

# Chapter 2: Building and Deploying LiveCycle Data Services ES Applications

LiveCycle Data Services ES applications consist of two parts: client-side code and server-side code. Client-side code is a Flex application written in MXML and ActionScript and deployed as a SWF file. Server-side code is written in Java and deployed as Java class files or Java Archive (JAR) files. Every LiveCycle Data Services ES application has client-side code, however you can implement an entire application without writing any server-side code.

For more information on the general application and deployment process for Flex applications, see *Building and Deploying Adobe Flex 3 Applications*.

## Topics

Setting up your development environment .....	6
Using the TestDrive sample application .....	10
Building your client-side application .....	11
Building your server-side application .....	17
Debugging your application .....	19
Deploying your application .....	23

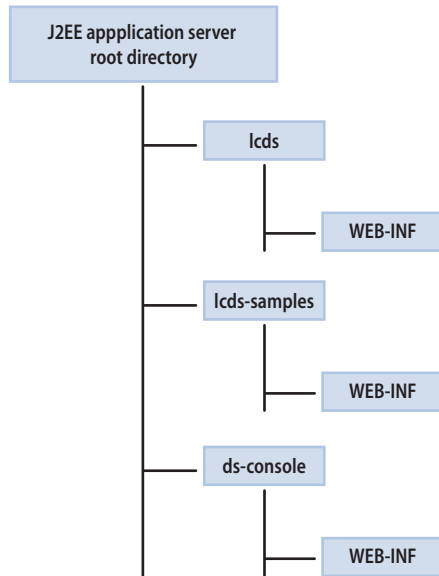
## Setting up your development environment

LiveCycle Data Services ES applications consist of two parts: client-side code and server-side code. Before you start developing your application, configure your development environment, including the directory structure for your source code; both for your client-side code and for your server-side code.

### Installation directory structure

The LiveCycle Data Services ES installer creates a directory structure on your computer that contains all of the resources necessary to build your application. As part of the installation, the installer creates three web applications that you can use as the basis of your development environment.

The following example shows the directory structure of the web applications installed with LiveCycle Data Services ES:



The installer gives you the option of installing the integrated Tomcat application server to host these web applications. Alternatively, you can install LiveCycle Data Services ES without installing Tomcat. Instead, you deploy the LiveCycle Data Services ES web application on your J2EE application server or servlet container.

The following table describes the directory structure of each web application:

Directory	Description
/lcds	The root directory of a web application. Contains the WEB-INF directory.
/lcds-samples	This directory also includes all files that must be accessible by the user's web browser, such as SWF files, JSP pages, HTML pages, Cascading Style Sheets, images, and JavaScript files. You can place these files directly in the web application root directory or in arbitrary subdirectories that do not use the reserved name WEB-INF.
/ds-console	
/META-INF	Contains package and extension configuration data.
/WEB-INF	Contains the standard web application deployment descriptor (web.xml) that configures the LiveCycle Data Services ES web application. This directory can also contain a vendor-specific web application deployment descriptor.
/WEB-INF/classes	Contains Java class files and configuration files.
/WEB-INF/flex	Contains LiveCycle Data Services ES configuration files.
/WEB-INF/flex/libs	Contains SWC library files used when compiling with Flex Builder.
/WEB-INF/flex/locale	Contains localization resource files used when compiling with Flex Builder.
/WEB-INF/lib	Contains LiveCycle Data Services ES JAR files.
/WEB-INF/src	Optionally contains Java source code used by the web application.

## Accessing a web application

To access a web application and the services provided by LiveCycle Data Services ES, you need the URL and port number associated with the web application. The following table describes how to access each web application assuming that you install LiveCycle Data Services ES with the integrated Tomcat application server.

**Note:** If you install LiveCycle Data Services ES into the directory structure of your J2EE application server or servlet container, modify the context root URL based on your development environment.

Application	Context Root URL for Tomcat	Description
Sample application	<code>http://localhost:8400/lcds-samples/</code>	<p>A sample web application that includes many LiveCycle Data Services ES examples. To start building your own applications, start by editing these samples.</p> <p>The root directory of the installed web application is <code>install_root\lcds-samples</code>. For example, if you install LiveCycle Data Services ES on Microsoft Windows with the integrated Tomcat application server, the root directory is <code>C:\lcds\tomcat\webapps\lcds-samples</code>.</p>
Template application	<code>http://localhost:8400/lcds/</code>	<p>A fully configured LiveCycle Data Services web application that contains no application code. You can use this application as a template to create your own web application.</p> <p>The root directory of the installed web application is <code>install_root\lcds</code>. For example, if you install LiveCycle Data Services ES on Microsoft Windows with the integrated Tomcat application server, the root directory is <code>C:\lcds\tomcat\webapps\lcds</code>.</p>
Console application	<code>http://localhost:8400/ds-console/</code>	<p>A console application that lets you view information about LiveCycle Data Services ES web applications.</p> <p>The root directory of the installed web application is <code>install_root\ds-console</code>. For example, if you install LiveCycle Data Services ES on Microsoft Windows with the integrated Tomcat application server, the root directory is <code>C:\lcds\tomcat\webapps\ds-console</code>.</p>

If you install LiveCycle Data Services ES with the integrated Tomcat application server, you can also access the ROOT web application by using the following URL:

`http://localhost:8400/`

## Creating a web application

To get started writing LiveCycle Data Services ES applications, you can edit the samples in the `lcds-samples` application, add your application code to the `lcds-samples` application, or add your application code to the empty `lcds` application. However, Adobe recommends leaving the `lcds` application alone, and instead copy its contents to a new web application. That leaves the `lcds` web application empty so that you can use as the template for creating web applications.

## Defining the directory structure for client-side code

You develop LiveCycle Data Services ES client-side applications in Flex, and compile them in the same way that you compile applications that use Flex SDK. That means you can use the compiler built in to Flex Builder, or the command line compiler, `mxmlc`, supplied with the Flex SDK.

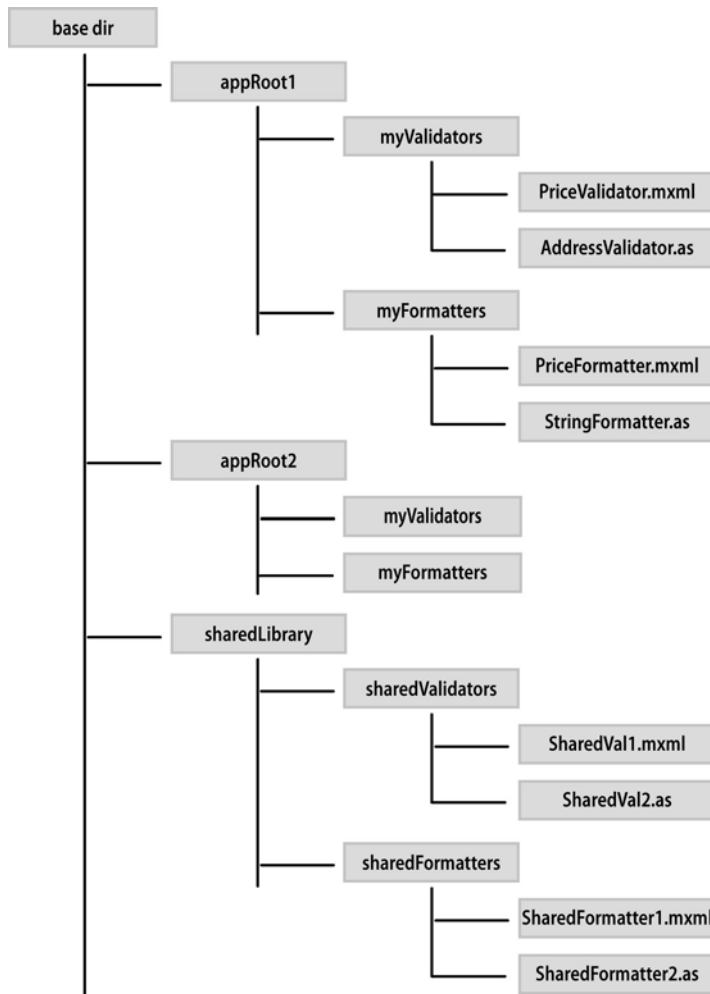
When you develop applications, you have two choices for how you arrange the directory structure of your application:

- Define a directory structure on your computer outside any LiveCycle Data Services ES web application. Compile the application into a SWF file and then deploy it, along with any runtime assets, to a LiveCycle Data Services ES web application.

- Define a directory structure in a LiveCycle Data Services ES web application. In this scenario, all of your source code and assets are stored in the web application. When you deploy the application, make sure to deploy only the application SWF file and runtime assets. Otherwise, you have the risk of deploying your source code on a production server.

You can define each application in its own directory structure, with the local assets for the application below the root directory. For assets shared across applications, such as image files, you can define a directory that is accessible by all applications.

The following example shows two applications, appRoot1 and appRoot2. Each application has a subdirectory for local MXML and ActionScript components, and can also reference a library of shared components:



### Defining the directory structure for server-side code

You develop the server-side part of an LiveCycle Data Services ES application in Java. For example, the client-side RemoteObject component lets you access the methods of server-side Java objects to return data to the client.

You also write Java classes to extend the functionality of the LiveCycle Data Services ES server. For example, a Data Management Service destination references one or more message channels that transport messages, and contains network- and server-related properties. It can also reference a *data adapter*, server-side code that lets the destination work with data through a particular type of interface such as a Java object. An *assembler class* is a Java class that interacts indirectly or directly with a data resource. For more information on assemblers, see [“Working with Data Adapters and Assemblers” on page 92](#).

When you develop server-side code, you have two choices for how you arrange the directory structure of your application:

- Define a directory structure that corresponds to the package hierarchy of your Java source code outside any LiveCycle Data Services ES web application. Compile the Java code and then deploy the corresponding class files and JAR files, along with any runtime assets, to a LiveCycle Data Services ES web application.
- Define a directory structure in a LiveCycle Data Services ES web application. In this scenario, all of your source code and assets are stored in the web application. When you deploy the application, make sure to deploy only the class and JAR files. Otherwise, you risk deploying source code on a production server.

Deploy compiled Java classes to the WEB-INF/classes or WEB-INF/lib directory of the web application. These directories are automatically included in the classpath of the web application. To deploy your server-side code, place class files in the WEB-INF/classes directory, in a directory structure that matches the package structure of the class. Place JAR files in the WEB-INF/lib directory.

## Using the TestDrive sample application

When you install LiveCycle Data Services ES, the installer creates the lcds-samples web application that contains running samples including the LiveCycle Data Services ES Test Drive application. Access the application by opening the following URL in a browser:

`http://localhost:8400/lcds-samples/testdrive.htm`

Many examples reference the Test Drive application. That makes it easier to get started developing applications because you only have to modify existing code, rather than creating it from scratch.

The client-side source code for these samples is shipped in the lcds-samples\WEB-INF\flex-src\flex-src.zip file. To modify the client-side code, extract the flex-src.zip file into the lcds-samples directory, and then edit, compile, and deploy the modified examples.

### Extract the client-side source code

- 1 Open lcds-samples\WEB-INF\flex-src\flex-src.zip file.
- 2 Extract the ZIP file into the lcds-samples directory.

Expanding the ZIP file adds a src directory to each sample in the lcds-samples directory. For example, the source code for the chat example, Chat.mxml, is written to the directory lcds-samples\testdrive-chat\src.

The server-side source code for these examples is shipped in the lcds-samples\WEB-INF\src\flex\samples directory. These source files are not zipped, but shipped in an expanded directory structure. To modify the server-side code you can edit and compile it in that directory structure, and then copy it to the lcds-samples directory to deploy it.

For more information on the sample applications, see [“Taking the LiveCycle Data Services ES Test Drive” on page 24](#).

### Run the Test Drive application

- 1 Change directory to `install_root/sampled`.
- 2 Start the sample database by using the following command:

```
startdb
```

You can stop the database by using the command:

```
stopdb
```

- 3 Start LiveCycle Data Services ES.
- 4 Open the following URL in a browser:  
<http://localhost:8400/lcds-samples/testdrive.htm>

## Building your client-side application

You write the client-side part of a LiveCycle Data Services ES application in Flex, and then use Flex Builder or the mxmclc command line compiler to compile it.

### Before you begin

Before you begin to develop your client-side code, understand the files required to perform the compilation, and make sure that you have configured your Flex installation to compile SWF files for LiveCycle Data Services ES applications.

#### Add the LiveCycle Data Services ES SWC files to the Flex SDK

To compile an application, Flex Builder and mxmclc reference the SWC library files that ships with the Flex 3 SDK in the frameworks/libs directory. However, LiveCycle Data Services ES ships additional SWC library files and SWC localization files that you must reference in the compilation:

- datavisualization.swc and fds.swc

The SWC library files that define LiveCycle Data Services ES. These SWC files must be included in the library path of the compiler. Typically you place them in the frameworks/libs directory of your Flex SDK, or specify the directory location of these SWC files by using the `library-path` option to the compiler. For more information on compiler options, see *Building and Deploying Adobe Flex 3 Applications*.

- airfds.swc and playerfds.swc

The SWC files required to build LiveCycle Data Services ES applications for Flash Player (`playerfds.swc`) or AIR (`airfds.swc`). One of these SWC files must be included in the library path of the compiler.

For the default Flex SDK installation, `playerfds.swc` must be in the `libs/player` directory, and `airfds.swc` must be in the `libs/air` directory. The `airfds.swc` and `playerfds.swc` must not both be available at the time of compilation. When you compile your application in Flex Builder, it automatically references the correct SWC file based on your project settings.

When you compile an application using mxmclc, by default the compiler references the `flex-config.xml` configuration file, which specifies to include the `libs/player` directory in the library path for Flash Player. When you compile an application for AIR, use the `load-config` option to the mxmclc compiler to specify the `air-config.xml` file, which specifies to include the `libs/air` directory in the library path.

- datavisualization\_rb.swc and fds\_rb.swc

The localization SWC files for LiveCycle Data Services ES. These SWC files must be in the library path of the compilation. Typically you place them in the `frameworks/locale/en_US` directory of your Flex SDK, or specify the directory location of these SWC files by using the `library-path` option to the compiler. For more information on localization, see *Adobe Flex 3 Developer Guide*.

### Add the necessary SWC files to the Flex 3 SDK

- 1 Unzip `install_root/resources/flex_sdk/flex_sdk_3.zip` to `install_root/resources/flex_sdk`, where `install_root` is the LiveCycle Data Services ES installation directory. For example, the default value of `install_root` is `c:\lcds` on Microsoft Windows.
- 2 Copy `install_root/resources/frameworks/libs/datavisualization.swc` and `fds.swc` to `install_root/resources/flex_sdk/frameworks/libs`.
- 3 Copy `install_root/resources/frameworks/libs/air/airfds.swc` to `install_root/resources/flex_sdk/frameworks/libs/air`.
- 4 Copy `install_root/resources/frameworks/libs/player/playerfds.swc` to `install_root/resources/flex_sdk/frameworks/libs/player`.
- 5 Copy `install_root/resources/frameworks/libs/locale/en_US/datavisualization_rb.swc` and `fds_rb.swc` to `install_root/resources/flex_sdk/frameworks/locale/en_US`.

### Specifying the services-config.xml file in a compilation

When you compile your Flex application, you typically specify the `services-config.xml` configuration file to the compiler. This file defines the channel URLs that the client-side Flex application uses to communicate with the LiveCycle Data Services ES server. These channel URLs are then compiled into the resultant SWF file.

Both client-side and server-side code use the `services-config.xml` configuration file. If you change anything in `services-config.xml`, you typically have to recompile your client-side applications and restart your server-side application for the changes to take effect.

In Flex Builder, the appropriate `services-config.xml` file is included automatically based on the LiveCycle Data Services ES web application that you specified in the configuration of your Flex Builder project. When you use the `mxmmlc` compiler, use the `services` option to specify the location of the file.

*Note:* You can also create channel definitions at runtime in ActionScript. In that case, you might be able to omit the reference to the `services-config.xml` configuration file from the compiler. For more information, see [“Run-Time Configuration” on page 202](#).

### Specifying the context root in a compilation

The `services-config.xml` configuration file typically uses the `context.root` token to specify the context root of a web application. At compile time, you use the compiler `context-root` option to specify that information.

During a compilation, Flex Builder automatically sets the value of the `context.root` token based on the LiveCycle Data Services ES web application that you specified in the configuration of your project. When you use the `mxmmlc` compiler, use the `context-root` option to set it.

## Using Flex Builder to compile client-side code

Adobe Flex Builder is an integrated development environment (IDE) for developing applications that use the Adobe Flex framework, MXML, Adobe Flash Player 9, AIR 1.0, ActionScript 3.0, Adobe LiveCycle Data Services ES, and the Adobe Flex Charting components.

Flex Builder is built on top of Eclipse, an open-source IDE. It runs on Microsoft Windows, Apple Mac OS X, and Linux, and is available in several versions. Installation configuration options let you install Flex Builder as a plug-in to an existing Eclipse workbench installation, or to install it as a stand-alone application.

For more information, see *Using Adobe Flex Builder 3*.

### Using Flex Builder standalone or plugin

The Flex Builder installer provides the following two configuration options:

**Plug-in configuration** This configuration is for users who already use the Eclipse workbench, who already develop in Java, or who want to add the Flex Builder plug-ins to their toolkit of Eclipse plug-ins. Because Eclipse is an open, extensible platform, hundreds of plug-ins are available for many different development purposes.

**Stand-alone configuration** This configuration is a customized packaging of Eclipse and the Flex Builder plug-in created specifically for developing Flex and ActionScript applications. The stand-alone configuration is ideal for new users and users who intend to develop only Flex and ActionScript applications.

Both configurations provide the same functionality and you select the configuration you want when installing Flex Builder.

Most LiveCycle Data Services ES developers choose to use the Eclipse plug-in configuration so that they can develop the Java code that runs on the server in the same IDE that they use to develop the MXML and ActionScript code for the client Flex application.

*Note: While the standalone version of Flex Builder does not contain tools to edit Java code, you can install them. Use the Help > Software Updates > Find and Install menu command to open the Install/Update dialog box, and then select Search for new features to install. In the results, choose Europa Discovery Site, and then select the Java Development package to install.*

If you aren't sure which configuration to use, follow these guidelines:

- If you already use and have Eclipse 3.11 (or later) installed, select the plug-in configuration. On Macintosh, Eclipse 3.2 is the minimum version.
- If you don't have Eclipse installed and your primary focus is on developing Flex and ActionScript applications, select the stand-alone configuration. This configuration also allows you to install other Eclipse plug-ins, so you can expand the scope of your development work in the future.

### Adding the Flex SDK to Flex Builder

Before you can build your first LiveCycle Data Services ES application in Flex Builder, add the Flex 3 SDK to Flex Builder that contains the LiveCycle Data Services SWC files. These instructions describe how to add the Flex SDK to Flex Builder that you configured as described in [“Add the LiveCycle Data Services ES SWC files to the Flex SDK” on page 11](#).

### Add the Flex SDK

You only have to perform this procedure once, before you create your first LiveCycle Data Services ES application.

- 1 Start Flex Builder
- 2 Select Window > Preferences to open the Preferences dialog box.
- 3 Select Flex > Install Flex SDKs to open the Installed Flex SDKs dialog box.
- 4 Click Add.
- 5 Specify `install_root/resources/flex_sdk` for the Flex SDK location in the Add Flex SDK dialog box, where `install_root` is the LiveCycle Data Services ES installation directory. For example, the default value of `install_root` is `c:\lcds` on Microsoft Windows.
- 6 Specify LCDS for the Flex SDK name in the Add Flex SDK dialog box.
- 7 Click OK.
- 8 (Optional) Select the checkbox next to the Flex LCDS SDK to make it the default SDK for new Flex projects.
- 9 Click OK.

## Creating a Flex Builder project

This procedure describes how to create a Flex Builder project to edit one of the samples shipped as part of the Test Drive application. While this procedure creates a Flex Builder project to edit an existing file, the procedure for creating and configuring a new project is almost the same. For more information on the Test Drive application, see [“Using the TestDrive sample application” on page 10](#).

### Complete the following steps to create the project

- 1 Start Flex Builder.
- 2 Select File > New > Flex Project.
- 3 Enter a project name. Since you are editing an existing application, use the exact name of the sample folder: testdrive-chat.

*Note: If you are creating a new empty project, you can name it anything that you want.*

- 4 If you unzipped flex-src.zip in the lcds-samples directory, deselect the use default location checkbox, and specify the directory as C:\lcds\tomcat\webapps\lcds-samples\testdrive-chat, or wherever you unzipped the file on your computer.

*Note: By default, Flex Builder creates the project directory based on the project name and operating system. For example, if you are using the plugin version of Flex Builder on Microsoft Windows, the default project directory is C:\Documents and Settings\USER\_NAME\workspace\PROJECT\_NAME.*

- 5 Select the application type as Web application (runs in Flash Player) to configure the application to run in the browser as a Flash Player application.

If you are creating an AIR application, choose Desktop application (runs in Adobe AIR). However, make sure that you do not have any server tokens in URLs in the service configuration files. In the web applications that ships with LiveCycle Data Services ES, server tokens are used in the channel endpoint URLs in the WEB-INF/flex/services-config.xml file, as the following example shows:

```
<endpoint
url="https://{server.name}:{server.port}/{context.root}/messagebroker/streamingamf"
class="flex.messaging.endpoints.StreamingAMFEndpoint"/>
```

You would change that line to the following:

```
<endpoint url="http://your_server_name:8400/lcds/messagebroker/streamingamf"
class="flex.messaging.endpoints.StreamingAMFEndpoint"/>
```

- 6 Select J2EE as the Application server type.
- 7 Select Use remote object access.
- 8 Select LiveCycle Data Services.
- 9 If the option is available, deselect Create combined Java/Flex Project with WTP.
- 10 Click Next.
- 11 Deselect Use default location for local LiveCycle Data Services server.
- 12 Set the root folder, root URL, and context root of your web application.

The root folder specifies the top-level directory of the web application (the directory that contains the WEB-INF directory). The root URL specifies the URL of the web application, and the context root specifies the root of the web application.

If you are using the integrated Tomcat application server, set these properties as shown follows:

Root folder: C:\lcds\tomcat\webapps\lcds-samples\

Root URL: http://localhost:8400/lcds-samples/

Context root: /lcds-samples/

Modify these settings as appropriate if you are not using the Tomcat application server.

**13** Make sure that your LiveCycle Data Services ES server is running, and click Validate Configuration to ensure that your project is valid.

**14** Make sure the option to compile the application locally in Flex Builder is selected.

**15** Clear the Output folder field to set the directory of the compiled SWF file to the main project directory.

By default, Flex Builder writes the compiled SWF file to the bin-debug directory under the main project directory. If you want to use a different output directory, set it using Output folder.

**16** Click Next.

**17** Set the name of the main application file to Chat.mxml.

**18** Click Finish.

After Flex Builder creates your project, make sure that your project uses the LiveCycle Data Services ES SDK to compile it:

**19** Select the project.

**20** Select Project > Properties to open the Project Properties dialog box.

**21** Select Flex Compiler.

**22** Choose "Use a specific SDK" under Flex SDK version.

**23** Select LCDS, or whatever you named the LCDS SDK, when you performed the procedure in [“Adding the Flex SDK to Flex Builder”](#) on page 13.

**24** Click OK.

You can now create, compile, and run an application that uses LCDS.

### Editing a LiveCycle Data Services ES application in Flex Builder

After you create the Flex Builder project to edit the Chat.mxml file, use the following procedure to edit and deploy the file:

**1** Open src\Chat.mxml in your Flex Builder project.

**2** Edit Chat.mxml to change the definition of the TextArea control so that it displays an initial text string when the application starts:

```
<mx:TextArea id="log" width="100%" height="100%" text="My edited file!"/>
```

**3** Save the file.

When you save the file, Flex Builder automatically compiles it. By default, the resultant SWF file is written to the C:\lcds\tomcat\webapps\lcds-samples\testdrive-chat\bin-debug directory, or to where you set the Output directory for the project.

*Note:* If you write the Chat.SWF file to any directory other than lcds-samples\testdrive-chat, deploy the SWF file by copying it to the lcds-samples\testdrive-chat directory.

**4** Make sure that you started the samples database and LiveCycle Data Services ES as described in [“Using the TestDrive sample application”](#) on page 10.

**5** Select Run > Run (or Control+F11) to run the application.

You can also request the application in a browser by using the URL <http://localhost:8400/lcds-samples/testdrive-chat/index.html>

**6** Make sure that your new text appears in the TextArea control.

## Editing config files as linked resources in Flex Builder

While working on the client-side of your applications, you often look at or change the LiveCycle Data Services ES configuration files. You can create a linked resource inside a Flex Builder project to make the LiveCycle Data Services ES configuration files easily accessible.

### Create a linked resource to the LiveCycle Data Services ES configuration files

- 1 Right-click the project name in the project navigation view.
- 2 Select New > Folder in the popup menu.
- 3 Specify the name of the folder as it will appear in the navigation view. This name can be different from the name of the folder in the file system. For example type server-config.
- 4 Click the Advanced button.
- 5 Check Link to folder in the file system.
- 6 Click the Browse button and select the flex folder under the WEB-INF directory of your web application. For example, on a typical Windows installation using the Tomcat integrated server, select `install_root/tomcat/webapps/lcds-samples/WEB-INF/flex`.
- 7 Click Finish. The LiveCycle Data Services ES configuration files are now available in your Flex Builder project under the server-config folder.

*Note:* If you change anything in `services-config.xml`, you typically have to recompile your client-side applications and restart your server-side application for the changes to take effect.

## Using mxmcl to compile client-side code

You use the mxmcl command line compiler to create SWF files from MXML, AS, and other source files. Typically, you pass the name of MXML file containing an `<mx:Application>` tag to the compiler. The output is a SWF file. The mxmcl compiler ships in the bin directory of the Flex SDK. You can run the mxmcl compiler as a shell script and executable file for use on Windows and UNIX systems. For more information, see *Building and Deploying Adobe Flex 3 Applications*.

The basic syntax of the mxmcl utility is as follows:

```
mxmcl [options] target_file
```

The target file of the compile is required. If you use a space-separated list as part of the options, you can terminate the list with a double hyphen before adding the target file; for example:

```
mxmcl -option arg1 arg2 arg3 -- target_file.mxml
```

To see a list of options for mxmcl, you can use the `help list` option, as the following example shows:

```
mxmcl -help list
```

To see a list of all options available for mxmcl, including advanced options, you use the following command:

```
mxmcl -help list advanced
```

The default output of mxmcl is `filename.swf`, where `filename` is the name of the target file. The default output location is in the same directory as the target, unless you specify an output file and location with the `output` option.

The mxmcl command line compiler does not generate an HTML wrapper. Create your own wrapper to deploy a SWF file that the mxmcl compiler produced. The wrapper is used to embed the SWF object in the HTML tag. It includes the `<object>` and `<embed>` tags, as well as scripts that support Flash Player version detection and history management. For information about creating an HTML wrapper, see *Building and Deploying Adobe Flex 3 Applications*.

*Note:* Flex Builder does automatically generate an HTML wrapper when you compile your application.

### Compiling LiveCycle Data Services ES applications

Along with the standard options that you use with the mxmhc compiler, use the following options to specify information about your LiveCycle Data Services ES application.

- `services filename`  
Specifies the location of the services-config.xml file.
- `context-root context-path`  
Sets the value of the context root of the application. This value corresponds to the `{context.root}` token in the services-config.xml file, which is often used in channel definitions. The default value is null.

### Edit, compile, and deploy the Chat.mxml file

- 1 Unzip flex-src.zip in the `lcds\tomcat\webapps\lcds-samples` directory as described in [“Using the TestDrive sample application” on page 10](#).
- 2 Open the file `C:\lcds\tomcat\webapps\lcds-samples\testdrive-chat\src\Chat.mxml` in an editor. Modify this path as necessary based on where you unzipped flex-src.zip.
- 3 Change the definition of the TextArea control so that it displays an initial text string when the application starts:  

```
<mx:TextArea id="log" width="100%" height="100%" text="My edited file!"/>
```
- 4 Change directory to `C:\lcds\tomcat\webapps\lcds-samples`.
- 5 Use the following command to compile Chat.mxml:

**Note:** This command assumes that you added the mxmhc directory to your system path. The default location is `install_root\resources\flex_sdk\bin`.

```
mxmhc -strict=true  
      -show-actionscript-warnings=true  
      -use-network=true  
      -services=WEB-INF/flex/services-config.xml  
      -context-root=lcds-samples  
      -output=testdrive-chat/Chat.swf  
      testdrive-chat/src/Chat.mxml
```

The compiler writes the Chat.swf file to the `lcds-samples\testdrive-chat` directory.

- 6 Start the samples database and LiveCycle Data Services ES as described in [“Using the TestDrive sample application” on page 10](#).
- 7 Request the application by using the URL `http://localhost:8400/lcds-samples/testdrive-chat/index.html`
- 8 Make sure that your new text appears in the TextArea control.

Rather than keeping your source code in your deployment directory, you can instead set up a separate directory, and then copy Chat.swf to `lcds-samples\testdrive-chat` to deploy it.

## Building your server-side application

You write the server-side part of a LiveCycle Data Services ES application in Java, and then use the javac compiler to compile it.

### Create a simple Java class to return data to the client

A common reason to create a server-side Java class is to represent data returned to the client. For example, the client-side RemoteObject component lets you access the methods of server-side Java objects to return data to the client.

The Test Drive sample application contains the sample *Accessing data using Remoting* where the client-side code uses the RemoteObject component to access product data on the server. The Product.java class represents that data. After starting the LiveCycle Data Services ES server and the samples database, you can see this example by opening the following URL in a browser: <http://localhost:8400/lcds-samples/testdrive-remoteobject/index.html>.

The source code for Product.java is in the `install_root\lcds-samples\WEB-INF\src\flex\samples\product` directory. For example, if you installed LiveCycle Data Services ES with the integrated Tomcat server on Microsoft Windows, the directory is `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\src\flex\samples\product`.

### Modify Product.java, compile it, and then deploy it on the lcds-sample server

- 1 Start the samples database.
- 2 Start LiveCycle Data Services ES.
- 3 View the running example by opening the following URL in a browser:  
<http://localhost:8400/lcds-samples/testdrive-remoteobject/index.html>
- 4 Click the Get Data button to download data from the server. Notice that the description column contains product descriptions.
- 5 In an editor, open the file `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\src\flex\samples\product\Product.java` (modify this path as necessary for your particular installation).
- 6 Modify the following `getDescription()` method definition so that it always returns the String "My description" rather than the value from the database:

```
public String getDescription() {  
    return description;  
}
```

The modified method definition appears as the following:

```
public String getDescription() {  
    return "My description.";  
}
```

- 7 Change directory to `lcds-samples`.
- 8 Compile `Product.java` by using the following `javac` command line:  

```
javac -d WEB-INF/classes/ WEB-INF/src/flex/samples/product/Product.java
```

This command creates the file `Product.class`, and deploys it to the `WEB-INF\classes\flex\samples\product` directory.
- 9 View the running example by opening the following URL in a browser:  
<http://localhost:8400/lcds-samples/testdrive-remoteobject/index.html>
- 10 Click the Get Data button.  
Notice that the description column now contains the String "My description" for each product.

### Creating a Java class that extends a LiveCycle Data Services ES class

As part of developing your server-side code, you can create a custom assembler class, factory class, or other type of Java class that extends the LiveCycle Data Services ES Java class library. For example, a *data adapter* is responsible for updating the persistent data store on the server in a manner appropriate to the specific data store type.

You perform many of the same steps to compile a Java class that extends the LiveCycle Data Services ES Java class library as you do for compiling a simple class as described in the previous topic. The one major difference is to make sure that you include the appropriate LiveCycle Data Services ES JAR files in the classpath of your compilation so that the compiler can locate the appropriate files.

The Test Drive sample application contains the sample *Data Management Service* where the server-side code uses a custom assembler represented by the `ProductAssembler.java` class. After starting the LiveCycle Data Services ES server and the samples database, you can see this example by opening the following URL in a browser:  
`http://localhost:8400/lcds-samples/testdrive-dataservice/index.html`.

The source code for `ProductAssembler.java` is in the directory `install_root\lcds-samples\WEB-INF\src\flex\samples\product`. For example, if you installed LiveCycle Data Services ES with the integrated Tomcat server on Microsoft Windows, the directory is `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\src\flex\samples\product`.

### Compile `ProductAssembler.java` and deploy it on the lcds-sample server

- 1 View the running example by opening the following URL in a browser:  
`http://localhost:8400/lcds-samples/testdrive-dataservice/index.html`
- 2 Click the Get Data button to download data from the server.
- 3 In an editor, open the file `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\src\flex\samples\product\ProductAssembler.java` (modify this path as necessary for your particular installation).  
For more information on assemblers, see [“Working with Data Adapters and Assemblers” on page 92](#).
- 4 Change directory to `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\src`.
- 5 Compile `ProductAssembler.java` by using the following `javac` command line:

```
javac
  -sourcepath .
  -d ..\classes
  -classpath ..\lib\flex-messaging-common.jar;
    ..\lib\flex-messaging-core.jar;
    ..\lib\flex-messaging-data-req.jar;
    ..\lib\flex-messaging-data.jar;
    ..\lib\flex-messaging-opt.jar;
    ..\lib\flex-messaging-proxy.jar;
    ..\lib\flex-messaging-remoting.jar
    flex\samples\product\ProductAssembler.java
```

Notice that the classpath contains many, but not all, of the JAR files in `C:\lcds\tomcat\webapps\lcds-samples\WEB-INF\lib`. If you are compiling other types of classes, you include additional JAR files in the classpath.

This command creates the file `ProductAssembler.class` in the same directory as `ProductAssembler.java`.

- 6 View the running example by opening the following URL in a browser:  
`http://localhost:8400/lcds-samples/testdrive-dataservice/index.html`
- 7 Click the Get Data button to make sure that your code is working correctly.

## Debugging your application

If you encounter errors in your applications, you can use the debugging tools to set and manage breakpoints in your code; control application execution by suspending, resuming, and terminating the application; step into and over the code statements; select critical variables to watch; evaluate watch expressions while the application is running; and so on.

Debugging Flex applications can be as simple as enabling `trace()` statements or as complex as stepping into a source files and running the code, one line at a time. The Flex Builder debugger and the command line debugger, `fdb`, let you step through and debug ActionScript files used by your Flex applications. For information on how to use the Flex Builder debugger, see *Using Adobe Flex Builder 3*. For more information on the command line debugger, `fdb`, see *Building and Deploying Adobe Flex 3 Applications*.

## Using the debugger version of Flash Player

To use the `fdb` command line debugger or the Flex Builder debugger, install and configure the debugger version of Flash Player. To determine if you are running the debugger version or the standard version of Flash Player, open any Flex application in Flash Player and right-click. If you see the Show Redraw Regions option, you are running the debugger version of Flash Player. For more information about installing the debugger version of Flash Player, see the LiveCycle Data Services ES installation instructions.

The debugger version of Flash Player comes in ActiveX, Plug-in, and stand-alone versions for Microsoft Internet Explorer, Netscape-based browsers, and desktop applications, respectively. You can find the debugger version of Flash Player installers in the following locations:

- Flex Builder: `install_dir/Player/os_version`
- Flex SDK: `install_dir/runtimes/player/os_version/`

Like the standard version of Adobe Flash Player 9, it runs SWF files in a browser or on the desktop in a stand-alone player. Unlike Flash Player, the debugger version of Flash Player enables you to do the following:

- Output statements and application errors to the debugger version of Flash Player local log file by using the `trace()` method.
- Write data services log messages to the local log file of the debugger version of Flash Player.
- View run-time errors (RTEs).
- Use the `fdb` command line debugger.
- Use the Flex Builder debugging tool.
- Use the Flex Builder profiling tool.

**Note:** ADL logs `trace()` output from AIR applications.

## Using logging to debug your application

One tool that can help in debugging is the logging mechanism. You can perform both server-side and client-side logging of requests and responses.

### Client-side logging

For client-side logging, you can directly write messages to the log file, or configure the application to write messages generated by Flex to the log file. The debugger version of Flash Player has two primary methods of writing messages to a log file:

- The global `trace()` method. The global `trace()` method prints a String to the log file. Messages can contain checkpoint information to signal that your application reached a specific line of code, or the value of a variable.
- Logging API. The logging API, implemented by the `TraceTarget` class, provides a layer of functionality on top of the `trace()` method. For example, you can use the logging API to log debug, error, and warning messages generated by Flex during application execution.

The debugger version of Flash Player sends logging information to the flashlog.txt file. The operating system determines the location of this file, as the following table shows:

Operating System	Log file location
Windows 95/98/ME/2000/XP	C:\Documents and Settings\username\Application Data\Macromedia\Flash Player\Logs
Windows Vista	C:\Users\username\AppData\Roaming\Macromedia\Flash Player\Logs
Mac OS X	/Users/username/Library/Preferences/Macromedia/Flash Player/Logs/
Linux	/home/username/.macromedia/Flash_Player/Logs/

Use settings in the mm.cfg text file to configure the debugger version of Flash Player for logging. If this file does not exist, you can create it when you first configure the debugger version of Flash Player. The location of this file depends on your operating system. The following table shows where to create the mm.cfg file for several operating systems:

Operating system	Create file in ...
Mac OS X	/Library/Application Support/Macromedia
Windows 95/98/ME	%HOMEDRIVE%\%HOMEPATH%
Windows 2000/XP	C:\Documents and Settings\username
Windows Vista	C:\Users\username
Linux	/home/username

The mm.cfg file contains many settings that you can use to control logging. The following sample mm.cfg file enables error reporting and trace logging:

```
ErrorReportingEnable=1
TraceOutputFileEnable=1
```

Once enabled, you can call the `trace()` method to write a String to the flashlog.txt file, as the following example shows:

```
trace("Got to checkpoint 1.");
```

Insert the following MXML line to enable the logging of all Flex-generated debug messages to flashlog.txt:

```
<mx:TraceTarget loglevel="2"/>
```

For information about client-side logging, see *Building and Deploying Adobe Flex 3 Applications*.

### Server-side logging

You configure server-side logging in the logging section of the services configuration file, services-config.xml. By default, output is sent to System.out.

You set the logging level to one of the following available levels:

- all
- debug
- info
- warn
- error
- none

You typically set the server-side logging level to debug to log all debug messages, and also all info, warning, and error messages. The following example shows a logging configuration that uses the Debug logging level:

```
<logging>
```

```

<!-- You may also use flex.messaging.log.ServletLogTarget. -->
  <target class="flex.messaging.log.ConsoleTarget" level="Debug">
    <properties>
      <prefix>[Flex]</prefix>
      <includeDate>>false</includeDate>
      <includeTime>>false</includeTime>
      <includeLevel>>false</includeLevel>
      <includeCategory>>false</includeCategory>
    </properties>
    <filters>
      <pattern>Endpoint</pattern>
      <!--<pattern>Service.*</pattern>-->
      <!--<pattern>Message.*</pattern>-->
    </filters>
  </target>
</logging>

```

After you edit `services-config.xml` to enable logging, restart the LiveCycle Data Services ES server. For information about server-side logging, see [“Configuring LiveCycle Data Services ES” on page 251](#).

## Measuring application performance

As part of preparing your application for final deployment, you can test its performance to look for ways to optimize it. One place to examine performance is in the message processing part of the application. To help you gather this performance information, enable the gathering of message timing and sizing data.

The mechanism for measuring message processing performance is disabled by default. When enabled, information regarding message size, server processing time, and network travel time is available to the client that pushed a message to the server, to a client that received a pushed message from the server, or to a client that received an acknowledge message from the server in response a pushed message. A subset of this information is also available for access on the server.

You can use this mechanism across all channel types, including polling and streaming channels, that communicate with the server. However, this mechanism does not work when you make a direct connection to an external server by setting the `useProxy` property to `false` for the `HTTPService` and `WebService` tags because it bypasses the LiveCycle Data Services ES Proxy Server.

You use two parameters in a channel definition to enable message processing metrics:

- `<record-message-times>`
- `<record-message-sizes>`

Set these parameters to `true` or `false`; the default value is `false`. You can set the parameters to different values to capture only one type of metric. For example, the following channel definition specifies to capture message timing information, but not message sizing information:

```

<channel-definition id="my-streaming-amf"
  class="mx.messaging.channels.StreamingAMFChannel">
  <endpoint
    url="http://{server.name}:{server.port}/{context.root}/messagebroker/streamingamf"
    class="flex.messaging.endpoints.StreamingAMFEndpoint"/>
  <properties>
    <record-message-times>true</record-message-times>
    <record-message-sizes>>false</record-message-sizes>
  </properties>
</channel-definition>

```

## Deploying your application

When it is time to deploy your LiveCycle Data Services ES application in a production environment, ensure that you deploy all the necessary parts of the application, including the following:

- The compiled SWF file containing your client-side application
- The HTML wrapper generated by Flex Builder or created manually if you use the mxmhc compiler
- The compiled Java class and JAR files that represent your server-side application
- Any runtime assets required by your application
- A LiveCycle Data Services ES web application
- Update LiveCycle Data Services ES configuration files that contain the necessary information to support your application

How you choose to perform your deployment is going to be based on your production environment. One option is to package your application and assets in a LiveCycle Data Services ES web application, and then create a single WAR file that contains the entire web application. You can then deploy the single WAR file on your production server.

Alternatively, you can deploy multiple LiveCycle Data Services ES applications on a single web application. In this case, your production server has an expanded LiveCycle Data Services ES web application on which you add the directories required to run your new application.