into `HandlerConfiguration` and it will be enforced when licenses are issued.

To add additional entries to an existing PolicyUpdateList, load an existing policy update list. Create a new PolicyUpdateListFactory instance. Call `PolicyUpdateListFactory.addEntries` to add all the entries from the old list to the new list. Call `PolicyUpdateListFactory.addRevocationEntry` or addUpdatedEntry to add any new revocation or update entries to the PolicyUpdateList.

The following Java file shows how to create a policy update list, load an existing policy update list, and check whether a policy has been revoked.

Example: *Creating and parsing a policy update list using the Java API*

```
/**************************************************************************
 *
 * ADOBE SYSTEMS INCORPORATED
 *  Copyright 2009 Adobe Systems Incorporated
 *  All Rights Reserved.
 *
 * NOTICE:  Adobe permits you to use, modify, and distribute this file in
 * accordance with the terms of the Adobe license agreement accompanying it.
 * If you have received this file from a source other than Adobe, then your use,
 * modification, or distribution of it requires the prior written permission of
 * Adobe.
 **************************************************************************/
package com.adobe.fmrms.samples.policyupdatelist;

import com.adobe.flashaccess.sdk.cert.ServerCredential;
import com.adobe.flashaccess.sdk.cert.ServerCredentialException;
import com.adobe.flashaccess.sdk.cert.ServerCredentialFactory;
import com.adobe.flashaccess.sdk.policyupdate.PolicyRevocationEntry;
import com.adobe.flashaccess.sdk.policyupdate.PolicyUpdateList;
import com.adobe.flashaccess.sdk.policyupdate.PolicyUpdateListException;
import com.adobe.flashaccess.sdk.policyupdate.PolicyUpdateListFactory;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.util.Date;

/**
 * Demonstrates how to create a policy update list,
 * how to load an existing policy update list,
 * and how to check if a policy has been updated or revoked.
 */
public class CreatePolicyUpdateList
{
   public static void main( String args[] )
   {
      // Specify the file containing the license server credentials issued by
Adobe.
     // These credentials are used to sign the policy update list.
     File licSvrCredentialFile = new File("C:/licenseServerCredentials.pfx");

     // Set the password needed to open the certificate file.
     String licSvrCredentialPassword = "password";

     try {
        // Load the credential for signing.
        ServerCredential signingCred =
ServerCredentialFactory.getServerCredential(
           licSvrCredentialFile,
            licSvrCredentialPassword
        );
```

```
        PolicyUpdateListFactory pulFactory = new
PolicyUpdateListFactory(signingCred);

        // Set the ID of the policy to revoke.
        String toRevoke = "A723329C-4D35-3080-8F19-35F32F5B552B";

        // Create the revocation entry and add it to the policy update list to
be generated.
        PolicyRevocationEntry revokedEntry = new PolicyRevocationEntry(
          toRevoke,
          new Date(),
           PolicyRevocationEntry.UNDEFINED_REVOCATION_REASON,
           null,
           null
        );
        pulFactory.addRevokedEntry(revokedEntry);

        // If policies have been updated, the full updated policies can also be
        // added to the policy update list using pulFactory.addUpdatedEntry

        // Generate the policy update list.
        PolicyUpdateList newPolUpdateList =
pulFactory.generatePolicyUpdateList();

        // Serialize the policy update list.
        byte[] pulBytes = newPolUpdateList.getBytes();

        // Given a PolicyUpdateList, verify its signature.
        PolicyUpdateList polUpdateList =
PolicyUpdateListFactory.loadPolicyUpdateList(ByteArrayInputStream(pulBytes))
;
        polUpdateList.verifySignature(signingCred.getCertificate());

        // Check if the entry was revoked.
        boolean isRevoked = polUpdateList.isRevoked(toRevoke);

        // To check if an updated version of a policy is available, use
polUpdateList.isUpdated(policyID)

     } catch (ServerCredentialException e) {
        // There was a problem loading the license server credential.
        e.printStackTrace();

     } catch (PolicyUpdateListException e) {
        // There was a problem creating or reading the revocation list.
        e.printStackTrace();
     }
   }
}
```

7 Creating video players

In order to play back protected content, your application must use the ActionScript 3 DRM APIs. Please refer to Programming ActionScript 3 and the ActionScript 3.0 Reference for the Adobe Flash Platform for more information.

8 Using the Flash Access Server for Protected Streaming

The Adobe® Flash® Access™ Server for Protected Streaming is a license server implementation based on the Flash Access SDK. This server issues licenses for protected content to Flash Access clients.

The Flash Access Server for Protected Streaming supports multiple tenants, meaning a single server can be hosted on behalf of multiple content publishers, each with its own configuration settings. In addition, the server supports custom authorization components, so custom logic can be executed before issuing a license.

This server is intended for use with HTTP Streaming. As a result, this server implementation does not support all of the capabilities available in Flash Access. For example, the Flash Access Server for Protected Streaming does not support user authentication requests or multiple policies.

Usage rules

With the Flash Access Server for Protected Streaming, all usage rules are specified on the server through configuration files. Any usage rules specified in the protected content are ignored, so it is recommended to use a simple anonymous policy during content packaging. Usage rules that are configurable can be set on a per-tenant basis.

The Flash Access Server for Protected Streaming supports the following usage rules:

- Output Protection
- Adobe® AIR® and SWF Application Restrictions
- DRM and Runtime Module Restrictions
- License Caching is disabled by default. License caching can be enabled by specifying the caching end date or an amount of time caching is allowed (starting when the license is issued).
- Multiple Play Rights, which lets you specify different combinations of Output Protection, Application Restrictions, and DRM/Runtime Restrictions. For example, it is possible to specify different Output Protection requirements for each client platform by using the DRM Module Restriction with Output Protection.

The following usage rules are fixed:

- Anonymous access
- Licenses are valid for 24 hours, starting when the license is issued
- No license chaining support
- No playback window support
- No custom property support

Requirements

- Microsoft Windows Server 2003 or Red Hat® Enterprise Linux®
- Java JRE 1.6 (Java JDK 1.6 is required to create custom authorization extensions)

- Apache Tomcat® 6 (Available in the Third Party\Tomcat\6.0.18 folder of the DVD)
- Flash Access Server for Protected Streaming (Available in the Flash Access Server for Protected Streaming folder on the DVD)
- Credentials issued by Adobe

Deploying the Flash Access Server for Protected Streaming

Before deploying the license server, make sure you have installed the versions of Java and Tomcat listed in the Requirements section.

The Flash Access Server for Protected Streaming download includes flashaccesserver.war. To deploy this WAR file, copy it to Tomcat's webapps directory. If you have previously deployed the WAR file, you may need to manually delete the unpacked WAR directory ("flashaccessserver" in Tomcat's webapps directory). To prevent Tomcat from unpacking WAR files, edit the server.xml file in Tomcat's conf directory and set the "unpackWARs" attribute to "false".

The server requires a platform-specific library (jsafe.dll on Microsoft Windows or libjsafe.so on Linux). Copy the appropriate library for your platform from thirdparty/jsafe/*platform* to a location specified by the PATH environment variable (or LD_LIBRARY_PATH on Linux). Note: the 64-bit version should only be used if both the operating system and JDK support 64-bit, otherwise use the 32-bit version.

Java system properties

The following two Java System properties may optionally be set to modify the location of configuration and log files for the license server:

- LicenseServer.ConfigRoot — Directory containing all the configuration files for the license server. For details on the contents of these files, see "License server configuration files" on page 45. If not set, the default is *CATALINA_BASE/licenseserver*.
- LicenseServer.LogRoot — Directory of the "logs" folder, where license server application logs are written. If not set, the default is *LicenseServer.ConfigRoot*.

If you are using catalina.bat or catalina.sh to start Tomcat, these System properties can easily be set using the JAVA_OPTS environment variable. Any Java options set here will be used when Tomcat is started. For example, set:

```
JAVA_OPTS=-DLicenseServer.ConfigRoot="absolute-path-to-config-folder"
   -DLicenseServer.LogRoot="absolute-path-to-log-folder"
```

Flash Access credentials

To issue valid licenses accepted by a Flash Access client, the Flash Access Server for Protected Streaming must be configured with a set of credentials issued by Adobe. These credentials can either be stored in PKCS#12 (.pfx) files or on an HSM.

The .pfx files may be located anywhere, but for ease of configuration, we recommend placing the .pfx files in the tenant's configuration directory. For more information, see "License server configuration files" on page 45.

HSM configuration

If you choose to use an HSM to store your server credentials, you must load the private keys and certificates onto the HSM and create a pkcs11.cfg configuration file. This file must be located in the *LicenseServer.ConfigRoot* directory. See the configs directory for an example PKCS11 configuration file. For information on the format of pkcs11.cfg, see the Sun PKCS11 provider documentation.

To verify that your HSM and Sun PKCS11 configuration file are configured properly, you can use the following command from the directory where the pkcs11.cfg file is located (keytool is installed with the Java JRE and JDK):

```
keytool -keystore NONE -storetype PKCS11
   -providerClass sun.security.pkcs11.SunPKCS11 -providerArg pkcs11.cfg -list
```

If you see your credentials in the list, the HSM is configured properly and the license server will be able to access the credentials.

License server configuration files

The Flash Access Server for Protected Streaming requires two types of configuration files: a global configuration file (flashaccess-global.xml) and a tenant configuration file for each tenant (flashaccess-tenant.xml).

After editing the configuration files, Adobe recommends using the utilities provided with the Flash Access Server for Protected Streaming to verify that the files are well-formed. For more information, see "Configuration Validator" on page 51.

To avoid making passwords available in clear text in the configuration files, encrypt all passwords specified in the global and tenant configuration files. For more information on encrypting passwords, see "Password Scrambler" on page 51.

Configuration Directory Structure

The configuration directories have the following structure:

```
LicenseServer.ConfigRoot/
   flashaccess-global.xml
   pkcs11.cfg  (optional)
   flashaccessserver/
      libs/ (optional)
      tenants/
         tenantname/
            flashaccess-tenant.xml
            credential.pfx  (optional)
            packagercert.cer (optional)
```

Global configuration file

The flashaccess-global.xml configuration file contains settings that apply to all tenants of the license server. This file must be located in *LicenseServer.ConfigRoot*. See the configs directory for an example global configuration file. The global configuration file includes the following:

- Caching — Controls caching of config files in memory. For an explanation of the caching settings, see "Updating configuration files" on page 49.

- Logging — Specifies the logging level and how frequently log files are rolled.
- HSM password — Required only if an HSM is used to store server credentials.

See the comments in the example global configuration file for more details.

Tenant configuration file

The flashaccess-tenant.xml configuration file contains settings that apply to a specific tenant of the license server. Each tenant has its own instance of this configuration file located in *LicenseServer.ConfigRoot*/flashaccessserver/tenants/*tenantname*. See the configs/flashaccessserver/tenants/sampletenant directory for an example tenant configuration file.

You can specify all file paths in the tenant configuration file as absolute paths or paths relative to the tenant's configuration directory (*LicenseServer.ConfigRoot*/flashaccessserver/tenants/*tenantname*).

The tenant configuration file includes:

- Transport Credential — Specifies one or more transport credentials (certificate and private key) issued by Adobe. Can be specified as a path to a .pfx file and a password, or an alias for a credential stored on an HSM. Several such credentials can be specified here, either as file paths, or key aliases, or both. See "Handling certificate updates" on page 32 for more information on when additional credentials are needed.
- License Server Credential — Specifies one or more license server credentials (certificate and private key) issued by Adobe. Can be specified as a path to a .pfx file and a password, or an alias for a credential stored on an HSM. Several such credentials can be specified here, either as file paths, or key aliases, or both. See "Handling certificate updates" on page 32 for more information on when additional credentials are needed.
- Custom Authorizers — Optional. Specifies custom authorizer classes to invoke for each license request. If multiple authorizers are specified, they are invoked in the order listed. For more information, see "Custom authorization extensions" on page 47.
- List of Authorized Packagers — Optional. Specifies certificates identifying entities authorized to package content for this license server. If no packager certificates are specified, the server issues licenses for content packaged by any packager.
- Usage Rules
  - License Caching — Optional. Specifies how long the license can be stored on the client. By default license caching is disabled. To enable license caching for a limited time period, set the end date or the number of seconds for which the license should be stored (starting when the license is issued). Setting the number of seconds to 0 disables license caching.

    Note that all licenses issued by the protected streaming license server have an expiration period of 24 hours (86400 seconds). This value therefore applies implicitly as an upper bound to whatever end date or duration is set for license caching as well, with a maximum value of 86400 seconds, even though the schema enforces higher bounds.
  - Play Right — At least one right must be specified. If multiple rights are specified, the client will use the first right for which it meets all the requirements.
    - Output Protection — Controls whether output to external rendering devices should be protected.
    - AIR and SWF Application Restrictions — Optional whitelist of SWF and AIR applications that may play the content (i.e. only the applications specified are permitted). SWF applications are identified by a URL or by the digest of the SWF and the maximum time to allow for download

and verification of the digest. For information on calculating the SWF digest, see the SWF Hash Calculator section of Flash Access Server Utilities. AIR applications are identified by a publisher ID and optional application ID, minimum version, and maximum version. If no application restrictions are specified, any SWF or AIR application may play the content.

- DRM and Runtime Module Restrictions — Specifies the minimum security level required for the DRM/Runtime module. Optionally includes a blacklist of versions that are not permitted to play the content. Module versions are identified by attributes such as operating system and/or a version number.

See the comments in the example tenant configuration file for more details.

Cross-domain policy file

If the license server is hosted on a different domain than the video playback SWF, then a cross-domain policy file (crossdomain.xml) might be necessary to allow the SWF to request licenses from the license server. A cross-domain policy file is an XML file that provides a way for the server to indicate that its data and documents are available to SWF files served from other domains. Any SWF file that is served from a domain that the server's cross-domain policy file specifies is permitted to access data or assets from that server.

Adobe recommends that developers follow best practices when deploying the cross-domain policy file by only allowing trusted domains to access the license server and limiting the access to the license sub-directory on the web server. For more information on cross-domain policy files, please see the following locations:

- Website controls (policy files)
- Cross-domain policy file specification

Custom authorization extensions

Custom authorization logic may be invoked during license acquisition to decide if a license should be issued to the requesting client.

To implement your own customer authorization extension, first look at the SampleAuthorizer.java sample code located in the samples directory (the compiled version of this sample is located in flashaccess-license-server-ext-sample.jar).

To build your own extension, implement the `com.adobe.flashaccess.server.license.extension.auth.IAuthorizer` interface and make sure flashaccess-license-server-exts.jar and commons-logging.jar are on the build path (adobe-flashaccess-sdk.jar must also be on the build path if you utilize certain fields in `IMessageFacade`). To deploy your extension, copy your jar or class files to *LicenseServer.ConfigRoot*/flashaccessserver/libs. If you need to update the jar or class files, the server must be restarted before the updated version is used. You also must add the authorizer class name to the tenant configuration file.

Performance tuning

This section outlines performance-related considerations.

Global Configuration File

The largest impact to performance that you can make is by using settings in the global configuration file, flashaccess-global.xml. These settings include the `<Caching>` and `<Logging>` elements.

- `<Caching>`

  The `<Caching>` element controls caching of configuration files in memory. The `<Caching>` element has the following syntax:

  ```
  <Caching refreshDelaySeconds="..." numTenants="..."/>
  ```

  - `refreshDelaySeconds` controls how often the server checks for updates to the configuration files. A low value for `refreshDelaySeconds` negatively impacts performance, while a higher value can improve performance. For more information on `refreshDelaySeconds`, see "Updating configuration files" on page 49.
  - `numTenants` specifies the number of tenants. A value that is lower than the number of tenants likely impacts performance because requests to the remaining tenants result in cache misses. A cache miss for configuration data negativley impacts performance. Therefore, Adobe recommends that you set this value higher than the number of tenants configured for the server, unless there are memory limitations to consider.

- `<Logging>`

  The `<Logging>` element specifies the logging level and how frequently log files are rolled. The `<Logging>` element has the following syntax:

  ```
  <Logging level="..." rollingFrequency=""/>
  ```

  - `level` specifies the messages to log. A value of "DEBUG" yields a lot of log messages, and can negatively impact performance. Adobe recommends a setting of "WARN" for optimal performance. However, that value does risk losing essential runtime information, such as license audits. To preserve valuable log information with minimal performance impact, use a value of "INFO".
  - `rollingFrequency` specifies how often log files are *rolled*. Rolling is the process where a new log file becomes the active log, while the previously active log file is no longer written to and is considered rolled. The rolling interval can be set to "MINUTELY", "HOURLY", "TWICE-DAILY", "DAILY", "WEEKLY", "MONTHLY", or "NEVER".

Deploying on a 64-bit operating system

A 64-bit operating system, such as the 64-bit version of Red Hat® or Microsoft® Windows®, provides much better performance over a 32-bit operating system.

Note: If you use a 64-bit version of Windows, HSM is currently not supported.

Running the License Server

Before running the license server, Adobe recommends that you verify that the configuration files are valid by using the utilities provided with the license server. For more details, see "Configuration Validator" on page 51.

To start Tomcat and the license server, run "catalina.bat start" or "catalina.sh start" from Tomcat's bin directory.

After the server has started, verify that it is configured properly by opening http://license-server-host:port/flashaccessserver/tenant-name/flashaccess/license/v1 in a browser window. If the tenant configuration was successfully loaded, a confirmation message is displayed.

Log files

The log files generated by the Flash Access Server for Protected Streaming application will be located in the directory specified by LicenseServer.LogRoot. Note: if the current log files are deleted or moved while the server is running, the log file may not be re-created, and some log information will be lost.

Log directory structure

The log directory has the following structure:

LicenseServer.LogRoot/
flashaccess-global.log
  flashaccessserver/
    flashaccess-partition.log
    tenants/
      tenantname/
        flashaccess-tenant.log

Global Log File

The global log file, flashaccess-global.log, is located in *LicenseServer.LogRoot*. This log can contain log messages generated by the Flash Access SDK or log messages generated during server initialization.

Partition Log File

The partition log file, flashaccess-partition.log, is located in *LicenseServer.LogRoot*/flashaccesserver. This log contains log messages generated during processing of license request.

Tenant Log File

Each tenant's tenant log file, flashaccess-tenant.log, is located in *LicenseServer.LogRoot*/flashaccesserver/tenants/tenantname. The tenant log contains audit information describing each license generated for this tenant.

Updating configuration files

Once the license server reads one of the license server configuration files (global or tenant configuration), the configuration information is cached in memory. Therefore, the files do not have to be read from disk for every license request. However, the server also allows most values in the configuration files to be modified without requiring a server restart for the changes to take effect. (See below for details on which configuration values are checked for updates.)

In order to reload the configuration when changes are made, the license server stores the time the file was last modified. At a configurable interval, the server checks if the file modification time has changed, and if so, reloads the contents of the file.

To control how often the server checks for updates, set the "refreshDelaySeconds" attribute in the Caching element of the global configuration file. For example, if "refreshDelaySeconds" is set to 3600 seconds, it takes at most one hour from the time the file is updated for any configuration updates to be detected by the server. If "refreshDelaySeconds" is set to 0, the server checks for configuration updates on every request. Setting "refreshDelaySeconds" to a low value is not recommended for production environments, as it could impact performance.

The Caching element also controls how many tenants' configurations are cached at once. You can set this value to a number smaller than the total number of tenants to limit the amount of memory used to cache the configuration information. If a request is received for a tenant not in the cache, the configuration is loaded before the request can be processed. If the cache is full, the least recently used tenant is removed from the cache.

If a change is saved to a configuration file or to any of the certificate files referenced within flashaccess-tenant.xml while the server is attempting to read the file, or if the file's timestamp is found to be less than one second before the current time or is in the future, the cached version of the configuration is used until the next time the server checks for updates. If there is no cached version, the loading of the configuration fails, and an error is returned to the client. The server attempts to load the file again the next time it receives a request for that tenant.

Updating the Global Configuration File

The HSM password in flashaccess-global.xml can be modified at any time, and the changes take effect the next time the server reloads the configuration file. However, changes to the "Logging" and "Caching" elements are not reloaded; any changes in these elements require a server restart.

Updating the Tenant Configuration File

All values specified in flashaccess-tenant.xml can be modified at any time, and the changes take effect the next time the server reloads the configuration file. Also, the server checks for changes in all credential (.pfx) files and packager whitelist certificate files referenced in the tenant configuration file.

Packaging content

When packaging content, the license server URL must be specified. The Flash Access Server URL has the format:

`http(s)://`*`license-server-host:port`*`/flashaccessserver/`*`tenant-name`*

For example, for license server hostname "mylicenseserver.com" listening on port 8080 and a tenant named "tenant1", the license server URL to specify at packaging time is:

```
http://mylicenseserver.com:8080/flashaccessserver/tenant1
```

If each tenant uses a different License Server and Transport Credential, be sure to specify the correct tenant's certificate in the packager.

To ensure the server issues licenses only to content packaged by known packagers, include the packager's certificate in the packager whitelist of the tenant configuration file.

Flash Access Server for Protected Streaming utilities

Configuration Validator

Adobe recommends running the Configuration Validator utility before starting the server any time changes are made to the configuration file. This utility can detect most configuration errors early, before they cause failures during request processing.

To run the validator, use the command:

```
Validator.bat options
```

or the command:

```
java -jar libs/flashaccess-validator.jar options
```

For each of the license server configuration files, the Validator can perform file-based validation, which ensures the XML file is well-formed and conforms to the configuration file schema. To perform file-based validation on the global configuration file, run the command:

```
Validator --file path/flashaccess-global.xml --global
```

To perform file-based validation on the tenant configuration file, run the command:

```
Validator --file path/flashaccess-tenant.xml --tenant
```

The Validator can also perform deployment-based validation; in addition to checking conformity with the schema, this level of validation also checks that the values specified are valid (for example, it ensures that referenced files exist). Deployment-based validation can be performed at two levels:

- Tenant — Validates configuration file and credentials for a specific tenant. To validate the configuration for "tenant1", run the command:

  ```
  Validator --root-path-to-LicenseServer.ConfigRoot
     -d flashaccessserver/tenant1 -t
  ```

- Global — Validates the global configuration file and tenant validation for all tenants. To perform global deployment-based validation, run the command:

  ```
  Validator --root-path-to-LicenseServer.ConfigRoot -g
  ```

Password Scrambler

The Password Scrambler utility encrypts a password so that it can be used in the Flash Access Server for Protected Streaming configuration files. To run the scrambler, run the command:

```
Scrambler.bat password
```

or the command:

```
java -jar libs/flashaccess-scrambler.jar password
```

The utility outputs the following message:

```
Encrypted password:  scrambled-password
```

All passwords specified in flashaccess-global.xml and flashaccess-tenant.xml must be encrypted.

SWF Hash Calculator

The SWF Hash Calculator utility calculated the digest of a SWF application located in a file. To run the hasher, run the command:

```
Hasher.bat filename.swf
```

or the command:

```
java -jar libs/flashaccess-hasher.jar filename.swf
```

The utility output the following message:

```
SWF Hash:  hash-of-swf
```

This value can be used to specify the SWF digest in flashaccess-tenant.xml.

9 Using the reference implementations

The Flash Access product comes with a reference implementation for the following components:

- "Command line tools for packaging content and creating revocation lists" on page 53
- "License server and watched folder packager" on page 64
- "Adobe Flash Access Manager AIR application usage" on page 73

Note: You should deploy either the Flash Access Server for Protected Streaming, the reference implementation, or your own license server.

Command line tools for packaging content and creating revocation lists

The reference implementation includes the following command line tools:

- Policy Manager: A tool for creating and managing policies
- Media Packager: A tool for creating encrypted FLV and F4V files
- Policy Update List Manager: A tool for creating and viewing policy update lists
- Revocation List Manager: A tool for creating and viewing revocation lists

Requirements

The requirements for using the command line tools available in the reference implementations are as follows:

- All of the command line tools require Java 1.5 or higher.
- Packager and License Server credentials (certificate and password) that are issued by Adobe. You need credentials to encrypt and sign video files and to sign Policy Update and Revocation lists.
- A 32-bit operating system. The tools are not officially supported on 64-bit operating systems (although they may work).

Note: Because of a Java bug, arguments that are used on the command line, such as file names or policy names or descriptions, must use characters only from the operating system's default character set.

Configuration file

The command-line tools require a configuration file that contains information for the tools to use to apply policies and encrypt files.

The default configuration file is flashaccesstools.properties and is located in the working directory; that is, the directory from which you run the tools (see Installing the command line tools). Each tool also contains an option (-c) that lets you point to the configuration file you want to use if you prefer not to use the default.

The configuration file uses the Java property file format. If values for any of the properties contain special characters, keep in mind the following restrictions:

- Escape backslashes with an additional backslash. For example, to specify the C:\credentials.pfx file, specify it as C:\\credentials.pfx or C:\credentials.pfx. To specify a file on a network server, specify \\\\server\\folder\\filename.pfx.
- The configuration file can contain only Latin-1 characters. If you must use non-Latin-1 characters, use the appropriate Unicode escape sequence (using, optionally, the native2ascii tool that comes with Java).

Set values for properties in the configuration file before you run the tools. For some of the command line tools, you can set the values for some properties through either the command line or the configuration file. In those cases, values that are set through the command line take precedence over any values in the configuration file.

Installing the command line tools

You can copy the files you need from the \Reference Implementation\Command Line Tools directory on the DVD, which contains the default flashaccesstools.properties configuration file, and a libs directory, which contains the JAR files for the tools. The samples directory contains all the sample code in this document, as well as additional samples. To build and run the samples, use the build-samples.xml Ant script.

Policy Manager

Using Policy Manager, you can create and manage policies. Before you run Policy Manager, set values for Policy Manager properties in the configuration file. The configuration file specifies information that will be applied to all policies. All Policy Manager properties may also be specified on the command line.

Configuration file properties

The configuration file specifies the following properties. For property names that include *n*, *n* represents an integer starting with 1 and increasing for each instance of the property.

Property	Description
policy.name	The human-readable policy name.
policy.useRootLicense	Indicates whether this policy has a root license (see License Chaining in the Usage Rules section of this document).
policy.startDate	The date after which content is valid. Use the format *yyyy-mm-dd* (for example, 2009-01-31 represents January 31 at 12:00 AM) or *yyyy-mm-dd-h24:min:sec* (for example, 2009-01-31-14:30:00 represents January 31 at 2:30 PM).
policy.expiration.endDate	The date before which content is valid. Both policy.expiration.endDate and policy.expiration.duration may not be specified concurrently. Use the format *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec* (for example, 2009-01-31-14:30:00 represents January 31 at 2:30 PM).

Property	Description
policy.expiration.duration	The amount of time the content is valid (in minutes), starting from when it is packaged. Both policy.expiration.endDate and policy.expiration.duration may not be specified at the same time.
policy.licenseCaching.duration	Amount of time a license may be cached on the client (in minutes). Set this property to 0 to disallow license caching. The value must be 0 or higher. Both policy.licenseCaching.duration and policy.licenseCaching.endDate may not be used concurrently.
policy.licenseCaching.endDate	The date after which licenses may not be cached. Both policy.licenseCaching.duration and policy.licenseCaching.endDate may not be used concurrently.
policy.anonymous	Indicates whether anonymous license acquisition is allowed. The default is "false" (username/password authentication is required) if not specified.
policy.authNamespace	If username/password authentication is required, this property specifies an optional name qualifier for user names.
policy.customProp.*n*	Custom name/value pairs to be used by the server during license acquisition. Use the following format for specifying properties: policy.customProp.*n*=*name*=*value*
policy.playbackWindow	Specifies the playback window (in minutes), which is the duration for which the license is valid after the first time it is used to play protected content.
policy.outputProtection.analog or policy.outputProtection.digital	Output protection constraints. Values must be one of the following: `NO_PROTECTION, USE_IF_AVAILABLE, REQUIRED, NO_PLAYBACK`
policy.drmMinSecurityLevel	The DRM module must have the specified minimum security level, or higher, to access protected content.
policy.drmVersionBlacklist.*n*	A list of versions of DRM modules that may not be used (*black list*). The property must use the following format: `os:`*`stringValue`* or: `release:`*`stringValue`* Additional name/value pairs must be comma-separated.
policy.runtimeMinSecurityLevel	The application runtime module must have the specified minimum security level, or higher, to access protected content.

Property	Description
policy.runtimeVersionBlacklist.*n*	A list of versions of runtime modules that may not be used (*black list*). The property must use the following format: `os` : stringValue or: `application` : stringValue or: `release` : stringValue Additional name/value pairs must be comma-separated.
policy.allowedAIRApplication.*n*	A white list of Adobe AIR applications allowed to play protected content. The property must use the following format: *`pubId`*[:*`appId`*[:[*`min`*]:[*`max`*]]]
policy.allowedSWFApplication.*n*	A white list of SWF applications allowed to play protected content. Use the following format: *`URL`* or file=*`swf_file`*,time=*max_time_to_verify swf_file* is the SWF file for which to compute the hash and *max_time_to_verify* is the maximum time to allow for download and verification of the SWF to complete (in seconds).
policy.license.customProp.*n*	Custom name/value pairs to be included in licenses issued to users. Use the following format: policy.license.customProp.*`n`*=*`name`*=*`value`* This option can be defined multiple times for multiple custom properties.

Command line usage

Before using Policy Manager, ensure that you fulfill the requirements listed in Requirements and that the configuration file contains the required information (see Configuration file).

Policy Manager is in the \Reference Implementation\Command Line Tools directory on the DVD. To run the tool, use the following syntax:

```
java -jar AdobePolicyManager.jar command filename [options]
```

The following table contains descriptions of the command line actions shown in the syntax above:

Command line action	Description
new	Creates a new policy.
detail	Describes an existing policy.
update	Updates an existing policy.

The following table describes the command line options that can be specified along with the syntax above:

Command line option	Description
-c *configfile*	Specify the location of the configuration file. If this option is not used, the Policy Manager will look for flashaccesstools.properties in the working directory. Options specified on the command line take precedence over those present in the configuration file.
-n *policyname*	The name of the policy.
-o	If the destination file already exists, overwrite it without prompting.
-noprompt	Do not ask if the destination file should be overwritten. If the destination file already exists and -o is not set, an error will be returned.
-root	Indicates the policy has a root license. Not allowed for updates.
-e *date*	The date before which licenses will be valid. Specify as *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec*. For example, 2008-12-1 or 2008-12-1-00:00:00 for midnight on December 1, 2008. The value must be greater than the value of -s, if present. This option cannot be used with -r. To remove the end date when updating a policy, use -e without specifying a date.
-r *minutes*	The duration (minutes) that content protected with this policy is valid, beginning when the content is protected with the packager. The value must be non-negative. This option cannot be used with -e. To remove the duration when updating a policy, use -r without specifying a number of minutes.
-s *date*	The date after which licenses will be valid. Specify as *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec*. For example, 2008-12-1 or 2008-12-1-00:00:00 for midnight on December 1, 2008. The value must be less than the value of -e, if present. This option cannot be used with -r. To remove the start date when updating a policy, use -s without specifying a date.
-w *minutes*	The playback window (the number of minutes the content may be viewed, beginning from the first playback). If this option is not specified or if -w is used without specifying the number of minutes, there is no playback window limitation. The value must be non-negative.
-l *minutes*	The license caching duration in minutes, which is the time a license will be allowed to be cached in the client's License Store after the license has been issued by the server. The value must be non-negative. Specify '-l 0' to indicate license caching is not permitted. Use -l without specifying a number of minutes for unlimited license caching.

Command line option	Description
-ldate *date*	The license caching end date (the date after which licenses may not be cached in the client's License Store, after the license has been issued by the server). Specify as *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec*. For example, 2008-12-1 or 2008-12-1-00:00:00 for midnight on December 1, 2008. Use -l without specifying a number of minutes for unlimited license caching.
-authNS	The authentication namespace. If specified, the client should authenticate with a user name and password issued by the specified authority. This option cannot be used with -x. It is not allowed for updates.
-x	Allow anonymous access. This option cannot be used with -authNS. It is not allowed for updates.
-air *pubId*[:*appId*[:[*min*]:[*max*]]]	A whitelist of AIR applications allowed to play protected content. Use this to restrict which publishers, applications, and versions may access content protected with this policy. If *appId* is not specified, all applications for publisher *pubId* are allowed. *min* and *max* version numbers are optional. Multiple -air options may be specified to allow multiple applications. If no AIR or SWF applications are specified, all applications may access this content. During an update, use -air without the remaining arguments to remove all entries from the list.
-drmBlacklist *name/value pairs*	The DRM clients restricted from accessing protected content. The value consists of comma separated name:value pairs with the following format: `os | release=stringValue` For example, 'os=Win,release=2.0.1'. During an update, use -drmBlacklist without the remaining arguments to remove all entries from the list.
-drmLevel *int*	Indicates that DRM clients must have the specified minimum security level to access protected content.
-opAnalog NO_PROTECTION | USE_IF_AVAILABLE | REQUIRED | NO_PLAYBACK	Analog output protection constraints.
-opDigital NO_PROTECTION | USE_IF_AVAILABLE | REQUIRED | NO_PLAYBACK	Digital output protection constraints.

Command line option	Description
-runtimeBlacklist *name/value pairs*	The application runtimes restricted from accessing protected content. The value consists of comma separated name:value pairs with the following format: `os | application | release=`*`stringValue`* For example, 'os=Win,release=2.0.1,application=AIR'. During an update, use -runtimeBlacklist without the remaining arguments to remove all entries from the list.
-runtimeLevel *int*	Indicates that the application runtimes must have the specified minimum security level to access protected content.
-swf *url* -swf file=*swf_file*, time=*max_time_to_verify*	A whitelist of SWF applications allowed to play protected content. Multiple -swf options may be specified to allow multiple applications. If no AIR or SWF applications are specified, all applications may access this content. During an update, use -swf without the remaining arguments to remove all entries from the list. To identify a SWF by its hash value, specify the SWF file for which to compute the hash and the maximum time to allow for SWF verification to complete (in seconds).
-k *name=value*	Specifies custom key/values to add to the policy. Multiple -k options may be specified. During update, use -k without the remaining arguments to remove all properties. The interpretation or handling of this data is completely up to the implementation of the Flash Access license server.
-p *name=value*	Adds a custom property, which will appear in the license generated for each client. Multiple -p options may be specified to add multiple properties. During an update, use -p without the remaining arguments to remove all properties. The interpretation or handling of this data is completely up to the implementation of the client application.

Media Packager

Using Media Packager, you can specify what data in the file to encrypt and the policy to apply to the content file. For example, you can specify that the video data is encrypted but the audio data is unencrypted.

Configuration file properties

Before you run Media Packager, specify values for the Media Packager properties. The configuration file specifies the following properties.

Property	Description
encrypt.contents.video	Indicates whether to encrypt video content.
encrypt.contents.audio	Indicates whether to encrypt audio.

Property	Description
encrypt.contents.script	Indicates whether to encrypt script data in FLVs. `onMetaData` and `onXMP` script data tags are never encrypted, even if this option is enabled.
encrypt.contents.video.level	Indicates the video encryption level. A value of high is used to encrypt all video content, while values of medium and low are used to encrypt portions of the video content for F4V files containing H.264 content. value = high | medium | low
encrypt.contents.secondsUnencrypted	If the value is greater than 0, the specified number of seconds of content at the beginning of the file will not be encrypted.
encrypt.keys.asymmetric.certfile	The license server certificate file used to encrypt the key. The encrypt.keys.asymmetric.certfile property specifies a file that contains the certificate only (either PEM or DER format is acceptable).
encrypt.keys.policyFile.*n*	This property is used repeatedly to create a list of policies to apply to the content. *n* is an integer whose value is 1 or greater. The client will use the first instance by default.
encrypt.license.serverurl	The license server URL.
encrypt.license.servercert	The transport certificate for the license server. This property specifies a .cer file that contains the certificate only (either PEM or DER format is acceptable).
encrypt.sign.certfile	The PKCS12 file containing packager credentials for signing content. The encrypt.sign.certfile should refer to a .pfx file containing a certificate and private key.
encrypt.sign.certpass	The password used to protect the file specified by `encrypt.sign.certfile`.

Command line usage

Before using Media Packager, ensure that you fulfill the requirements listed in Requirements and that the configuration file contains the required information (see Configuration file).

Media Packager is in the \Reference Implementation\Command Line tools directory on the DVD. To encrypt a single file, use the following syntax:

```
java -jar AdobePackager.jar source dest [options]
```

- *source* is the file to be encrypted.
- *dest* specifies where the encrypted content will be written. If a directory is specified, the encrypted file will be saved in this folder using the same file name as the source file, but the directory must not be the directory which contains the source file.

To encrypt multiple files with the same key (for multi-bit-rate support), use the following syntax:

```
java -jar AdobePackager.jar sourcefiles dest-directory [options]
```

- *sourcefiles* is a series of whitespace-delimited `source` entries representing the files to be encrypted.
- *dest-directory* specifies where the encrypted content will be written. The encrypted files will be saved in this folder using the same file names as the source files, but the directory must not be the directory that contains the source files.

To view information about an encrypted file, use the following syntax:

```
java -jar AdobePackager.jar -d encryptedfile [-e] [-m]
```

- *encryptedfile* is the encrypted file.

The following table contains descriptions of the command line options shown in the syntax above:

Command line option	Description
-c *configfile*	Specifies the location of the configuration file. If this option is not used the Media Packager will look for flashaccesstools.properties in the working directory.
-d *encryptedfile*	Shows information about a file that was already packaged. The source and destination files are not required.
-e	Use this option with -d to extract policies from a packaged file. A file will be created in the same directory as the encrypted file using the file name and policy identifier.
-h	Use with -d to extract the DRM header from a packaged file. A file is created in the same directory as the encrypted file, using the file name and the extension '.header'
-i *contentID*	Specifies a unique identifier for this piece of content. If no identifier is specified, the destfile file name will be used.
-k *key*= *value*	Specifies a custom key/value to add to content metadata. Multiple -k options may be specified.
-m	Use this option with -d to extract metadata from a packaged file. A file will be created in the same directory as the encrypted file using the file name and the extension ".metadata".
-noprompt	Do not ask whether the destination file should be overwritten. If the destination file already exists and -o is not set, an error will be returned.
-o	Overwrites the destination file without prompting, if it already exists.
-p *filename*	Specifies the name of the file containing the policy. Multiple -p options may be specified, and the client will use the first by default. The values specified on the command line take precedence over those specified in the configuration file.

Policy Update List Manager

Before using Policy Update List Manager, ensure that you fulfill the requirements listed in Requirements and that the configuration file contains the required information (see Configuration file).

Configuration file properties

The following are the Policy Update List Manager properties, which specify a PKCS12 file containing credentials for signing revocation lists (License Server Certificate):

```
revocation.sign.certfile=license-server-credentials.pfx
revocation.sign.certpass=password
```

Command line usage

Policy Update List Manager is in the \Reference Implementation\Command Line Tools directory on the DVD. To run the tool, use one of the following syntaxes:

```
java -jar AdobePolicyUpdateListManager.jar destfile [options]
java -jar AdobePolicyUpdateListManager.jar -d filename
```

- *destfile* indicates where the policy update list will be written.

The following table contains descriptions of the command line options shown in the syntax above:

Command line option	Description
-c *configfile*	Specifies the location of the configuration file. If this option is not used, the Policy Update List Manager will look for flashaccesstools.properties in the working directory.
-d *filename*	Displays information about the policy update list.
-e *date*	(Optional) The expiration date of the policy update list. Use the format *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec* (for example, 2009-01-31-14:30:00 represents January 31 at 2:30 PM).
-f *filename*	Adds all entries from the existing policy update list. Only one existing file may be specified.
-noprompt	Do not ask if the destination file should be overwritten. If the destination file already exists and -o is not set, an error will be returned.
-o	If the destination file already exists, overwrite it without prompting.
-r *policyID date* "*reasonCode*" "*reasonText*" "*reasonURL*"	(Optional) Revokes the policy ID on the specified date. An optional reason code, reason text, and reason URL may also be provided. Specify an empty string "" to indicate that no value is provided for the optional parameters. Specify the date as *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec* (for example 2008-12-1 or 2008-12-1-00:00:00 for midnight on December 1, 2008). If a date is not specified, the current date is used. The reason code must be greater than or equal to 0. Multiple -r options may be specified.

Command line option	Description
-rf *policyFilename date* "*reasonCode*" "*reasonText*" "*reasonURL*"	Performs the same action as the -r flag, but extracts the policy identifier from the given file.
-u *policyFilename* "*reasonCode*" "*reasonText*" "*reasonURL*"	Replaces any matching policy in a license request with this policy using the given reason code (optional), reason text (optional), and reason URL (optional). Specify an empty string "" to indicate that no value is provided for the optional parameters. The reason code must be greater than or equal to 0. Multiple -u options may be specified.

Revocation List Manager

Before using Revocation List Manager, ensure that you fulfill the requirements listed in Requirements and that the configuration file contains the required information (see Configuration file).

Configuration file properties

The following are the Revocation List Manager properties, which specify a PKCS12 file containing credentials for signing revocation lists (License Server Certificate):

```
revocation.sign.certfile=license-server-credentials.pfx
revocation.sign.certpass=password
```

Command line usage

Revocation List Manager is in the \Reference Implementation\Command Line Tools directory on the DVD. To run the tool, use one of the following syntaxes:

```
java -jar AdobeRevocationListManager.jar destfile crlNumber [options]
java -jar AdobeRevocationListManager.jar -d filename
```

- *destfile* indicates where the revocation list will be written.
- *crlNumber* is a non-negative version number of the CRL. This number should be incremented each time the CRL is updated.

The following table contains descriptions of the command line options shown in the syntax above:

Command line option	Description
-c *configfile*	Specifies the location of the configuration file. If this option is not used, the Revocation List Manager will look for flashaccesstools.properties in the working directory.
-d *filename*	Displays information about the revocation list.
-e *date*	(Optional) The expiration date of the revocation list. Use the format *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec* (for example, 2009-01-31-14:30:00 represents January 31 at 2:30 PM).
-f *filename*	Adds all entries from the existing revocation list. Only one existing file may be specified.

Command line option	Description
-noprompt	Do not ask if the destination file should be overwritten. If the destination file already exists and -o is not set, an error will be returned.
-o	If the destination file already exists, overwrite it without prompting.
-r *issuerName serialNumber revocationDate*	Revokes the certificate identified by issuerName and serialNumber on the given date. The issuerName must follow the 509 name format (for example, "CN=12345,O=Adobe Systems Incorporated,C=US"). Specify serial numbers in hexadecimal form. Specify the revocation date as *yyyy-mm-dd* or *yyyy-mm-dd-h24:min:sec*, for example 2008-12-1 or 2008-12-1-00:00:00 for midnight on December 1, 2008. If the revocation date is not specified, the current date is used.

License server and watched folder packager

The reference implementation server can help you create a license server using the Flash Access SDK. In this implementation, users are authenticated based on user entries in a database. The server includes demonstration business logic for issuing licenses. It also implements compatibility support for Flash Media Rights Management Server 1.0 and 1.5.

The reference implementation server also includes a watched folder implementation of the packager. This component may be deployed along with the license server or on a separate machine. With this packager implementation, multiple watched folders can be created. When content is dropped into the watched folder, the packager automatically packages the content.

The license server and packager are deployed as separate WAR files, so you can choose whether to run them on separate servers or in a single Apache Tomcat® instance. The license server is in the flashaccess.war and the packager is in flashaccess-packager.war. The optional edcws.war contains support for license requests from FMRMS 1.x clients.

The Reference Implementation sample code demonstrates the following features:

- License Server:
  - Handling authentication requests, using a database to validate username/password
  - Handling license requests
  - Issuing licenses for content containing multiple policies
  - Using database to determine if user is authorized to view content
  - Using policy update lists
  - Using machine revocation lists
  - Using an HSM or PKCS12 file to store credentials
  - Encrypting passwords specified in properties file
  - Specifying multiple license server or transport credentials (after credentials are renewed, the old credentials are kept on the server so existing content can be consumed without needing to repackage)

- Restricting DRM/Runtime versions allowed to make requests to the license server
- Setting client clock windback preferences
- Restricting time difference allowed between request time and server time (to prevent replay attacks)
- Handling requests from 1.x clients (triggers 1.x client to upgrade to 2.0)
- Converting 1.x metadata to 2.0 metadata on the fly, using 1.x license information stored in a database
- Sample code for converting 1.x policies to 2.0 policies
- Sample scripts for importing 1.x license information from an existing database

- Packager Server:
  - Implementing a packager implementation that automatically packages content added to a watched folder
  - Using an HSM or PKCS12 file to store credentials
  - Encrypting passwords specified in properties file
  - Configuring the packager, creating policies, and creating policy update lists using an AIR application

Requirements

You will need to ensure that you have the following installed:

- Tomcat 6.0
- A database such as MySQL (only required for License Server component)
- Java 1.6
- Ant (to use the sample build scripts)

Once you have installed Tomcat and MySQL, obtain the credentials from Adobe.

Building the license server

The reference implementation license server includes WAR files for deploying the license server. It also includes all the license server source code and an Ant build script (Reference Implementation\Server\refimpl\build-refimpl.xml) so you can easily make changes to the code.

Note: This step is only needed if you want to modify the source code. For evaluation purposes, you can skip this step and use the WAR files as shipped.

Before running the Ant script, modify the script to specify the locations of the Flash Access SDK, Tomcat, MySQL, and Log4J. Open build-refimpl.xml in a text editor and edit the values of the properties sdkdir, tomcatdir, mysqldir, and log4jdir. To compile the source code and create the WAR files for the reference implementation, run the script using "ant -f build-refimpl.xml all" in the directory containing the Ant script. When the script is complete, a refimpl-build/wars directory containing the server WAR files will be created.

Configuration

You will need to configure the server properties files, watched folder properties, set up the database, and configure the HSM.

Server properties files

The server requires two configuration files, one for the license server and one for the packager. Both files must be placed on the classpath. The properties files contain the location of the credentials issued by Adobe. These credentials can be specified as a .pfx file and password or by providing an alias and password for a credential stored on an HSM.

Please refer to the property files for details about the specific values and usage of the each parameter. Sample properties files can be found in the "resources" directory of the reference implementation (Reference Implementation\Server\resources).

To ensure the security of your credential's password, a tool is provided (ScrambleUtil.class) to encrypt the password before it is entered into the flashaccess-refimpl.properties or flashaccess-refimpl-packager.properties file.

To properly prepare your credential's password:

1. Go to Reference Implementation\Server\refimpl\scrambler.
2. From the command prompt, enter the command:

```
java -classpath path_to_adobe-flashaccess-sdk.jar;.
   com.adobe.flashaccess.refimpl.util.ScrambleUtil "your_pfx_password"
```

Note: The previous example uses a semicolon (;) as the delimiter. For platforms other than Microsoft Windows, use a colon (:) as the delimiter.

The utility outputs the encrypted password, which you must copy to the .properties file.

License server properties file

The flashaccess-refimpl.properties file is used to configure the License Server component of the reference implementation. At a minimum, be sure to configure the properties related to the Transport Credential and the License Server Credential. The locations of the credential files must be specified relative to the directory specified by the "config.resourcesDirectory" property. This file contains several properties related to packaging content: these properties are only used for Flash Media Rights Management Server 1.x metadata conversion. If you modify any of the values in this property file, you need to restart the license server for the changes to take effect.

Packager properties file

The flashaccess-refimpl-packager.properties is used to configure the Watched Folder Packager component of the reference implementation. At a minimum, be sure to set the license server URL, license server certificate, packager credential, and key protection options. This file also contains the location of each watched folder (packager.watchfolder.source.*n*). Any changes made to the values in this property file will take effect the next time the watched folder packager runs (restarting the server is not required). However, if there is a configuration error in the packager, the watched folder packager thread will exit, and the server will need to be restarted to restart the packager thread.

Watched folder properties

Each watched folder contains a watchedfolder.properties file. This file contains the packaging options for content placed in this folder, including what to encrypt and which policies to apply. Any changes made to the values in the property file take effect the next time the watched folder packager runs (you do not need to restart the server).

If there is a configuration error in the packager properties file, the packager thread stops. To resume the watched folder packager, restart the server. If there is a configuration error in a watched folder properties file, the watched folder is temporarily removed from the list of folders the packager processes. To add the watched folder back to the list, restart the server or modify the packager properties file. If an error occurs during packaging of a particular file (for example, because the file is corrupt), the file is skipped and the remaining files in the folder are processed.

Setting up the database and configuring the JNDI datasource

The reference implementation license server requires a database to support the following features:

- User authentication
- Usage model demo business rules
- Metadata conversion

Anonymous license acquisition does not require a database to be running.

Note: The instructions in this section are for the Microsoft Windows platform. For other operating systems, see the documentation for your operating system or see the MySQL documentation.

To run the license server, you will need to install and configure MySQL 5.1.34:

1. Run the MySQL installer (found in the Third Party\MySQL\Installer\5.1 folder on the DVD).
2. At the end of the installation procedure, check "Configure MySQL Server Now" to start the configuration wizard. Use the default settings or select specific settings for your testing purposes, with the exception that on the 5th screen you must select "Online Transaction Processing (OLTP)" or "Manual Setting" and enter the maximum number of connections allowed.
3. Make a note of the root password.
4. If you need to re-install MySQL, follow these steps to avoid problems in starting the server afterward:
   - Delete the folder system drive \Documents and Settings\All Users\Application Data\MySQL.
   - Delete the old MySQL install folder: for example, system drive:\Program Files\MySQL\MySQL\Server 5.1.

Next, you will need to install MySQL JDBC Driver 5.1.7. To do this, copy mysql-connector-java-5.1.7-bin.jar (found in the Third Party\MySQL\Installer\5.1 folder on the DVD) to Tomcat Server lib directory: ...\Tomcat6.0\lib.

Note: MySQL JDBC Driver 5.1.7 works with Tomcat 6.0. Older versions of Tomcat are not supported.

Set up the sample database by setting up the database schema and populating the database with sample data. To do this, perform the following steps:

1. Go to Window's Start Menu, MySQL -> MySQL Server 5.1 -> MySQL Command Line Client.
2. After typing in the password, execute the following SQL script to add the user account `dbuser` for establishing a connection through a web application and create database schema (make sure that there is no ";" at the end. Just press enter.):

   ```
   mysql> source Reference Implementation\Server\dbscript\createsampledb.sql
   ```

3. Edit the script that populates sample data in the tables to include data for your testing purposes: Reference Implementation\Server\dbscript\PopulateSampleDB.sql.

4. Execute this script to populate the data as you did in step 2.

Note: The first time you run the CreateSampleDB.sql.script you will receive the following error:

ERROR 1396 (HY000): Operation DROP USER failed for 'dbuser'@'localhost' Query OK, 0 rows affected (0.00 sec).

You can safely ignore this error. This only happens the first time you run this script.

At this point you will need to configure Database Connection Pooling (DBCP). DBCP uses the Jakarta-Commons Database Connection Pool. A JNDI Datasource TestDB is configured to take advantage of this application server connection pooling. To change database connection to point to a MySQL server that is not on localhost, modify the META-INF\context.xml file (which specifies the location, username, and password of the license server's database) located in flashaccess.war, or modify \Reference Implementation\Server\refimpl\WebContent\META-INF\context.xml and recreate the WAR file using the updated files. To change any of these parameters, edit the context.xml located in the WebContent directory and use the Ant script to recreate the WAR file. To tune the database, change the JNDI datasource settings in this file.

If you debug the Reference Implementation project within Eclipse, you need to add $CATALINA_HOME\lib\tomcat-dbcp.jar to your run/debug configuration. This step is not required if you run the flashaccess.war file on a standalone Tomcat 6.0 server.

HSM configuration

Use of an HSM is not required, but it is recommended. The reference implementation can be configured to use the Sun PKCS11 provider for HSM support. In order to use a credential on an HSM, you must create a configuration file for the Sun PKCS11 provider. See the Sun documentation for details. To verify that your HSM and Sun PKCS11 configuration file are configured properly, you can use the following command (keytool is installed with the Java JDK):

```
keytool -keystore NONE -storetype PKCS11
   -providerClass sun.security.pkcs11.SunPKCS11 -providerArg pkcs11.cfg -list
```

If you see your credentials in the list, the HSM is configured properly.

Note: If you use a 64-bit version of Windows, HSM is currently not supported by the Reference Implementation.

Crossdomain policy file

If the license server is hosted on a different domain than the video playback SWF, then a cross-domain policy file (crossdomain.xml) might be necessary to allow the SWF to request licenses from the license server. A cross-domain policy file is an XML file that provides a way for the server to indicate that its data and documents are available to SWF files served from other domains. Any SWF file that is served from a domain that the server's cross-domain policy file specifies is permitted to access data or assets from that server.

Adobe recommends that developers follow best practices when deploying the cross-domain policy file by only allowing trusted domains to access the license server and limiting the access to the license sub-directory on the web server. For more information on cross-domain policy files, please see the following locations:

- Website controls (policy files)
- Cross-domain policy file specification

Deploying the license server and watched folder packager

Copy the license server WAR files to Tomcat's webapps directory. If you have previously deployed the WAR file, you may need to manually delete the unpacked WAR directories ("flashaccess", "edcws", and "flashaccess-packager" in Tomcat's webapps directory). To prevent Tomcat from unpacking WAR files, edit the server.xml file in Tomcat's conf directory and set the "unpackWARs" attribute to "false".

The properties file (flashaccess-refimpl.properties) must be on the classpath for the server to load the properties. Copy this file to a directory and update the file with the appropriate values. Edit the catalina.properties file in Tomcat's conf directory and add the directory containing flashaccess-refimpl.properties to the "shared.loader" property. The log4j.xml file for configuring logging must also be on the classpath (see resources\log4j.xml for an example).

The reference implementation server uses several certificate files, policy files, and other resources. Those files are all located in one resource folder. By default, the resource folder is C:\flashaccess-server-resources, but this location can be modified in flashaccess-refimpl.properties. Be sure to copy all the required resources to this location before starting the server.

To start Tomcat and the license server, run "catalina.bat start" from Tomcat's bin directory.

Troubleshooting

Listed below are common problems and solutions for deployment:

- If you see the following error:

  ```
  "Error decoding the password for
  HandlerConfiguration.ServerTransportCredential.password

  javax.crypto.IllegalBlockSizeException: Input length must be multiple of 8
  when decrypting with padded cipher"
  ```

  Make sure the password is encrypted using the provided ScrambleUtil class.

- If you see the following error:

  ```
  "Unable to load credential from file.pfx -- possibly wrong password."
  ```

  Make sure you specified the correct encrypted password for the PFX file.

- If you see the following error:

  ```
  "javax.crypto.BadPaddingException: Given final block not properly padded"
  ```

  Make sure you used the password scrambler class provided with the Reference Implementation (this scrambler utility is different from the one provided with the Adobe® Flash® Access™ Server for Protected Streaming).

Determining if Reference Implementation License Server is properly running

There are several ways to determine whether your server has started correctly. Viewing the catalina.log logs may not be sufficient, as the license server logs to its own log files. Follow the steps below to ensure your Reference Implementation has started up properly.

- Check your "AdobeFlashAccess.log" file. This is where the Reference Implementation writes log information. The location of this log file is indicated by your log4j.xml file and can be modified to point

to any location you'd like. By default, the log file will be output to the working directory where you've run catalina.

- Navigate to the following URL: http://*your server*:*server port*/flashaccess/license/v1. You should see the text "License Server is setup correctly".

Another way to test if your server is running correctly is to package a piece of test content, set up a sample video player, and play it. The following procedure describes this process:

1. Navigate to \Reference Implementation\Command Line folder. For information on installing the command line tools, see "Installing the command line tools" on page 54.
2. Create a simple anonymous policy by using the following command:

   ```
   java -jar libs\AdobePolicyManager.jar new policy_test.pol
   ```

   For mor information on creating policies using the Policy Manager, see "Command line usage" on page 56.
3. Set the `encrypt.license.serverurl` property in the flashaccesstools.properties file to the URL of the license server (for example, http://localhost:8080/). The flashaccesstools.properties file is located under the \Reference Implementation\Command Line Tools folder.
4. Package a piece of content by using the following command:

   ```
   java -jar libs\AdobePackager.jar test_input_FLV output_file
   ```

5. Copy the 3 generated files to the Tomcat webapps\ROOT\content folder.
6. Extract SampleVideoPlayers.zip and copy FlashPlayer\Release to the Tomcat webapps\ROOT\SVP\ folder.
7. Install Flash Player 10.1 or later.
8. Open the web browser and navigate to the following URL:

   http://localhost:8080/SVP/SampleVideoPlayer_FP.html
9. Navigate to the following URL, then click "Show DRM Events", and then click "Play":

   http://localhost:8080/content/*your_encrypted_FLV*.
10. If the video fails to play, check if any error codes were written in the logging pane of the Sample Video Player, or in the AdobeFlashAccess.log file. The location of the AdobeFlashAccess.log log file is indicated by your log4j.xml file, and can be modified to point to any location you'd like. By default, the log file is written to the working directory where you've run catalina.

Implementing the usage models

The Reference Implementation includes business logic for demonstrating how to enable the following four different usage models for a piece of packaged content:

- Download-to-own (DTO)
- Rental/Video-on-demand (VOD)
- Subscription (all-you-can-eat)
- Ad-funded

To enable the usage model demo, specify the custom property "RI_UsageModelDemo=true" at packaging time. If you are packaging content using the Media Packager command line tool, specify:

```
java -jar AdobeMediaPackager.jar source.flv dest.flv -k
RI_UsageModelDemo=true
```

Note: If you do not activate the optional demo mode at packaging time, the license server uses the policy specified at packaging time to issue a license. If multiple policies were specified, the license server uses the first valid policy.

In the demo, the business logic on the server controls the actual attributes of the licenses generated. At packaging time, only minimal policy information must be included in the content. Specifically, the policy only needs to indicate whether authentication is required to access the content. To enable all four usage models, include one policy that allows anonymous access (for the Ad-funded model) and one policy that requires user name/password authentication (for the other 3 usage models). When requesting a license, a client application can determine whether to prompt the user for authentication based on the authentication information in the policies.

To control the usage model under which a particular user is to be issued a license, entries may be added to the Reference Implementation database. The Customer table contains user names and passwords for authenticating users. It also indicates whether the user has a subscription. Users with subscriptions will be issued licenses under the Subscription usage model. To grant a user access under the Download to Own or Video on Demand usage models, an entry may be added to the CustomerAuthorization table, which specifies each piece of content the user is allowed to access and the usage model. See the PopulateSampleDB.sql script for details on populating each table.

When a user requests a license, the Reference Implementation server checks the metadata sent by the client to determine if the content was packaged using the `RI_UsageModelDemo` property. If so, the following business rules are used:

- If one of the policies requires authentication:
  - If the request contains a valid authentication token, look for the user in the Customer database table. If the user was found:
    - If the `Customer.IsSubscriber` property is `true`, generate a license for the Subscription usage model and send it to the user.
    - Look for a record in the CustomerAuthorization database table for this user and content ID. If a record was found:
      - If `CustomerAuthorization.UsageType` is `DTO`, generate a license for the Download To Own usage model and send it to the user.
      - If `CustomerAuthorization.UsageType` is `VOD`, generate a license for the Video On Demand usage model and send it to the user.
  - If none of the policies allow anonymous access:
    - If there is not a valid authentication token in the request, return an “authentication required” error.
    - Otherwise return a “not authorized” error.
- If one of the policies allows anonymous access, generate a license for the Ad-funded usage model and send it to the user.

Before the Reference Implementation server can issue licenses for the usage model demo, the server needs to be configured to specify how licenses are generated for each of the four usage models. This is done by specifying a policy for each usage model. The Reference Implementation includes four sample

policies (dto-policy.pol, vod-policy.pol, sub-policy.pol, ad-policy.pol) or you may substitute your own policies. In flashaccess-refimpl.properties, set the following properties to specify the policy to use for each usage model and place the policy files in the directory specified by the `config.resourcesDirectory` property:

```
# Policy file name for Download To Own usage
RefImpl.UsageModelDemo.Policy.DTO=dto-policy.pol
# Policy file name for Rental usage
RefImpl.UsageModelDemo.Policy.VOD=vod-policy.pol
# Policy file name for Subscription usage
RefImpl.UsageModelDemo.Policy.Subscribe=sub-policy.pol
# Policy file name for Ad Supported (free) usage
RefImpl.UsageModelDemo.Policy.Free=ad-policy.pol
```

Download-to-own

With the DTO usage model, a user may download the content for use online or offline and is issued a permanent license for the content. When requesting a license, the user must authenticate so the server can verify that the user has purchased the content.

Rental/Video-on-demand

With the VOD usage model content is offered with time-based restrictions. For example, a user has the option to play the content during a 30-day period; however, once playback begins, the user has up to 48 hours to finish watching, after which time the content will no longer be playable. When requesting a license, the user must authenticate so the server can verify that the user has a rental account.

Subscription

Some services offer paid subscriptions that give users unlimited access to a large library of content for as long as they continue to pay the monthly fees. The license server issues a unique license for each piece of content and also issues a root license whose expiration coincides with the subscription period. Each month, when the user renews his subscription, the root license can also be renewed. When requesting a license, the user needs to authenticate so the server can verify that the user's subscription is up to date.

Ad-funded

Content is monetized by including advertising as part of the experience. With this model, content can be distributed without requiring user authentication.

Migrating from FMRMS 1.0 or 1.5 to Flash Access 2.0

In order to continue to issue licenses for content packaged using Flash Media Rights Management Server (FMRMS) 1.0 or 1.5, license and policy data must be migrated from the LiveCycle ES server to the customer's new server based on the Flash Access 2.0 SDK. The important steps are:

1. Importing license information
2. Converting FMRMS policies to Flash Access 2.0 format
3. Supporting the 1.x compatibility requests via the `FMRMSv1RequestHandler` and `FMRMSv1MetadataHandler`

To import license information from LiveCycle ES into your Flash Access 2.0-based server, refer to the sample database scripts provided in the Reference Implementation\Server\migration\db folder. Sample scripts are provided for exporting the relevant data from a MySQL, Oracle, or SQL Server database into a CSV file format. Once the data is exported, it can be imported into the database of your choice. The exported license information includes the License ID, a Content ID assigned at packaging time, the ID of the Policy used, the time the content was packaged, and the content encryption key. For 2.0, this information is required in order to convert the 1.x content metadata into the 2.0 metadata format (see `FMRMSv1RequestHandler` and `FMRMSv1MetadataHandler`). In the reference implementation, this data is stored in the License database table and used by `RefImplMetadataConvReqHandler`.

Existing policies will need to be converted to the Flash Access 2.0 format in order to use those policies when converting metadata and issuing licenses for 1.0 or 1.5 content. The Reference Implementation\Server\migration folder contains sample code for creating a 2.0 policy based on older policies.

If you are migrating from FMRMS 1.0 to Flash Access 2.0, see the V1_0PolicyConverter.java sample. Compile the sample code by running "ant-f build-migration.xml build-1.0-converter" (the script expects the 1.0 and 2.0 libraries to be in libs/1.0 and libs/2.0 respectively). Edit the converter.properties file to point to your LiveCycle ES server. Then run "ant -f build-migration.xml migrate-all-1.0-policies" to convert all FMRMS 1.0 policies to 2.0 format.

If you are migrating from FMRMS 1.5 to Flash Access 2.0, see the V1_5PolicyConverter.java sample. Compile the sample code by running "ant-f build-migration.xml build-1.5-converter" (the script expects the 1.5 and 2.0 libraries to be in libs/1.5 and libs/2.0 respectively). Edit the converter.properties file to point to your LiveCycle ES server. Then run "ant -f build-migration.xml migrate-all-1.5-policies" to convert all FMRMS 1.5 policies to 2.0 format.

The converted policies will be written to a set of files. In addition, PolicyConverter will output a CSV file containing the mapping of old policy IDs to new policy IDs. This file can be imported into the "PolicyConversion" table in the reference implementation database, and will be used by `RefImplMetadataConvReqHandler`.

Once the relevant data has been migrated to your Flash Access 2.0-based server, you are ready to implement support for 1.x compatibility requests. See `RefImplUpgradeV1ClientHandler` and `RefImplMetadataConvReqHandler` in the reference implementation for examples of how to process these types of requests.

Adobe Flash Access Manager AIR application usage

The Flash Access Manager is an Adobe AIR application for packaging Flash Access content. Using this application, you can create policies, manage a policy update list, and package content. You can also set up Watched Folders to automatically package content with certain settings when new content is added to the folder.

Building the Packager Server and AIR Application

There are two components required to use the Adobe Flash Access Manager: the Adobe Flash Access Manager AIR application and the Packager Server (flashaccess-packager.war). Both components are distributed in both source and binary forms with the Reference Implementation.

Building the Packager Server

If you wish to modify the source code, see the instructions on compiling the Reference Implementation in "Building the license server" on page 65.

Building the Flash Access Manager AIR Application

To build the Flash Access Manager AIR file from the source code, you need the Flex and AIR SDK installed on your machine. Before you can package and run the application, you must compile the MXML code into a SWF file using the amxmlc compiler. The amxmlc compiler can be found in the bin directory of the Flex 3 or later SDK. If desired, you can set your path environment variable to include the Flex SDK bin directory to make it easier to run the utilities on the command line.

Use the following procedure to build the Flash Access Manager AIR file:

1. Open a command shell or a terminal and navigate to the project folder of the Flash Access Manager AIR application (UI Tools\Flash Access Manager in the Reference Implementation directory).

2. Enter the following command:

   ```
   amxmlc src\FlashAccessmanager.mxml
   ```

   Running amxmlc produces FlashAccessManager.swf, which contains the compiled code of the application. For more information see, http://livedocs.adobe.com/flex/3/html/help.html?content=CommandLineTools_5.html.

The Adobe AIR SDK includes the AIR Developer Tool (ADT) utility to package AIR applications and generate certificates. AIR applications should be digitally signed; users will receive a warning when installing applications that are not properly signed or are not signed at all. To generate a certificate using the command line, open a console window in the same folder as your AIR application and type the following:

```
adt -certificate -cn SelfSigned 1024-RSA testCert.pfx some_password
```

Substitute *some_password* with a password of your choice. After a few seconds, ADT should complete its certificate generation process and you should have a new testCert.pfx file in your application directory.

Next, use ADT to package the application into an .air file, by using the command:

```
adt -package -storetype pkcs12 -keystore testCert.pfx FlashAccessManager.air
src\FlashAccessManager-app.xml . -C src assets
```

This command tells ADT to package your application, using the key file in testCert.pfx. In the line above, you configure ADT to package your entire application into a file named FlashAccessManager.air, and to include the files FlashAccessManager-app.xml and FlashAccessManager.swf and the images from the assets directory.

As part of this process, you'll be prompted for the password that you set for your new certificate file. Enter it, wait a moment, and a FlashAccessManager.air file should appear in the same directory as your project files.

Initial Flash Access Manager setup

Use the following procedure to set up Flash Access Manager:

1. Deploy the Packager Server. This server should only be available to users within your firewall (do not deploy this software on a public-facing machine). For more information on deploying the server, see "Deploying the license server and watched folder packager" on page 69.
   - Copy flashaccess-packager.war to Tomcat's webapps folder.
   - Copy flashaccess-refimpl-packager.properties from resources to a location on the classpath.
   - Start the server. You will see some errors due to problems in the properties file; this is expected since the properties have not been filled in yet.
2. Install the Adobe Flash Access Manager AIR application by launching the .air file (requires AIR 1.5 or higher).
3. Launch the Adobe Flash Access Manager AIR application.

   If your server is running somewhere other than http://localhost:8080, you see errors stating that the application cannot connect to the server. Dismiss the error dialog and fill in the correct URL for the "Packager Server URL" in the Preferences Tab. If the server is running at the specified URL and the properties file is on the classpath, the Preferences screen will be populated with the values in the properties file. After you set the packager server URL, the AIR application remembers this setting, and you will not have to enter it the next time you launch the application.
4. Fill in the values in the Preferences tab and click Save. For an explanation of each parameter, see "Setting preferences" on page 75.
5. If you want to use the Watched Folders, you will need to restart the server to recover from the errors you saw in step 3. If the preferences are configured properly, no errors should appear during startup.

Setting preferences

With the exception of the Packager Server URL, all the preferences specified below are stored in the flashaccess-refimpl-packager.properties file on the server. All the settings can be modified either directly in the properties file or through the AIR application. Passwords are encrypted when they are stored in the properties file on the server. Type the unencrypted password into the UI, and it will be encrypted before it is stored in the file.

Note: All directories and paths refer to directories on the packager server, not on the client running the AIR application.

Any changes made here take effect immediately once the preferences are saved. There is no need to restart the server unless the Packager Thread terminated due to configuration problems.

The preference descriptions use the following terms:

Term	Description
Packager Server URL	Location of server running flashaccess-packager.war, for example, http://localhost:8080
Resource Directory	Directory containing policies, certificates, credentials, and any other resources required for the packager server

Packager Preferences

This tab contains settings required for packaging content. The following table describes these preferences:

Preference	Preference	Preference	Description
License Server URL			URL of the server from which the client should request a license, for example, http://mylicenseserver.com:8080
License Server Transport Certificate			The server transport certificate, issued by Adobe. This certificate is used to secure communications between the client and license server. The file must be located in the Resource Directory.
Enable HSM			Specifies whether certificates and credentials are stored on an HSM. If so, preferences related to certificates and credentials will be disabled, and the properties on the HSM tab must be specified.
Key Encryption Options			Specifies how the Content Encryption Key is encrypted at packaging time
	License Server Certificate		The License Server Certificate, issued by Adobe. The file must be located in the Resource Directory. The CEK is encrypted with the public key of the license server. Only holders of the license server private key may decrypt the CEK.
Packager Credential			The packager credential, issued by Adobe. This file is used to sign the metadata during packaging.
	File Name		The PKCS#12 (.pfx) file containing certificate and private key. The file must be located in the Resource Directory.
	File Password		Password for .pfx file
Global Watched Folder Properties			Specifies settings common to all Watched Folders configured on this server.
	Check Interval in Milliseconds		Specifies how often Watched Folders should check for new content to package. The server iterates through all the configured Watched Folders, then sleeps for this amount of time.
	Output File Name Suffix		Specifies a file extension to add to output files. For example, if ".out" is specified and the input file is "video.flv", the output file would be "video.out.flv".
	Backup Input Files		Specifies whether a copy of the original content should be saved. If this option is not selected, the original file will be deleted after packaging has been completed successfully.

Preference	Preference	Preference	Description
	Input Backup Subfolder Name		If the Backup Input Files option is selected, specifies a folder where input files will be saved. This option specifies a folder name relative to the Watched Folder input directory. If the folder does not exist, it will be created during packaging.
	Overwrite Existing Output Files		Specifies whether the output file may be overwritten if a file already exists with the same name. If this option is not selected and the output file already exists, processing of the input file will be skipped.

Policy Update List Preferences

This tab contains settings required for creating Policy Update Lists. The following table describes the preferences:

Preference	Description
License Server Credential	The License Server credential, issued by Adobe. This credential is used to sign Policy Update Lists.
File Name	The PKCS#12 (.pfx) file containing certificate and private key. The file must be located in the Resource Directory.
File Password	The password for .pfx file

HSM Preferences

Preferences in this tab only need to be specified if the "Enable HSM" checkbox is selected in the Packager tab. The following table describes these preferences:

Preference	Description
Sun PKCS#11 Config File Name	The full path to the Sun PKCS#11 provider's configuration file. See the Java PKCS#11 Reference Guide on Sun's website for details on the contents of this configuration file.
Partition Password	The password for the HSM partition specified in the PKCS#11 configuration file.
License Server Certificate Alias	Alias for Adobe-issued license server certificate stored on HSM. This certificate is used to encrypt the CEK during packaging. Specify this instead of "License Server Certificate" in the Packager tab.
License Server Transport Certificate Alias	Alias for Adobe-issued server transport certificate stored on HSM. This certificate is used to secure communications between the client and license server. Specify this instead of "License Server Transport Certificate" in the Packager tab.

Preference	Description
Packager Credential Alias	Alias for Adobe-issued packager credential (certificate and private key) stored on HSM. This is used to sign the metadata during packaging. Specify this instead of "Packager Credential" in the Packager tab.
License Server Credential Alias	Alias for Adobe-issued license server credential (certificate and private key) stored on HSM. This credential is used to sign Policy Update Lists. Specify this instead of "License Server Credential" in the Policy Update List tab. (This alias will likely be the same as "License Server Certificate Alias.)

Policy creation

Before any content can be packaged, one or more policies must be created. For an overview of the usage rules that may be specified in a policy, see "Usage rules" on page 7.

Create a new policy

To create a new policy, click New and enter a policy name. Fill in the desired policy attributes (all settings are optional). When done, click Save. The policy will be saved as policyname.pol in the Resource Directory.

Basic Policy Options

The following table describes the Basic Policy preferences:

Preference	Preference	Description
Policy Duration		Specifies the validity period of content protected with this policy.
	Start at	Licenses cannot be used until this date/time.
	End at	Licenses cannot be used after this date/time.
	End after	Specifies the amount of time a license is valid (in minutes), starting from the time it is packaged.
License Caching		Specifies whether licenses may be cached by the client.
	Delete at	Licenses cannot be used after this date/time.
	Delete after	Specifies the amount of time a license is valid (in minutes), starting from the time it the license is issued by the license server.
	Cache Indefinitely	License may be cached on the client indefinitely.
	No License Caching	License may not be cached by the client. A new license must be obtained from the server each time the user plays the content.
Authentication		
	Anonymous	No authentication is required to view the content.

Preference	Preference	Description
	Authenticated	Username/password authentication is required.
Enable License Chaining		Allows a license to be updated using a parent root license for batch updating of licenses. Once the leaf license expires, the server may issue the client a root license, which will renew all content protected with this policy.

Play Rights

The following table describes the Play Rights preferences:

Preference	Preference	Description
Playback Window		The duration a license is valid (in minutes) after the first time the user plays the protected content.
Output Protection		Controls whether output to external rendering devices should be protected. Analog and digital outputs can be specified independently.
Restrictions		Blacklist of client versions not permitted to play content. All columns are optional.
	DRM	Specifies a list of DRM versions that are not permitted to play protected content.
	Runtime	Specifies a list of Runtime versions that are not permitted to play protected content.
Minimum Security Level		
	DRM	Minimum DRM security level required to play protected content.
	Runtime	Minimum Runtime security level required to play protected content.
Allowed Applications		Whitelist of client applications permitted to play content. If not applications are specified, any SWF or AIR application is allowed.
	SWF	List of SWF URLs permitted to play protected content.
	AIR	List of AIR applications permitted to play protected content. Publisher ID is required, the remaining fields are optional.

Flash Access Manager supports policies containing multiple Play Rights. To create a policy with more than one Play Right, use the "Add additional Play Right" button, and fill in the desired attributes for each Play Right.

When consuming a license, the client uses the first Play Right for which it meets all the requirements. Multiple play rights may be used to specify different restrictions for different operating systems. For example, it is possible to specify one right with Output Protection required for Windows (by blacklisting

DRM versions on Macintosh and Linux) and to specify a second right with Output Protection "Use if available" on other platforms (by blacklisting DRM versions on Windows).

Custom Data

The following table describes the Custom Data preferences:

Preference	Description
Custom Policy Properties	Specify custom properties, which the license server may use when issuing licenses.
Custom License Properties	Specify custom properties, which will appear in the license issued to the client. Client applications will have access to these properties.

Update an existing policy

To update an existing policy, choose the filename from the drop down list and click Open. Modify any desired policy attributes. All attributes can be modified except those related to Authentication and License Chaining.

When done, click Save. The policy file in the Resource Directory will be replaced with the updated version.

Note: Even if the policy name is changed, the name of the file in the Resource Directory will not be modified.

Delete a policy

To delete an existing policy, choose the filename from the drop down list and click Delete.

Policy update list

You can use Policy Update Lists to communicate policy changes to a License Server. If a policy is modified after it is used to package content, it is desirable to have the License Server aware of the most recent version of the policy, so that version can be used to issue a license. For more information about Policy Update Lists, see Protecting Content.

To create a Policy Update List for the first time, click "Add policies" to view all available policies on the server. For any policies that have been updated since they were used to package content, select the "update" radio button.

If you no longer want to use a policy to issue any licenses and the policy was already used to package content, you may wish to revoke the policy. To do so, select the "revoke" radio button. When the desired policies have been selected, choose "Create Policy Update List". A file called PolicyUpdateList.dat will be saved in the Resources Directory.

To modify an existing Policy Update List, click "Add policies" to view all available policies on the server. Choose the additional policies to add or revoke. Existing entries in the Policy Update List can be changed in the upper section of the screen. Policies that are marked "updated" may be changed to "revoked", but once a policy is "revoked", it cannot be changed back to "updated".

When the desired changes have been made, choose "Create Policy Update List", and the PolicyUpdateList.dat file is regenerated. If a policy is already in the policy update list and was updated since the last time the list was generated, the most recent version of the policy will be used when the Policy Update List is generated again.

Package media

Use the Package Media tab to package content. The Packager Properties section displays the Packager settings that were entered in the Preferences tab. To modify these settings, go to the Preferences tab, change the settings, and Save.

If you want to package a single FLV or F4V file, choose the "Select Single File" option and enter the full path to the source file and full path where the encrypted file should be saved.

If you want to package all files in a folder, choose the "Select Single Folder" option. Specify the folder containing the source files. Only files in the Input Folder matching the "Input Media File Selection" criteria will be packaged (files in subfolders are not packaged). Choose to encrypt .flv files, .f4v files, or enter a custom regular expression (for example ".*" encrypts all files in the folder). The encrypted files will be saved in the specified output folder, using the same filename as the original file.

Note: File paths must refer to files available to the packaging server. If you are running the Flash Access Manager on a different machine than the packaging server, you must specify a path that is accessible by the server (either located on a network drive or on the server itself).

The following table describes the Package Media preferences:

Preference	Description
Policy File Name(s)	Select one or more policies from the drop-down list to apply to the content. To select multiple policies, hold down the CTRL key while selecting policies.
Seconds Unencrypted	Specifies the number of seconds of content to leave unencrypted at the beginning of the file. To encrypt starting from the beginning, enter "0".
Encrypt Video	Select this checkbox to encrypt video data
Encryption Level	If video encryption is enabled, select the encryption level for video data. High encrypts all video data. Medium and Low selectively encrypt portions of the video. (Only for F4V with H.264 video)
Encrypt Audio	Select this checkbox to encrypt audio data
Encrypt Script	Select this checkbox to encrypt script data (FLV only)
Custom Properties	Specify custom properties to include in the packaged content. These properties will be available to the license server when issuing a license. (Optional)

After the packaging options are selected, click the "Package Media" button to begin packaging the files.

Watched Folders

You can use Watched Folders to automatically package content created in certain folders. Each Watched Folder can be configured with different packaging options. To test packaging options before creating a Watched Folder, use the Package Media tab.

To create a Watched Folder, click "Add New Watched Folder" and fill in the packaging options. See the "Package Media" section for a description of each option. When done, click "Save Watched Folder Properties".

When a Watched Folder is saved, the packaging options are saved to *[Input Folder]*\packager\watchfolder.properties. Any content added to the Input Folder which meets the Input Media File Selection criteria will automatically be packaged and placed in the Output Folder. See the Global Watched Folder Preferences in the section "Packager Preferences" on page 75 to configure additional Watched Folder options.

To modify Watched Folder settings, select the Watched Folder input path from the list at the top of the screen. Modify the settings and click "Save Watched Folder Properties".

To delete a Watched Folder, select the Watched Folder input path from the list at the top of the screen and click "Delete Watched Folder Properties".