



# **Creating Flex® Applications Enabled for LiveCycle® Workspace ES**

July 2008

**Adobe® LiveCycle® Workspace ES**

Version 8.2

© 2008 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle® Workspace ES Creating Flex® Applications Enabled for LiveCycle Workspace ES for Microsoft® Windows®, Linux®, and UNIX®

Edition 2.1, July 2008

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end-user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names, company logos and user names in sample material or sample forms included in this documentation and/or software are for demonstration purposes only and are not intended to refer to any actual organization or persons.

Adobe, the Adobe logo, Acrobat, Distiller, Flash, Flex, FrameMaker, LiveCycle, PageMaker, Photoshop, PostScript, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the US and other countries.

All other trademarks are the property of their respective owners.

This product contains either BSAFE and/or TIPEM software by RSA Security, Inc.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

This product includes software developed by the IronSmith Project (<http://www.ironsmith.org/>).

This product includes software developed by the OpenSymphony Group (<http://www.opensymphony.com/>).

This product includes software developed by the Indiana University Extreme! Lab (<http://www.extreme.indiana.edu/>).

This product includes copyrighted software developed by E. Wray Johnson for use and distribution by the Object Data Management Group (<http://www.odmg.org/>).

Portions © Eastman Kodak Company, 199- and used under license. All rights reserved. Kodak is a registered trademark and Photo CD is a trademark of Eastman Kodak Company.

Powered by Celequest. Copyright 2005-2008 Adobe Systems Incorporated. All rights reserved. Contains technology distributed under license from Celequest Corporation. Copyright 2005 Celequest Corporation. All rights reserved.

Single sign-on, extending Active Directory to Adobe LiveCycle ES provided by Quest Software “[www.quest.com/identity-management](http://www.quest.com/identity-management)” in a subsequent minor release that is not a bug fix (i.e., version 1.1 to 1.2 but not 1.1.1 to 1.1.2) of the Licensee Product that incorporates the Licensed Product.

The Spelling portion of this product is based on Proximity Linguistic Technology.

©Copyright 1989, 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1990 Merriam-Webster Inc. © Copyright 1990 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2003 Franklin Electronic Publishers Inc. © Copyright 2003 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2004 Franklin Electronic Publishers, Inc. © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1991 Dr.Lluís de Yzaguirre I Maura © Copyright 1991 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1990 Munksgaard International Publishers Ltd. © Copyright 1990 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1995 Van Dale Lexicografie bv © Copyright 1996 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1990 IDE a.s. © Copyright 1990 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2004 Franklin Electronics Publishers, Inc. © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1992 Hachette/Franklin Electronic Publishers, Inc. © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2004 Bertelsmann Lexikon Verlag © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2004 MorphoLogic Inc. © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 1990 Williams Collins Sons & Co. Ltd. © Copyright 1990 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA. © Copyright 1993-95 Russicon Company Ltd.

© Copyright 1995 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

© Copyright 2004 IDE a.s. © Copyright 2004 All Rights Reserved Proximity Technology A Division of Franklin Electronic Publishers, Inc. Burlington, New Jersey USA.

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

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

---

# Contents

---

<b>About This Document.....</b>	<b>5</b>
Who should read this document? .....	5
Before you begin .....	5
Additional information .....	5
<b>1 Introduction .....</b>	<b>7</b>
Creating a Flex application enabled for Workspace ES .....	8
Enabling an existing Flex application for Workspace ES .....	8
Sample files.....	9
<b>2 Configuring Your Development Environment .....</b>	<b>10</b>
Installing the Flex SDK for LiveCycle ES.....	10
Configuring Flex Builder to use the Flex SDK for LiveCycle ES .....	11
<b>3 Configuring your Flex projects .....</b>	<b>13</b>
Including the Workspace API in Flex projects.....	13
Using the Workspace API in Flex applications.....	14
<b>4 Creating Flex Applications for Data Capture .....</b>	<b>16</b>
Creating forms as Flex applications.....	17
Adding data models .....	19
Adding form validation .....	22
<b>5 Enabling Flex Applications for Workspace ES .....</b>	<b>24</b>
Adding the FormConnector component.....	26
Validating data stored in Flex applications.....	27
Indicating that form data stored in Flex applications have changed .....	28
Creating events listeners to handle events from Workspace ES .....	31
<b>6 Deploying and Testing Flex Applications in Workspace ES .....</b>	<b>36</b>
Deploying Flex applications .....	36
Testing Flex applications .....	37
Troubleshooting .....	38
<b>A Sample Code of Flex Application enabled in LiveCycle Workspace ES .....</b>	<b>39</b>

# About This Document

This document describes how to perform the following tasks:

- Create an Adobe Flex® application to provide form-like capabilities for capturing data for process management.
- Enable a Flex application for use within Adobe® LiveCycle® Workspace ES.

## Who should read this document?

This document is intended for programmers who are familiar with Flex, ActionScript™ 3.0, MXML, XML, and Adobe Flex® Builder™ (or the Adobe Flex SDK compiler) who want to develop a Flex application that functions within Workspace ES.

It is beneficial if programmers have had exposure to Adobe® LiveCycle® ES (Enterprise Suite) Update 1 and Adobe LiveCycle Workbench ES, and are familiar with Workspace ES.

## Before you begin

Before you can begin creating Flex applications enabled for use in Workspace ES, you must have access to the following items:

- The Flex 3.0.1 SDK that is compatible with LiveCycle ES, Flex Builder 3.0.1, or the Eclipse Flex 3.0.1 plug-in. The Flex 3.0.1 SDK is available on the LiveCycle ES DVD.
- The LiveCycle ES SDK folder. The LiveCycle ES SDK is available from the LiveCycle ES server or from the location Adobe LiveCycle Workbench ES is installed on your computer.
- Workspace ES to test your form. The process you use must be configured to use a Flex application as a form.

## Additional information

The resources in this table can help you learn more about LiveCycle ES.

For information about	See
A LiveCycle ES Overview	<a href="#">LiveCycle ES Overview</a>
The end-to-end process of creating a LiveCycle ES application that includes a form and human-centric processes, configuring services, and testing within Workspace ES	<a href="#">Creating Your First LiveCycle Application</a>
Adobe LiveCycle ES Workspace ES	<a href="#">LiveCycle Workspace ES Help</a>
ActionScript classes and properties included with LiveCycle ES and Flex	<a href="#">Adobe LiveCycle ES Update 1 ActionScript Reference</a>

For information about	See
Rendering and deploying Flex forms using processes created in Adobe® LiveCycle® Workbench ES	<a href="#">LiveCycle Workbench ES Help</a>
LiveCycle ES terminology	<a href="#">LiveCycle ES Glossary</a>
Installing Flex Builder	<a href="#">Flex Documentation</a>
Patch updates, technical notes, and additional information on this product version	<a href="http://www.adobe.com/go/learn_lc_support">http://www.adobe.com/go/learn_lc_support</a>

# 1

## Introduction

You can create Flex applications to provide data capture capabilities in the same manner as a form created in Adobe LiveCycle Designer ES or Adobe Acrobat®. Flex applications, like PDF or HTML forms, can be associated with automated processes to capture data. The reason that you use a Flex application instead of a regular form created in Designer ES or Acrobat is to provide an enriched user experience.

A Flex application provides different visualizations and integrations for tasks to be created that go beyond those offered by PDF forms, HTML forms, and form guides. For example, dashboards can be presented to users, audio and video can be incorporated for training or help purposes, and more advanced user interface components can be added to the Flex application. None of these things are possible from PDF or HTML forms.

For example, when a Flex application is displayed within LiveCycle Workspace ES, the Flex application responds to user actions when the Complete button is clicked to submit a form by sending data stored by the Flex application, which is called *form data*, back to Workspace ES. The following illustration shows a Flex application within Workspace ES. The Flex application is created to look much like a form, and users can interact with the Flex application and click buttons in Workspace ES to complete or save form data.

The screenshot displays the Adobe LiveCycle Workspace ES interface. At the top, the title bar reads 'ADOBE® LIVECYCLE® WORKSPACE ES' with navigation links for 'Preferences', 'Messages', 'Help', 'Logout', and a user greeting 'Welcome Rye Woodard'. Below the title bar, there are tabs for 'Start Process', 'To Do (0)', and 'Tracking'. On the left side, a sidebar shows 'Favorites' with items like 'Fin@nce Loan Applications' and 'Samples'. The main content area is titled 'Loan Application (Flex Form)' and contains a 'Mortgage Application Form' from 'Fin@nce Corp.'. The form includes a message: 'After you complete this form, one of our representatives will contact you within two business days.' Below this, there are input fields for 'Property Sale Price' (429900), 'Down Payment' (100000), 'Mortgage Amount' (329900), 'Last Name' (Woodard), 'First Name' (Rye), and 'Phone Number' (111-999-0000). A 'Complete' button with a green checkmark is located at the bottom right of the form area.

Flex application within Workspace ES

A Flex application that is enabled to communicate with Workspace ES functions as if it were part of Workspace ES, which is itself a Flex application. Flex applications must be configured with human-centric processes to appear in Workspace ES. (See [Creating Processes > Designing Human-Centric Processes in the LiveCycle Workbench ES Help](#).)

You configure the Flex application for use in a process using LiveCycle Workbench ES, which allows users to access the form within Workspace ES when they start or are involved in a process. If you have an existing Flex application, you can modify it for use within LiveCycle Workspace ES. This document describes how to create a new Flex application and how to modify an existing Flex application for use with Workspace ES.

## Creating a Flex application enabled for Workspace ES

Before you begin to create Flex applications enabled for Workspace ES, you need to understand the four basic tasks:

1. Setting up the development environment to develop Flex applications enabled for Workspace ES. (See [Configuring Your Development Environment](#).)
2. Configure the Flex project for creating Flex applications enabled for Workspace ES. (See [Configuring your Flex projects](#).)
3. Implementing the application logic for the Flex application. (See [Creating forms as Flex applications](#).)
4. Enabling the Flex application for Workspace ES. (See [Enabling Flex Applications for Workspace ES](#).)
5. Deploying and testing the Flex application. (See [Deploying and Testing Flex Applications in Workspace ES](#).)

## Enabling an existing Flex application for Workspace ES

Instead of creating a new Flex application, you can enable existing Flex applications by simply adding the required event listeners.

You can add code to an existing Flex application to validate form data and to notify Workspace ES when the validity of the form data has changed. Assuming that your Flex application has a data model and you have access to the Workspace ES API, you can enable a Flex application for use in a process by completing these tasks:

1. Setting up the development environment to develop Flex applications enabled for Workspace ES. (See [Configuring Your Development Environment](#).)
2. Configure the Flex project for creating Flex applications enabled for Workspace ES. (See [Configuring your Flex projects](#).)
3. Enabling the Flex application for Workspace ES. (See [Enabling Flex Applications for Workspace ES](#).)
4. Deploying and testing the Flex application. (See [Deploying and Testing Flex Applications in Workspace ES](#).)



## Sample files

The example code creates a Flex application that is similar to the Flex application used as part of the simple process found in the LiveCycle ES SDK. The sample process is located at *[install folder]/LiveCycle\_ES\_SDK/samples/LiveCycleES/SimpleMortgageLoan-Flex*, where *[install folder]* represents the location where you installed the LiveCycle ES server or Workbench ES. The sample code for the Flex application is located in the FlexBuilderProject.zip file.

You must set up your development environment to use the same Flex SDK version that is compatible with LiveCycle ES (version 3.0.1) and include the necessary SWC file to access the Workspace API. The tasks described in this section need to be performed on each computer that you use to create Flex applications enabled for LiveCycle Workspace ES.

To use the Workspace API, the Workspace ES SWC files must be configured in your build path in Flex Builder or included as part of the compile options when you use the command line. These SWC files are located in the LiveCycle ES SDK folder and let you use components, interfaces, and classes from the Workspace API. When using Flex Builder, you simply build your project; however, when you build your Flex application using the Flex SDK, you must specify the necessary SWC files as a compile option each time you build.

Before you can configure your development environment, you must verify that the following tasks were completed:

- The LiveCycle ES SDK folder is accessible and all the samples are installed. You can access this folder on the LiveCycle ES server or from a computer where Workbench ES is installed.
- You have either Flex Builder or your own development environment configured to work with Flex.

### Summary of steps

This document presumes that you are using Flex Builder to develop Flex applications enabled for Workspace ES. The following summary outlines the steps to take to configure your development environment with Flex Builder 3.0.x:

1. Install the Flex SDK that is provided with LiveCycle ES. (See [Installing the Flex SDK for LiveCycle ES](#)).
2. Configure Flex Builder to use the newly installed Flex SDK. (See [Configuring Flex Builder to use the Flex SDK for LiveCycle ES](#).)

## Installing the Flex SDK for LiveCycle ES

You must install the Flex SDK version that is compatible with LiveCycle ES. The Flex SDK version provided for LiveCycle ES includes localization SWC files and fixes that ensure that your Flex applications function within a LiveCycle ES environment, and provides proper localization support. The Flex SDK compatible with LiveCycle ES is available on the LiveCycle ES DVD.

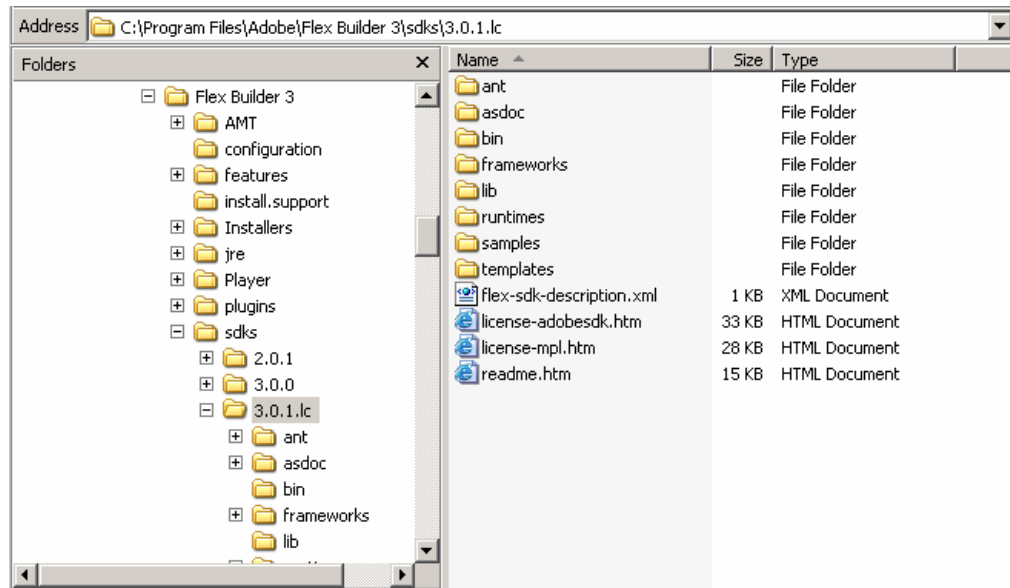
Before you perform these steps, you must have Flex Builder 3 installed in your development environment and access to the LiveCycle ES DVD.

To install the Flex SDK for LiveCycle in Flex Builder 3:

1. Exit Flex Builder if you have it running.
2. On the LiveCycle ES DVD, go to the `lifecycle_dataservices` folder and copy the `flex_sdk_3.zip` to your development computer.

3. Go to the `sdk`s folder located in the root folder of where you installed Flex Builder 3. For example, go to the `C:\Program Files\Adobe\Flex Builder 3\sdk`s folder.
4. Create a new folder, such as `3.0.1.lc`, for extracting the `flex_sdk_3.zip` file that you copied from the LiveCycle ES DVD.
5. Using an archiving tool, extract the `flex_sdk_3.zip` file that you copied from the LiveCycle ES DVD to the folder you created in step 4.

The folder you create should look like the following illustration after you install the Flex SDK from the LiveCycle ES DVD installed on your computer.



## Configuring Flex Builder to use the Flex SDK for LiveCycle ES

You must configure Flex Builder to compile your Flex applications for use with Workspace ES.

### ► To configure Flex Builder to use the Flex SDK for LiveCycle ES:

1. Start Flex Builder.
2. Select **Window > Preferences**.
3. In the Preferences dialog box, select **Flex > Installed Flex SDKs** and then click **Add**.
4. In the Add Flex SDK dialog box, perform the following steps and then click **OK**.
  - Click **Browse**, and in the Browse For Folder dialog box, go to and select the folder you created in step 4 in procedure [Installing the Flex SDK for LiveCycle ES](#), and then click **OK**.
  - In the **Flex SDK name** box, type a name to reflect the specific Flex SDK version used for applications you develop for LiveCycle ES, such as `Flex 3.0.1 - LCUpdate1`.

- [illegible]

- You have installed the Flex SDK version compatible with LiveCycle ES and made it the default Flex SDK version.

After you configure your development environment you must configure each Flex project for creating Flex applications enabled for LiveCycle Workspace ES. For each Flex project, you must configure your build path to include a SWC file that is located in the LiveCycle ES SDK folder. After you include the SWC file in your Flex project, you can use components, interfaces, and classes from the Workspace API.

When using Flex Builder, you simply build your project; however, when you build your Flex application using the Flex SDK, you must specify the necessary SWC files as a compile option each time you build.

It is recommended that you use Flex Builder. The following procedures presume that you use Flex Builder. Before you configure your Flex project, verify that you have performed the following steps:

Before you configure your Flex project, you must verify the following tasks were completed:

- You have configured your development environment to include the proper Flex SDK version compatible with LiveCycle ES. (See [Configuring Your Development Environment](#).)
- The LiveCycle ES SDK folder is accessible and all the samples are installed. You can access this folder on the LiveCycle ES server or from a computer where Workbench ES is installed.

### Summary of steps

This document presumes that you are using Flex Builder to develop Flex applications enabled for Workspace ES. The following summary outlines the steps to take to configure your development environment with Flex Builder 3.0.x:

1. Create a Flex project and include the Workspace ES API. (See [Including the Workspace API in Flex projects](#).)
2. Include the namespace to use the Workspace API in your Flex application. (See [Using the Workspace API in Flex applications](#).)

## Including the Workspace API in Flex projects

For each Flex application that you want to enable for Workspace ES, you must configure it to use the Workspace API. To use the Workspace API, you include the workspace-runtime.swc file in the build path for your Flex project, and also create a namespace in your MXML file to access classes, components, and interfaces from the Workspace API.

It is recommended that you do these steps for each project you create.

### ► To include the Workspace API in a Flex project:

1. In Flex Builder, select **File > New > Flex Project**.
2. In the Create a Flex project dialog box, in the **Project name** box, type a name, such as `LoanFlexForm`.
3. (Optional) If you want to select an alternative location on your computer to store the Flex project, deselect the **Use default location** check box and then click **Browse** beside the **Location** box.

4. Click **Next**.
5. (Optional) If you do not want to use the default bin-debug folder for compiled output, in the Output folder, click **Browse** and in the Select Output Folder dialog box, choose an existing location or create a new folder on your computer to store the output and then click **OK**.
6. Click **Next**.
7. Click **Finish** to create a Flex project with all the predefined defaults.

**Note:** You can create Flex applications that run only in a default Web application (runs in Adobe Flash® Player). AIR applications are not supported in Workspace ES.

8. Access the LiveCycle ES SDK folder at one of the following locations:

The LiveCycle ES SDK is available on the server when samples are installed. It is also available on the computer where Workbench ES is installed.

- To access the LiveCycle ES SDK when you have Workbench ES installed on your computer, browse to `[install_dir]/LiveCycle ES/Workbench ES/LiveCycle_ES_SDK` where `[install_dir]` represents where Workbench ES is installed on your computer.
- To access the LiveCycle ES SDK from the LiveCycle ES server, you must access the server and browse to `[install_dir]/LiveCycle_ES_SDK` where `[install_dir]` represents where LiveCycle ES is installed on a server.

For example, using a JBoss® turnkey installation, browse to the `C:\Adobe\LiveCycle8.2\LiveCycle_ES_SDK` folder on the server where LiveCycle ES is installed.

9. Browse to `[LC_SDK]/misc/Process_Management/Workspace` where `[LC_SDK]` is the location of the LiveCycle ES SDK folder, and perform one of the following tasks to use the LiveCycle Workspace ES API in your Flex project:
  - Drag the following workspace-runtime.swc file to the **libs** folder in the Flex project.
  - Copy the workspace-runtime.swc to the **libs** folder in the Flex project.

## Using the Workspace API in Flex applications

To use the Workspace API, you must create a namespace in the top-level component in your code. For creating a Flex application enabled in Workspace ES, this is typically the `mx:Application` component.

### ► To access the Workspace API in the project:

- In the MXML file that contains the top level component for your project, add a namespace, such as `lc`, to access the Workspace API by typing `xmlns:lc="http://www.adobe.com/2006/livecycle"` as an attribute.

**Note:** For the purposes of this document, the namespace will be `lc`.

The following MXML code creates a new namespace to access Workspace ES API classes:

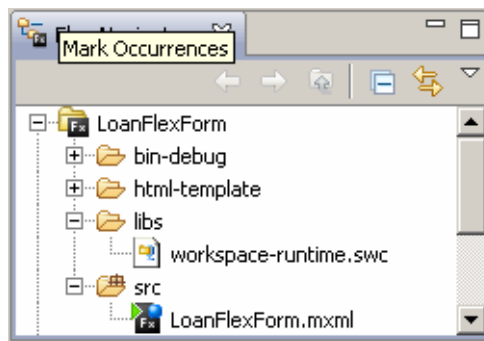
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:lc="http://www.adobe.com/2006/livecycle"
  layout="absolute">
</mx:Application>
```

- Select **Project > Build Project** to build your project. You should see output created output folder, such as the bin-debug folder, in your project.

**Note:** Earlier, you configured the Flex SDK for LiveCycle ES to be the default version in step 5 in procedure [Configuring Flex Builder to use the Flex SDK for LiveCycle ES](#). If you did not configure your Flex Builder to compile with the Flex SDK for LiveCycle ES, you must follow these steps:

- Select **Project > Properties**. In the Properties for *[flex project name]* dialog box, where *[flex project name]* is the name of the Flex project you created, select **Flex Compiler**.
- In the Flex SDK version pane, select **Use a specific SDK** and select the Flex SDK for LiveCycle ES that you installed, such as Flex 3.0.1-LCUpdate1, click **Apply**, and then click **OK**.
- A warning dialog box may appear because you are changing SDK versions after you click **Apply** in the previous step. Click **OK** to dismiss the warning.

After you complete the steps, your Flex project should look similar to the following illustration.

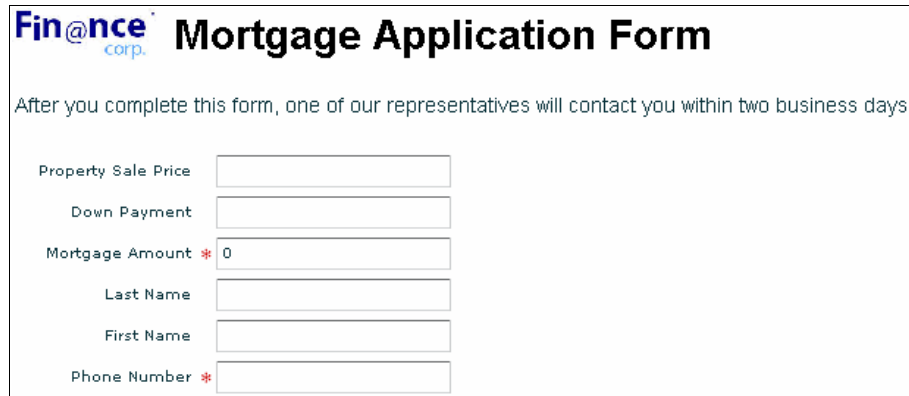


**Caution:** The SWC file will be automatically included each time you compile the project using Flex Builder. When you compile using the command line compiler, you must reference the workspace-runtime.swc file in the options. For more information about including SWCs as compile options, see the [Flex documentation](#).

## 4

# Creating Flex Applications for Data Capture

You can create a variety of sophisticated Flex applications as forms. After completing the tasks described in this section, you will have a Flex application that functions as a simple form that you can use for data capture purposes. For example, a simple Flex application may look like the following illustration.



The screenshot shows a web form titled "Mortgage Application Form" with the "Fin@nce corp." logo in the top left. Below the title is a line of text: "After you complete this form, one of our representatives will contact you within two business days." The form contains six input fields arranged vertically, each with a label to its left: "Property Sale Price", "Down Payment", "Mortgage Amount" (with a red asterisk and the value "0"), "Last Name", "First Name", and "Phone Number" (with a red asterisk). Each field is represented by a rectangular text box.

The example code that is displayed in this section creates a simple form in Flex that has these items:

- A Fin@nce Corp. logo in the upper-left corner of the form.

**Note:** You get the Fin@nce Corp. logo when you install the LiveCycle ES Samples , which is part of the LiveCycle ES SDK.

To access the LiveCycle ES Samples when LiveCycle Workbench ES is installed on your computer, browse to the `[installdir]/LiveCycle ES/Workbench ES/LiveCycle_ES_SDK/samples` folder where `[installdir]` represents where Workbench ES is installed on your computer.

To access the LiveCycle ES Samples from the LiveCycle ES server, you must access the server and browse to the `[installdir]/LiveCycle_ES_SDK/samples` folder where `[installdir]` represents where LiveCycle ES is installed on a server.

For example, using a JBoss turnkey installation, on the LiveCycle ES server, go to the `/Adobe/LiveCycle8.2/LiveCycle_ES_SDK/samples/LiveCycleES/MortgageLoan-Prebuilt/Images` folder and copy the `financeCorpLogo.gif` file to your Flex project.

- A Mortgage Application Form title.
- Instructions for filling out the form.
- Several fields arranged horizontally such as Property Sale Price, Down Payment, Mortgage Amount, Last Name, First Name, and Phone Number. The required fields are marked with a red asterisk to the left of the text box. Only the Mortgage Amount and Phone Number are marked as required.

You can also add a data model and bind it to the objects that are used to present and capture information from the user. A data model is necessary if you plan to pass form data from your Flex application to Workspace ES, which passes the form data to the appropriate automated business process.

In your Flex application, although optional, it is recommended that you validate the form data before you pass it to Workspace ES (and ultimately to an automated business process) for these reasons:

- If an error occurs in the data, the user can correct the fields immediately and then resubmit the form data.



- This extra step reduces the likelihood of an error during the execution of your business process.

## Summary of steps

The following summary outlines the steps to take to create a form as a Flex application:

1. Create a form in Flex by using the `mx:Form` and `mx:FormItem` components. (See [Creating forms as Flex applications](#).)
2. Add a data model to the Flex application. (See [Adding data models](#).)
3. (Optional) Add objects by using classes from the `mx:validator` package to verify the data in the Flex application. (See [Adding form validation](#).)

## Creating forms as Flex applications

You can create a form as a Flex application by using `mx:Form` and `mx:FormItem` components. You are not limited to using only the components described in this section. The components you choose will depend on the application logic you are implementing. For instance, you can add any of the following controls to your form.

**`mx:Image`:** For example, use this component to add a company logo to your form.

**`mx:RadioButton`:** For example, use this component to add a list of options that a user can choose from.

**`mx:Button`:** For example, add a button that performs a calculation based on the user's input.

For more information about creating Flex applications to behave as forms, see the [Flex 3 Developer Guide](#) and, using LiveDocs, go to Flex 3 Developer Guide > User Interfaces > Using Layout Containers > Form, FormHeading, and FormItem layout containers.

This section, describes the basic steps in building a form. Specifically, the steps describe how to use the `mx:Form` and `mx:FormItem` containers to build the layout of your form. In addition, as part of the example, the `mx:TextInput` component is used to provide an area for a user to type input into. For example, you may have a process where an applicant fills in basic information about a mortgage. The mortgage application, based on the amount of the loan, is routed to a specific person for review and approval. The example code in this section describes how to build a form with the following fields:

- Name of the applicant
- Phone number
- Property Sale Price
- Down payment
- Mortgage Amount

To create a Flex application as a form, perform the following steps:

1. If you have not already done so, add a new namespace to the Application component for accessing the Workspace ES API components. For example, add `xmlns:lc="http://www.adobe.com/2006/livecycle"`.
2. Add a `mx:VBox` container within the `mx:Application` container.

3. (Optional) Within the `mx:VBox` container from step 2, add other controls to the layout for images and headings. For example, you can add a corporate logo by using the `mx:Image` component or instructions for the form by adding a `mx:Label` control.

```
<!-- Create the Form -->
<mx:VBox backgroundColor="white">
  <mx:HBox>
    <mx:Image id="banner"
      source="@Embed('financeCorpLogo.jpg')"
      scaleContent="true" width="100" height="50"/>
    <mx:Label text="Mortgage Application Form" fontSize="28"
      color="black" fontWeight="bold" fontFamily="Arial"/>
  </mx:HBox>
  <mx:Label text="After you complete this form, one of our
    representatives will contact you within two
    business days."
    fontSize="14" fontThickness="10" fontFamily="Arial"/>
</mx:VBox>
```

4. Within the `mx:VBox` container, add an `mx:Form` container.
5. In the `mx:Form` container, configure properties such as width, height, backgroundColor, and foregroundColor for the form.
6. Within the `mx:Form` container, add as many `mx:FormItem` containers as required to represent each field that you want in your Flex application. For example, we have added fields to represent the name of the applicant, phone number, property sale price, down payment, and mortgage amount. Nest each `mx:FormItem` container within the `mx:Form` container. For each `mx:FormItem` container, perform the following tasks:
  - Set the `label` attribute to provide a name that will be displayed on the form.
  - Set the `required` attribute to specify whether the field is required. Required fields appear with a red asterisk beside the label.
  - Add an `mx:TextInput` control within each `mx:FormItem` container, and then add an `id` attribute and assign a unique name for each `mx:TextInput` control.

### Example: Creating a simple mortgage application that a user fills

**Note:** It is not possible to submit this form as written in Flex Builder. You can add additional controls, such as a button, for testing purposes.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:lc="http://www.adobe.com/2006/livecycle"
  layout="absolute">
  <!-- Create the Form -->
  <mx:VBox backgroundColor="white">
    <mx:HBox>
      <mx:Image id="banner"
        source="@Embed('financeCorpLogo.jpg')"
        scaleContent="true" width="100" height="50"/>
      <mx:Label text="Mortgage Application Form" fontSize="28"
        color="black" fontWeight="bold" fontFamily="Arial"/>
    </mx:HBox>
    <mx:Label text="After you complete this form, one of our
      representatives will contact you within two
      business days."
    >
  </mx:VBox>
</mx:Application>
```

```
                fontSize="14" fontThickness="10" fontFamily="Arial"/>
<mx:HBox>
    <mx:Form backgroundColor="white">
        <mx:FormItem label="Property Sale Price" required="false">
            <mx:TextInput id="propertySalePrice"/>
        </mx:FormItem>
        <mx:FormItem label="Down Payment" required="false">
            <mx:TextInput id="downPayment"/>
        </mx:FormItem>
        <mx:FormItem label="Mortgage Amount" required="true">
            <mx:TextInput id="mortgageAmount" />
        </mx:FormItem>
        <mx:FormItem label="Last Name" required="false">
            <mx:TextInput id="lastName" />
        </mx:FormItem>
        <mx:FormItem label="First Name" required="false">
            <mx:TextInput id="firstName" />
        </mx:FormItem>
        <mx:FormItem label="Phone Number" required="true">
            <mx:TextInput id="phoneNum" />
        </mx:FormItem>
    </mx:Form>
</mx:HBox>
</mx:VBox>
</mx:Application>
```

**Note:** The financeCorpLogo.jpg image used in this example is located in the LiveCycle ES SDK in the `[LC_SDK]/samples/LiveCycleES/MortgageLoan-Prebuilt/Images` folder where `[LC_SDK]` is the location of the LiveCycle ES SDK.

To access the LiveCycle ES SDK when Workbench ES is installed on your computer, browse to `[installdir]/LiveCycle ES/Workbench ES/LiveCycle_ES_SDK` where `[installdir]` represents where Workbench ES is installed on your computer.

To access the LiveCycle ES SDK from the LiveCycle ES server, you must access the server and browse to `[installdir]/LiveCycle_ES_SDK` where `[installdir]` represents where LiveCycle ES is installed on a server.

For example, using a JBoss turnkey installation, on the LiveCycle ES server, go to the `C:\Adobe\LiveCycle8.2\LiveCycle_ES_SDK` folder.

## Adding data models

In your Flex application, you can add and configure a data model to pass data based on a schema. *Form data*, which is data from a form or your Flex application, is passed as XML to be used by an automated business process. For more information about adding a data model, see the [Flex 3 Developer Guide](#) and, using LiveDocs, go to [Flex 3 Developer Guide > Data Access and Interconnectivity > Storing Data > Defining a data model](#).

It is recommended that you use the `mx:XML` component to define the model. Using the `mx:XML` component permits you to define the model and write the code to populate that model when changes occur on your form or when there is default data provided for your form.

To add a data model to a Flex application, perform the following steps:

1. Add an `mx:XML` component:
  - Add an `mx:XML` component and define a unique name for it.
  - Define the XML schema that you want to use by using the `<data>` tags in the `mx:XML` component element tags.
2. Map form data to the data model defined in the `mx:XML` component:
  - For each element within the `mx:FormItem` container, bind the text properties to the corresponding XML element. For example, for an `mx:TextInput` control, perform these tasks:
    - Set the `text` attribute to a value that represents the XML element defined in the `mx:XML` component.
    - Set the `change` attribute to update the data in the data model each time the field is updated.

**Example: Creating a two-way binding between the defined data model and each field in the form**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:lc="http://www.adobe.com/2006/livecycle"
                 layout="absolute">
    ...
    ...
    <!-- Define the model -->
    <mx:XML id="mydata">
        <data>
            <mortgageInfo>
                <propertyPrice> </propertyPrice>
                <downPayment> </downPayment>
                <mortgageAmount> </mortgageAmount>
            </mortgageInfo>
            <contactInfo>
                <lastName> </lastName>
                <firstName> </firstName>
                <phoneNum> </phoneNum>
            </contactInfo>
        </data>
    </mx:XML>

    <!-- Create the Form -->
    <mx:VBox backgroundColor="white">
        <mx:HBox>
            <mx:Image id="banner"
                      source="@Embed('financeCorpLogo.jpg')"
                      scaleContent="true" width="100" height="50"/>
            <mx:Label text="Mortgage Application Form" fontSize="28"
                      color="black" fontWeight="bold" fontFamily="Arial"/>
        </mx:HBox>
        <mx:Label text="After you complete this form, one of our
                        representatives will contact you within two
                        business days."
                  fontSize="14" fontThickness="10" fontFamily="Arial"/>
        <mx:HBox>
            <mx:Form backgroundColor="white">
                <mx:FormItem label="Property Sale Price" required="false">
                    <mx:TextInput id="propertySalePrice"
```

```
                text="{mydata.mortgageInfo.propertyPrice}"
                change="mydata.mortgageInfo.propertyPrice =
                    propertySalePrice.text;
                    mydata.mortgageInfo.mortgageAmount =
                        mydata.mortgageInfo.propertyPrice -
                        mydata.mortgageInfo.downPayment;
                    lcConnector.setDirty();"/>
            </mx:FormItem>
            <mx:FormItem label="Down Payment" required="false">
                <mx:TextInput id="downPayment"
                    text="{mydata.mortgageInfo.downPayment}"
                    change="mydata.mortgageInfo.downPayment =
                        downPayment.text;
                        mydata.mortgageInfo.mortgageAmount =
                            mydata.mortgageInfo.propertyPrice -
                            mydata.mortgageInfo.downPayment;"/>
            </mx:FormItem>
            <mx:FormItem label="Mortgage Amount" required="true">
                <mx:TextInput id="mortgageAmount"
                    text="{mydata.mortgageInfo.propertyPrice -
                        mydata.mortgageInfo.downPayment}"
                    change="mydata.mortgageInfo.mortgageAmount =
                        mortgageAmount.text;"/>
            </mx:FormItem>
            <mx:FormItem label="Last Name" required="false">
                <mx:TextInput id="lastName"
                    text="{mydata.contactInfo.lastName}"
                    change="mydata.contactInfo.lastName =
                        lastName.text;"/>
            </mx:FormItem>
            <mx:FormItem label="First Name" required="false">
                <mx:TextInput id="firstName"
                    text="{mydata.contactInfo.firstName}"
                    change="mydata.contactInfo.firstName =
                        firstName.text;"/>
            </mx:FormItem>
            <mx:FormItem label="Phone Number" required="true">
                <mx:TextInput id="phoneNum"
                    text="{mydata.contactInfo.phoneNum}"
                    change="mydata.contactInfo.phoneNum =
                        phoneNum.text;"/>
            </mx:FormItem>
        </mx:Form>
    </mx:HBox>
</mx:VBox>
</mx:Application>
```

## Adding form validation

You can add form validation to the form to ensure that the information provided by the user is correct.

You add an instance of a class from the `mx:validator` package to validate data in the Flex application. If more than one validation is being performed on a value, you must ensure that form data is marked as valid. Conversely, if at least one validation on a value fails, you must ensure that the form data is marked as invalid.

For more information about handling form validation, see the [Flex 3 Developer Guide](#) and, using LiveDocs, go to Flex 3 Developer Guide > Data Access and Interconnectivity > Validating Data.

To add validation to a form, perform the following tasks:

1. Identify the field that you want to verify.

For example, you can validate a string value by using the `mx:StringValidator` component, validate a numeric value by using the `mx:NumberValidator` component, or verify whether a phone number exists by using the `mx:PhoneNumberValidator` component. Validation classes are in the `mx.validators` package and, depending on which component you choose to use, different properties are used to perform the validation of data.

2. Add the `source` property to the `mx.validator` class you chose and bind it to the `id` of the `mx:TextInput` component that represents the field you want to validate in the form.
3. Add the conditions and error messages that you want to use for the `mx.validator` class. For example, you can use the `minValue` attribute or the `mx:NumberValidator` component to specify the smallest value the field can contain.

**Example: Creating an object from the `mx:validator` package to verify that the mortgage amount is greater than zero and that a phone number exists.**

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:lc="http://www.adobe.com/2006/livecycle"
                 layout="absolute">

    <!-- Define the model -->
    <mx:XML id="mydata">
        <data>
            ...
            ...
        </data>
    </mx:XML>

    <!-- Create the Form -->
    <mx:VBox backgroundColor="white">
        <mx:HBox>
            <mx:Image id="banner"
                      source="@Embed('financeCorpLogo.jpg')"
                      scaleContent="true" width="100" height="50"/>
            <mx:Label text="Mortgage Application Form" fontSize="28"
                      color="black" fontWeight="bold" fontFamily="Arial"/>
        </mx:HBox>
        <mx:Label text="After you complete this form, one of our
                        representatives will contact you within two
                        business days."
    </mx:Label>
    </mx:VBox>
</mx:Application>
```

```
                fontSize="14" fontThickness="10" fontFamily="Arial"/>
<mx:HBox>
    <mx:Form backgroundColor="white">
        <mx:FormItem label="Property Sale Price" required="false">
            <mx:TextInput id="propertySalePrice"
                text="{mydata.mortgageInfo.propertyPrice}"
                change="mydata.mortgageInfo.propertyPrice =
                    propertySalePrice.text;"/>
        </mx:FormItem>
        <mx:FormItem label="Down Payment" required="false">
            <mx:TextInput id="downPayment"
                text="{mydata.mortgageInfo.downPayment}"
                change="mydata.mortgageInfo.downPayment =
                    downPayment.text;"/>
        </mx:FormItem>
        <mx:FormItem label="Mortgage Amount" required="true">
            <mx:TextInput id="mortgageAmount"
                text="{mydata.mortgageInfo.propertyPrice -
                    mydata.mortgageInfo.downPayment}"
                change="mydata.mortgageInfo.mortgageAmount =
                    mortgageAmount.text;"/>
        </mx:FormItem>
        ...
        ...
        <mx:FormItem label="Phone Number" required="true">
            <mx:TextInput id="phoneNum"
                text="{mydata.contactInfo.phoneNum}"/>
        </mx:FormItem>
    </mx:Form>
</mx:HBox>
</mx:VBox>

<!-- The validator for the fields that store the mortgage amount
and phone number. -->
<mx:NumberValidator id="mortgageValidator"
    source="{mortgageAmount}"
    property="text"
    required="true"
    minValue="0"
    lowerThanMinError="Mortgage must be greater than 0."
    domain="real"
    listener="mortgageAmount"/>

<mx:PhoneNumberValidator id="phoneNumValidator"
    source="{phoneNum}"
    property="text"
    required="true"
    requiredFieldError="You must provide a phone number."/>
</mx:Application>
```

When a Flex application is enabled for LiveCycle Workspace ES, the Flex application responds to actions in Workspace ES like a PDF or HTML form. Before you start this section, you should have a Flex application written as a form. (See [Creating Flex Applications for Data Capture](#).) The Flex application you are enabling for Workspace ES must have a data model associated with it, and you can optionally validate any data that it stores.

To enable a Flex application for Workspace ES, you use Flex event handling to facilitate the communication between the Flex application you create and Workspace ES. This communication is necessary for exchanging form data and responding to button clicks that occur within Workspace ES for actions such as saving the form data or submitting form data. In addition to dispatching events for handling when form data is to be saved or submitted, the Flex application dispatches events to notify Workspace ES whether form data is valid or whether form data has changed.

You may want to write the code to provide the communication between the Flex application and Workspace ES by using Flex event-handling APIs; however, it is recommended that you use the `lc:FormConnector` component, which is available from the Workspace ES API, to enable a Flex application for Workspace ES.

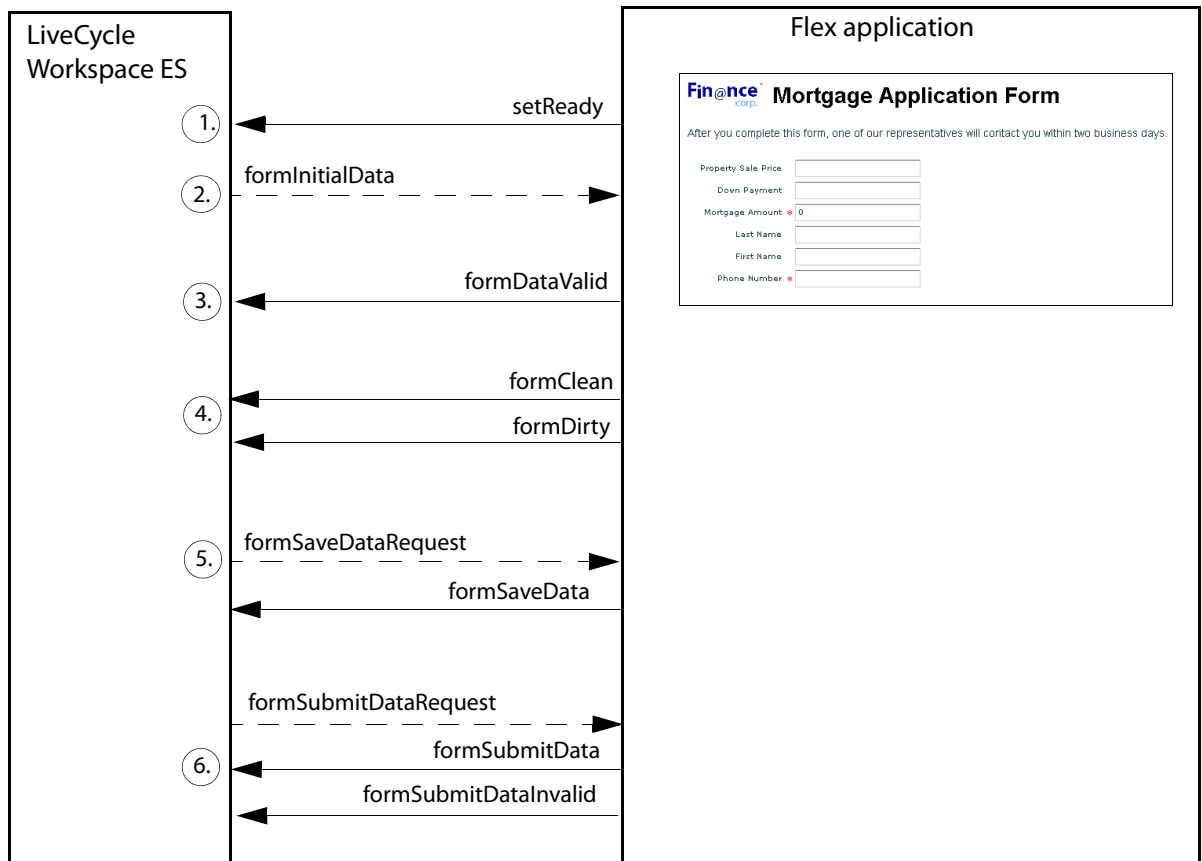
If you choose to use the Flex event-handling API to write the code to communicate with Workspace ES, the event type string constants that you support are in the `FormEvents` class, which is in the `lc.core.Events` package. (See [LiveCycle ES Update 1 ActionScript Language](#)). The `flash.events.DataEvent` type is used for events that carry data, and the `flash.events.Event` type is used for all other events.

**Note:** Simple types from the `flash.events` package are used instead of custom Flex types to support Flex applications in separate Flash security domains. For example, the data is transferred as a `String` in a `DataEvent` object.

You enable a Flex application for Workspace ES by providing event listeners to respond to events from Workspace ES and dispatching the appropriate events. Knowledge of the events that are sent between the Flex application and Workspace ES is important for understanding how to build Flex applications that are enabled for Workspace ES. The following illustration shows the events that are sent between a Flex application and Workspace ES.



**Note:** A solid arrow indicates an event from the Flex application to Workspace ES and a dashed arrow indicates an event from Workspace ES to the Flex application.



The following list describes the events that can occur between a Flex application and Workspace ES:

**Note:** Each item in the following list corresponds to a number in the diagram.

1. After the Flex application is finished loading, it dispatches a `setReady` event to Workspace ES. The `setReady` event notifies Workspace ES that the Flex application is ready to start receiving events. This is the first step that must occur before Workspace ES can dispatch any events.
2. Workspace ES dispatches a `formInitialData` event with data to the Flex application, which is used to populate data values in the Flex application. If no data exists, the `DataEvent.data` value is `null` or an empty space.

The data that is loaded into a Flex application is retrieved during the execution of a business process or specified during the design of a business process within LiveCycle Workbench ES when starting a process. For information about developing processes, see [LiveCycle Workbench ES Help](#) and go to Creating Processes > Designing Human-Centric Processes.

3. At minimum, the Flex application must dispatch the `formIsValid` event to enable the Complete button or route buttons in Workspace ES.

Optionally, both `formIsValid` and `formIsInvalid` events can be sent to indicate the validity of the data that the Flex application stores. Instead of dispatching separate `formIsValid` and `formIsInvalid` events as the Flex application runs, you can optimize the performance of your Flex application by dispatching a single `formIsValid` event in the event listener for the `formInitialData` event. In this situation, your Flex application must validate the data when a

`formSubmitDataRequest` event is received from Workspace ES, and then dispatch a `formSubmitdata` or `formSubmitDataInvalid` event back to Workspace ES.

4. In Workspace ES, the user is always prompted by default to save a draft form when they leave a task because the form is assumed to be in a dirty status. A *dirty* status means that the form needs to be saved, and a *clean* status means that the form does not need to be saved. The Flex application can track the status of the data by dispatching a `formClean` or `formDirty` event to Workspace ES.

When the `formClean` and `formDirty` events are used, Workspace ES prompts a user to save a draft form only when the data in the Flex application changes (`formDirty`). It is recommended that the Flex application dispatch a `formClean` event after the `formInitialData` and `formSaveData` events to set the form as clean.

5. Workspace ES dispatches a `formSaveDataRequest` event when the user chooses to save draft (incomplete) data. In response to a `formSaveDataRequest` event, the Flex application must dispatch a `formSaveData` event with the form data specified in XML format.
6. Workspace ES dispatches a `formSubmitDataRequest` event when the user chooses to submit the form data. In response to the `formSubmitDataRequest` event, the Flex application must dispatch either a `formSubmitData` event with the valid form data or a `formSubmitDataInvalid` event.

After Workspace ES receives a `formSubmitData` event in response to the `formSubmitDataRequest` event, Workspace ES uploads the Flex application.

## Summary of steps

The following summary outlines the steps to take to enable a Flex application for Workspace ES:

1. Add the `lc:FormConnector` component. (See [Adding the FormConnector component](#).)
2. (Optional) Dispatch events to indicate whether data is valid. (See [Validating data stored in Flex applications](#).)
3. (Optional) Dispatch events to indicate that data has changed. (See [Indicating that form data stored in Flex applications have changed](#).)
4. Create event listeners to handle events dispatched by Workspace ES. (See [Creating events listeners to handle events from Workspace ES](#).)

## Adding the FormConnector component

Before you use the `lc:FormConnector` component, you must configure your development environment (see [Configuring Your Development Environment](#)) and configure your Flex project to use the Workspace API (see [Configuring your Flex projects](#)).

By using the `lc:FormConnector` component, which is available from the Workspace API, you can use events and methods that ease the communication between your Flex application and Workspace ES. After the Flex application finishes loading and is ready to communicate with Workspace ES, the `setReady` event must be dispatched to notify Workspace ES that the Flex application is ready to start receiving events. The `setReady` event must be the first event dispatched to Workspace ES by your Flex application.

To use the `lc:FormConnector` component to facilitate communication with Workspace ES, perform the following steps:

1. If you have not already done so, add a new namespace to the `mx:Application` component for accessing the Workspace ES API components. For example, add `xmlns:lc="http://www.adobe.com/2006/livecycle"`.
2. Add an instance of the `lc:FormConnector` component and assign a name to the `id` attribute.
3. Add a `creationComplete` attribute to the `mx:Application` component and bind using the `setReady` method from the `lc:FormConnector` component you created in step 2.

**Example: Creating a `FormConnector` object and dispatching the `setReady` event when the Flex application finishes loading**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:lc="http://www.adobe.com/2006/livecycle"
                 layout="absolute"
                 creationComplete="lcConnector.setReady()" >
...
...

<!-- Add a FormConnector object -->
  <lc:FormConnector id="lcConnector" />

...
...

</mx:Application>
```

## Validating data stored in Flex applications

If your Flex application validates form data, you can use the `lc:FormConnector` component to dispatch events to indicate to Workspace ES users whether the data is valid. Validator classes from the `mx.validators` package are used to verify the validity of fields values.

These events can be dispatched to indicate the validity of form data in your Flex application:

**formDataValid:** Notifies Workspace ES that the form data is valid. This event must be dispatched to Workspace ES for the Complete button to be available.

**formDataInvalid:** Notifies Workspace ES that the form data is not valid.

For Workspace ES to enable the Complete button or route buttons so that a user can submit a task, the `formDataValid` event must be received from your Flex application. If the application logic in your Flex application validates more than one field, you must ensure that only one event is dispatched to represent the validity of all the data that is stored.

**Tip:** If you have multiple pieces of form data to verify, it is a good practice to verify the validity of the data in the event listener that handles the event for submit form data requests. The data should be properly formatted and valid before it is sent to a Workspace ES.

To bind data validity verification to the `lc:FormConnector` object, perform the following steps:

1. Add the `valid` attribute to the instances of `mx:Validator` classes that you are using, and use the `setDataValid` method from the `lc:FormConnector` component to dispatch the `formDataValid` event to Workspace ES.

2. Add the `invalid` attribute to the Validator object you are using and use the `setDataInvalid` method from the `lc:FormConnector` component to dispatch the `formDataInvalid` event to Workspace ES.

**Example: Dispatching a `formDataValid` event when the field indicated by the source attribute is valid; otherwise, a `formDataInvalid` event is dispatched**

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:lc="http://www.adobe.com/2006/livecycle"
                 layout="absolute">

    ...
    ...
    <!-- Add a Form Connector object -->
    <lc:FormConnector id="lcConnector" />

    ...
    ...

    <!-- Validate fields in the form using classes from the -->
    <!-- mx.validator package -->
    <mx:NumberValidator id="mortgageValidator"
                       source="{mortgageAmount}"
                       required="true"
                       minValue="0"
                       lowerThanMinError="Mortgage must be greater than 0."
                       property="text"
                       domain="real"
                       listener="mortgageAmount"
                       valid="lcConnector.setDataValid()"
                       invalid="lcConnector.setDataInvalid()" />

    <mx:PhoneNumberValidator id="phoneNumValidator"
                            source="{phoneNum}"
                            requiredFieldError="You must provide a phone number."
                            valid="lcConnector.setDataValid();"
                            invalid="lcConnector.setDataInvalid();"
                            property="text" />

</mx:Application>
```

## Indicating that form data stored in Flex applications have changed

In Workspace ES, a user who leaves a form associated with a task is always prompted by default to save the form data regardless of whether changes were made. To change this behavior so that the user is prompted to save the form only when changes occur to the form data, the Flex application must dispatch an event to indicate whether the form data that it stored changed.

The `lc:FormConnector` component provides methods to dispatch events that can be used to indicate whether the form data stored in the Flex application changed. If you choose to use events to indicate when to save changes to the form data, your Flex application must dispatch the `formClean` event

immediately after your Flex application initializes and then dispatch the `formDirty` event when changes occur to the form data in your Flex application.

Here is a summary of the events you can dispatch for indicating that form data has changed:

**formDirty:** Notifies Workspace ES that the form data has changed.

**formClean:** Notifies Workspace ES that the form data has not changed.

**Note:** If the `formDirty` or `formClean` events are not used, a message is always displayed to the user indicating to save the form regardless of whether changes occurred in the form data that is stored by the Flex application.

To indicate that form data has changed in the Flex application, perform the following steps:

- In your Flex component, locate each component or control where data is entered or changed. For most components, when a field changes its value, a change event is available to bind to and add the attribute that specifies a change event.

Bind the change attribute to the `setDirty()` method from the `lc:FormConnector` component.

For example, for the `mx:TextInput` control, for the `change` attribute, bind the component to the `setDirty()` method from the `lc:FormConnector` component.

**Example: Dispatching the `formDirty` event, which is bound to the change event for each instance of a `mx:TextInput` control**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:lc="http://www.adobe.com/2006/livecycle"
  layout="absolute"
  creationComplete="lcConnector.setReady()" >

  ...

  ...

  <mx:VBox backgroundColor="white">
    <mx:HBox>
      <mx:Image id="banner"
        source="@Embed('financeCorpLogo.jpg')"
        scaleContent="true" width="100" height="50"/>
      <mx:Label text="Mortgage Application Form" fontSize="28"
        color="black" fontWeight="bold" fontFamily="Arial"/>
    </mx:HBox>
    <mx:Label text=  "After you complete this form, one of our
      representatives will contact you within two
      business days."
      fontSize="14" fontThickness="10"
      fontFamily="Arial"/>
    <mx:HBox>
      <mx:Form backgroundColor="white">
        <mx:FormItem label="Property Sale Price" required="false">
          <mx:TextInput id="propertySalePrice"
            text="{mydata.mortgageInfo.propertyPrice}"
            change="mydata.mortgageInfo.propertyPrice =
              propertySalePrice.text;
              mydata.mortgageInfo.mortgageAmount =
                mydata.mortgageInfo.propertyPrice -
```

```

        mydata.mortgageInfo.downPayment;
        lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="Down Payment" required="false">
        <mx:TextInput id="downPayment"
            text="{mydata.mortgageInfo.downPayment}"
            change="mydata.mortgageInfo.downPayment =
                downPayment.text;
                mydata.mortgageInfo.mortgageAmount =
                    mydata.mortgageInfo.propertyPrice -
                    mydata.mortgageInfo.downPayment;
                lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="Mortgage Amount" required="true">
        <mx:TextInput id="mortgageAmount"
            text="{mydata.mortgageInfo.propertyPrice -
                mydata.mortgageInfo.downPayment}"
            change="mydata.mortgageInfo.mortgageAmount =
                mortgageAmount.text;
                lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="Last Name" required="false">
        <mx:TextInput id="lastName"
            text="{mydata.contactInfo.lastName}"
            change="mydata.contactInfo.lastName = lastName.text;
                lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="First Name" required="false">
        <mx:TextInput id="firstName"
            text="{mydata.contactInfo.firstName}"
            change="mydata.contactInfo.firstName =
                firstName.text;
                lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="Phone Number" required="true">
        <mx:TextInput id="phoneNum"
            text="{mydata.contactInfo.phoneNum}"
            change="mydata.contactInfo.phoneNum =
                phoneNum.text;
                lcConnector.setDirty();" />
    </mx:FormItem>
    </mx:Form>
    </mx:HBox>
</mx:VBox>

...
...

</mx:Application>

```

## Creating events listeners to handle events from Workspace ES

A significant part of enabling a Flex application for Workspace ES is to create the event listeners that respond to initial form data events and events that are triggered when the user decides to save or submit a form. Communication between the Flex application that you create and Workspace ES is facilitated by using events and event listeners.

The following events are dispatched by Workspace ES:

**formInitialData:** Sends initialization data that can be used to pre-populate field values before the Flex application is displayed in Workspace ES. The initialization data is sent to the Flex application as soon as Workspace ES receives the `setReady` event. The initialization data is specified during the design of an automated process created within Workbench ES. The XML data should be formatted based on the same schema used by the Flex application, such as the schema defined by the `mx:XML` component.

When Workspace ES first loads a Flex application, if initial data exists, it can populate your Flex application. The initial data that appears in your Flex application when you start a process is configured in the automated business process using Workbench ES. For more information about configuring initialization data, see [LiveCycle Workbench ES Help](#) and go to Creating Processes > Designing Human-Centric Processes > Prepopulating forms with default data.

**formSaveDataRequest:** Specifies that the user clicked the Save As Draft button in Workspace ES.

The event listener you create must respond by dispatching the `formSaveData` event with the form data formatted as XML. There is no requirement that the data is valid because it can represent in-progress form data.

**formSubmitDataRequest:** Specifies that the user clicked the Complete button in Workspace ES.

The event listener you create must respond by dispatching a `formSubmitData` event with the form data formatted as XML. It is recommended that the data is validated before dispatching the `formSubmitData` event. If you are performing validation of multiple pieces of data, it is recommended that a verification is performed in the event listener for the `formSubmitDataRequest` event. You would dispatch the `formDataValid` event before dispatching the `formSubmitData` event. If the data does not validate, you should not dispatch the `formSubmitData` event.

The following events can be dispatched by your Flex application to Workspace ES:

**formDataValid:** Notifies Workspace ES that the form data is valid. This event must be dispatched to Workspace ES for the Complete button to be available.

**formDataInvalid:** Notifies Workspace ES that the form data is not valid.

**formSaveData:** Passes the form data stored by a Flex application to Workspace ES.

**formSubmitData:** Indicates to Workspace ES that the data to submit is valid.

**formSubmitDataInvalid:** Indicates to Workspace ES that the data to submit is not valid. The Flex application must dispatch this event when validation of data fails.

You must create an event listener to handle each of the events that are dispatched by Workspace ES.

To add an event listener to handle the `formInitialData` event, perform the following tasks:

1. Add an `mx:Script` component if one does not exist.
2. Add a function with one parameter of type `DataEvent` that returns void to handle the `formInitialData` event within the `mx:Script` object.

3. If the initial data is passed from the `DataEvent` object, extract the data as XML and populate the fields in the Flex application. If you have an instance of the `mx:XML` component, the values can be populated; otherwise, set an empty XML value by using the root node of your data model.
4. (Optional) Dispatch the `formClean` event by using the `setClean()` method to indicate that the form no longer needs to be saved in an initialization function using the `lc:FormConnector` component. This action ensures that when the user first loads the form, the form does not need to be saved if no changes were made to it.
5. In the `lc:FormConnector` object, add the `formInitialData` attribute and set it to the name of the function you created in step 2.

### Example: Handling the `formInitialData` event from Workspace ES

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:lc="http://www.adobe.com/2006/livecycle"
                 layout="absolute"
                 creationComplete="lcConnector.setReady()">

  <mx:Script>
    <![CDATA[
      import mx.events.ValidationResultEvent;
      /* The formInitDataHandler event handler is dispatched by Workspace ES
       * and loads any initial data into the form. Initial data is null when
       * there is no data to be displayed in the form.
       */
      private function formInitDataHandler(event:DataEvent):void
      {
        if (event.data != null && event.data != "")
        {
          mydata = new XML(event.data);
        }
        else
        {
          //Empty data.
          mydata = new XML("<data/>");
        }
        lcConnector.setClean();

        var validationMortgageAmount:ValidationResultEvent =
          mortgageValidator.validate(null,true);
        var validationPhoneNum:ValidationResultEvent =
          phoneNumValidator.validate(null, true);
        if (( validationMortgageAmount.type == ValidationResultEvent.INVALID)
            || (validationPhoneNum.type == ValidationResultEvent.INVALID ))
        {
          lcConnector.setDataInvalid();
        }
      }
    ]]>
  </mx:Script>

  <lc:FormConnector id="lcConnector"
                   formInitialData="formInitDataHandler(event)"/>
```



```
...  
...  
  
</mx:Application>
```

To add an event listener to handle the `formSaveDataRequest` event, perform the following tasks:

1. Add an `mx:Script` component if one does not exist.
2. Add a function with one parameter of type `Event` that returns `void` to handle the `formSaveDataRequest` event within the `mx:Script` object. For the function, perform the following tasks:
  - Dispatch the `formSaveData` event by calling the `setSaveData` method from the `lc:FormConnector` component, sending the data from the `mx:XML` object that you created as part of this step, and send the XML data back to Workspace ES.
  - Dispatch the `formClean` event by calling the `setClean()` method from the `lc:FormConnector` component.
3. In the `lc:FormConnector` component, add the `formSaveDataRequest` attribute and set it to the name of the function you created in step 2.

**Example: Handling the `formSaveDataRequest` event and dispatching the `formSaveData` event with the form data to save**

```
<?xml version="1.0" encoding="utf-8"?>  
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  
    xmlns:lc="http://www.adobe.com/2006/livecycle"  
    layout="absolute"  
    creationComplete="lcConnector.setReady()">  
    <mx:Script>  
    <![CDATA[  
        ...  
        ...  
  
        import mx.events.ValidationResultEvent;  
  
        /* The formSaveDataRequestHandler event handler executes when the user  
        * clicks the Save As Draft button within Workspace ES. There is  
        * no validation performed to allow the data to save in-progress data  
        * even though the data is invalid.  
        */  
        private function formSaveDataListener (event:Event):void  
        {  
            lcConnector.setSaveData(mydata);  
            lcConnector.setClean();  
        }  
  
        ...  
        ...  
  
    ]]>  
</mx:Script>  
<!-- Add a FormConnector object -->  
<lc:FormConnector id="lcConnector"
```

```
        formInitialData="formInitDataHandler(event)";  
        formSaveDataRequest="formSaveDataListener(event)"/>  
        ...  
        ...  
  
</mx:Application>
```

To add an event listener to handle the `formSubmitDataRequest` event, perform the following tasks:

1. Add a function with one parameter of type `Event` that returns `void` to handle the `formSubmitDataRequest` event within the `mx:Script` object.
2. If your Flex application validates the data it stores, perform one of the following tasks after determining whether the data is valid:
  - If the form data is valid, store the form data as an XML object from the `mx:XML` component that defines and stores your data for the application. Then dispatch a `formSubmitDataValid` event by using the `setSubmitData` method from the `lc:FormConnector` component. Include the form data formatted as XML when using the `setSubmitData` method.
  - If the data is not valid, dispatch a `formSubmitDataInvalid` event by calling the `setSubmitDataInvalid` method from the `lc:FormConnector` component.
3. In the `lc:FormConnector` component, add the `formSubmitDataRequest` attribute and set it to the name of your event listener that handles submission requests from Workspace ES.

**Example: Handling the `formSubmitDataRequest` event and dispatching the `formSubmitDataValid` event with the form data to save**

```
<?xml version="1.0" encoding="utf-8"?>  
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  
                xmlns:lc="http://www.adobe.com/2006/livecycle"  
                layout="absolute"  
                creationComplete="lcConnector.setReady()">  
    ...  
    ...  
    <mx:Script>  
    <![CDATA[  
  
        import mx.events.ValidationResultEvent;  
        ...  
        ...  
  
        /* The formSubmitDataRequestListener responds to the  
        * formSubmitDataRequest event. The formSubmitDataRequest event  
        * is triggered when the a user clicks the Complete button  
        * or one of the task route buttons within Workspace ES.  
        * A setSubmitData event is dispatched if the data contains no  
        * errors; otherwise, the setSubmitDataInvalid event  
        * should be dispatched.  
        */  
        private function formSubmitDataListener(event:Event):void  
        {  
            var validationMortgageAmount:ValidationResultEvent =  
                mortgageValidator.validate(null,true);
```

```
var validationPhoneNum:ValidationResultEvent =
    phoneNumValidator.validate(null, true);

// Verify that all the necessary data is valid before
// dispatching the formSubmitData event with the
// setSubmitData (xmldata) method.
if (( validationMortgageAmount.type == ValidationResultEvent.VALID) &&
    (validationPhoneNum.type == ValidationResultEvent.VALID ))
{
    lcConnector.setSubmitData(mydata);
}
//Data was not valid, dispatch the formSubmitDataInvalid
//event using the setSubmitDataInvalid method.
else
{
    lcConnector.setSubmitDataInvalid();
}
}
]]>
</mx:Script>
<!-- Add a Form Connector object -->
    <lc:FormConnector id="lcConnector"
        formInitialData="formInitDataHandler(event);"
        formSaveDataRequest="formSaveDataListener(event)"
        formSubmitDataRequest="formSubmitDataListener(event)"/>
</mx:Script>

...
...

</mx:Application>
```

## Deploying and Testing Flex Applications in Workspace ES

---

After you finish creating and enabling your Flex application for LiveCycle Workspace ES, you can compile, deploy, and test it. Although you can compile your Flex application and display the SWF file in Flash Player or Flex Builder, you cannot test your Flex application with LiveCycle Workbench ES until you configure it in a process.

**Caution:** You can test Workspace ES-enabled Flex applications only within Workspace ES because the communication that occurs between the Flex application that you create and Workspace ES cannot be tested in a simple web browser.

Testing requires you to configure your Flex application in a human-centric process after you compile it as a SWF file. The SWF file must be copied to the repository in LiveCycle ES. After you create the process, you must configure the process to be accessible from Workspace ES. For information about deploying your Flex application, see [Deploying Flex applications](#).

After you configure your Flex application in a human-centric process, you can test it in Workspace ES. (See [Testing Flex applications](#).)

During your testing, you may encounter problems. You may also encounter problems during the development of your Flex application and enabling it for Workspace ES. For a list of common problems and suggested resolutions, see [Troubleshooting](#).

### Deploying Flex applications

After you create and enable your Flex application for Workspace ES, you need to deploy your Flex application to LiveCycle ES. Before you can test your application, you must configure it in a human-centric process.

► **To copy your Flex application to LiveCycle ES:**

1. Save your Flex project and compile the Flex application as a SWF file.

**Note:** If you are using the command-line Flex compiler, ensure that you include the required `workspace-runtime.swc` file.

2. Copy the SWF file to the LiveCycle ES repository by using Workbench ES. For information about importing a SWF file into LiveCycle ES, see the [LiveCycle Workbench ES Help](#) and go to Managing Resources > Creating folders and resources.

► **To configure your Flex application for testing:**

1. In Workbench ES, create a simple human-centric process that includes your Flex application and include initial data based on the data model that was defined in the Flex application. Save and activate the process as a service.

For information about creating a human-centric process, see [LiveCycle Workbench ES Help](#) and go to Creating Processes > Designing Human-Centric Processes. To pass data from Workspace ES to the process, the form variable you define as input in the process must be a `Form` data type.

For information about configuring a Flex application in a process, see the “Leverage Flex applications in LiveCycle Workspace ES” Quick Start in [LiveCycle Workbench ES Help](#).

2. Configure a TaskManager endpoint with the proper permissions in Applications and Services within LiveCycle Administration Console.

For information about configuring a process to be invoked from Workspace ES, see [Applications and Services Help](#) and go to Managing Endpoints > Adding endpoints to services.

## Testing Flex applications

After you deploy and configure your human-centric process, you can test your Flex application from within Workspace ES.

► **To test the Flex application within Workspace ES:**

1. Log in to Workspace ES.
2. Click **Start Process**. You should see the category and endpoint you created for the automated process that uses your Flex application.
3. Click the process that you want to use to test your Flex application. Your Flex application appears.
4. Validate that the Flex application is displayed correctly.

**Note:** If you configured the process to have initialization data, some fields may already contain values in the Flex application.

5. Fill the form in the Flex application and step through the application logic. These tasks are specific to the application logic of your Flex application.
6. Click the **Save As Draft** button to verify that the form is saved as a draft.
7. Log out of and then log back in to Workspace ES.
8. Click **Drafts** and verify that the form contains the data you specified in step 5.
9. Click **Complete** and verify that the next person in your business process received the form.
  - If your Flex application validates form data, verify that those validation checks work properly.
  - If your Flex application provides the status of form data, verify that you do not need to save your form when you do not make changes to the Flex application and exit the task.

## Troubleshooting

This table contains common problems you may experience when developing and testing your Flex application.

Problem	Proposed solution
Compile problems: FormConnector not found.	Verify that you loaded the workspace-runtime.swc file into your Flex Builder project or, if you are compiling using the Flex SDK, ensure that you included the SWC file as one of the parameters.
Flex application does not prepopulate with information.	Verify that the data is being sent from the process. Verify that the form data that is being sent as initialization data matches the schema your Flex application uses.
Complete button does not become enabled in Workspace ES.	Verify that you did not inadvertently send a <code>formDataInvalid</code> event.
Workspace ES cannot display your form.	Verify the permissions of the SWF file you copied into the repository in LiveCycle ES. Verify that you are using a form variable of data type <code>Form</code> to handle XML data if your form does not conform to the XFA-compliant schema.
The automated process cannot access form data.	Verify that the schema you are using matches the one used in the Flex application. Verify that the <code>setSubmitData</code> method from the <code>lc:FormConnector</code> component is called and the form data is provided as an XML object.

## Sample Code of Flex Application enabled in LiveCycle Workspace ES

---

The following example is the MXML file that contains the code used as examples in this document:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
*
* ADOBE SYSTEMS INCORPORATED
* Copyright 2008 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.
-->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:lc="http://www.adobe.com/2006/livecycle"
    layout="absolute"
    creationComplete="lcConnector.setReady()">

    <!-- Define the data model -->
    <mx:XML id="mydata">
        <data>
            <mortgageInfo>
                <propertyPrice> </propertyPrice>
                <downPayment> </downPayment>
                <mortgageAmount> </mortgageAmount>
            </mortgageInfo>
            <contactInfo>
                <lastName> </lastName>
                <firstName> </firstName>
                <phoneNum> </phoneNum>
            </contactInfo>
        </data>
    </mx:XML>

    <mx:Script>
        <![CDATA[

            import mx.events.ValidationResultEvent;
            import mx.controls.Alert;

            /* The formInitDataHandler event handler is dispatched by Workspace ES
             * and loads any initial data into the form. Initial data is null when
             * there is no data to be displayed in the form.
             */
            private function formInitDataHandler(event:DataEvent):void
            {
                if (event.data != null && event.data != "")
                {
```

```
        mydata = new XML(event.data);
    }
    else
    {
        //Empty data.
        mydata = new XML("<data/>");
    }
    lcConnector.setClean();

    var validationMortgageAmount:ValidationResultEvent =
        mortgageValidator.validate(null,true);
    var validationPhoneNum:ValidationResultEvent =
        phoneNumValidator.validate(null, true);
    if (( validationMortgageAmount.type == ValidationResultEvent.INVALID)
        || (validationPhoneNum.type == ValidationResultEvent.INVALID ))
    {
        lcConnector.setDataInvalid();
    }
}

/* The formSaveDataRequestHandler event handler executes when the user
 * clicks the Save As Draft button within Workspace ES. No
 * validation is performed on the data that is saved.
 */
private function formSaveDataListener (event:Event):void
{
    lcConnector.setSaveData(mydata);
    lcConnector.setClean();
}

/* The formSubmitDataRequestListener responds to the
 * formSubmitDataRequest event.The formSubmitDataRequest event
 * is triggered when the a user clicks the "Complete" button
 * or one of the task route buttons within Workspace ES.
 * A setSubmitData event is dispatched if the data contains no
 * errors; otherwise, the setSubmitDataInvalid event
 * should be dispatched.
 */
private function formSubmitDataListener(event:Event):void
{
    var validationMortgageAmount:ValidationResultEvent =
mortgageValidator.validate(null,true);
    var validationPhoneNum:ValidationResultEvent =
phoneNumValidator.validate(null, true);

    // Verify that all the necessary data is valid before
    // dispatching the formSubmitData event with the
    // setSubmitData (xmldata) method.
    if (( validationMortgageAmount.type == ValidationResultEvent.VALID) &&
        (validationPhoneNum.type == ValidationResultEvent.VALID ))
    {
        lcConnector.setSubmitData(mydata);
    }
    //Data was not valid, dispatch the formSubmitDataInvalid
    //event using the setSubmitDataInvalid method.
    else
```



```
        {
            lcConnector.setSubmitDataInvalid();
        }
    }
]]>
</mx:Script>

<!-- Add a FormConnector object -->
<lc:FormConnector id="lcConnector"
    formInitialData="formInitDataHandler(event)"
    formSaveDataRequest="formSaveDataListener(event)"
    formSubmitDataRequest="formSubmitDataListener(event)" />

<!-- Create the Form -->
<mx:VBox backgroundColor="white">
    <mx:HBox>
        <mx:Image id="banner"
            source="@Embed('financeCorpLogo.jpg')"
            scaleContent="true" width="100" height="50"/>
        <mx:Label text="Mortgage Application Form" fontSize="28"
            color="black" fontWeight="bold" fontFamily="Arial"/>
    </mx:HBox>
    <mx:Label text="After you complete this form, one of our representatives
        will contact you within two business days."
        fontSize="14" fontThickness="10" fontFamily="Arial"/>
    <mx:HBox>
        <mx:Form backgroundColor="white">
            <mx:FormItem label="Property Sale Price" required="false">
                <mx:TextInput id="propertySalePrice"
                    text="{mydata.mortgageInfo.propertyPrice}"
                    change="mydata.mortgageInfo.propertyPrice =
                        propertySalePrice.text;
                        mydata.mortgageInfo.mortgageAmount =
                            mydata.mortgageInfo.propertyPrice -
                                mydata.mortgageInfo.downPayment;
                        lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Down Payment" required="false">
                <mx:TextInput id="downPayment"
                    text="{mydata.mortgageInfo.downPayment}"
                    change="mydata.mortgageInfo.downPayment =
                        downPayment.text;
                        mydata.mortgageInfo.mortgageAmount =
                            mydata.mortgageInfo.propertyPrice -
                                mydata.mortgageInfo.downPayment;
                        lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Mortgage Amount" required="true">
                <mx:TextInput id="mortgageAmount"
                    text="{mydata.mortgageInfo.propertyPrice -
                        mydata.mortgageInfo.downPayment}"
                    change="mydata.mortgageInfo.mortgageAmount =
```

```
                mortgageAmount.text;
                lcConnector.setDirty();"/>
</mx:FormItem>
<mx:FormItem label="Last Name" required="false">
    <mx:TextInput id="lastName"
        text="{mydata.contactInfo.lastName}"
        change="mydata.contactInfo.lastName = lastName.text;
                lcConnector.setDirty();"/>
</mx:FormItem>
<mx:FormItem label="First Name" required="false">
    <mx:TextInput id="firstName"
        text="{mydata.contactInfo.firstName}"
        change="mydata.contactInfo.firstName =
                firstName.text;
                lcConnector.setDirty();"/>
</mx:FormItem>
<mx:FormItem label="Phone Number" required="true">
    <mx:TextInput id="phoneNum"
        text="{mydata.contactInfo.phoneNum}"
        change="mydata.contactInfo.phoneNum =
                phoneNum.text;
                lcConnector.setDirty();"/>
</mx:FormItem>
</mx:Form>
</mx:HBox>
</mx:VBox>

<!-- The validator for the field that stores the mortgage amount
and phone number. -->
<mx:NumberValidator id="mortgageValidator"
    source="{mortgageAmount}"
    required="true"
    minValue="0"
    lowerThanMinError="Mortgage must be greater than 0."
    valid="lcConnector.setDataValid()"
    invalid="lcConnector.setDataInvalid()"
    property="text"
    domain="real"
    listener="mortgageAmount"/>

<mx:PhoneNumberValidator id="phoneNumValidator"
    source="{phoneNum}"
    requiredFieldError="You must provide a phone number."
    valid="lcConnector.setDataValid();"
    invalid="lcConnector.setDataInvalid();"
    property="text"/>

</mx:Application>
```