



# Creating Flex™ Applications Enabled for LiveCycle® Workspace ES

August 2007

**Adobe® LiveCycle® Workspace ES**

Version 8.0

© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle® ES 8.0 Creating Flex™ Applications Enabled for LiveCycle Workspace ES for Microsoft® Windows®  
Edition 1.0, August 2007

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, Flex Builder, ImageReady, LiveCycle, Minion, Myriad, PostScript, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

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

Sun and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

All other trademarks are the property of their respective owners.

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

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

This product includes code licensed from RSA Data Security.

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

Macromedia Flash 8 video is powered by On2 TrueMotion video technology. © 1992-2005 On2 Technologies, Inc. All Rights Reserved.  
<http://www.on2.com>.

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

Portions of this code are licensed from Nellymoser ([www.nellymoser.com](http://www.nellymoser.com)).

MPEG Layer-3 audio compression technology licensed by Fraunhofer IIS and THOMSON Multimedia (<http://www.iis.fhg.de/amm/>).

This product includes software developed by L2FProd.com (<http://www.L2FProd.com/>).

The JBoss library is licensed under the GNU Library General Public License, a copy of which is included with this software.

The BeanShell library is licensed under the GNU Library General Public License, a copy of which is included with this software.

This product includes software developed by The Werken Company.

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

The OpenOffice.org library is licensed under the GNU Library General Public License, a copy of which is included with this software.

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>4</b>
Who should read this document? .....	4
Before you begin .....	4
Additional information .....	4
<b>1 Introduction .....</b>	<b>6</b>
Creating a Flex application enabled for Workspace ES .....	7
Enabling an existing Flex application for Workspace ES .....	7
Sample files.....	7
<b>2 Configuring Your Development Environment .....</b>	<b>8</b>
Synchronizing with the Flex SDK version used by LiveCycle ES .....	8
Accessing Workspace ES component library .....	9
Creating Flex projects to use the Workspace ES API .....	10
<b>3 Creating a Flex Application for Data Capture .....</b>	<b>12</b>
Creating a Flex application as a form .....	13
Adding a data model .....	15
Adding form validation .....	17
<b>4 Enabling a Flex Application for Workspace ES.....</b>	<b>19</b>
Adding the FormConnector component.....	21
Validating the data stored in the Flex application .....	22
Indicating that form data stored in the Flex application has changed .....	23
Creating events listeners to handle events from Workspace ES .....	26
<b>5 Deploying and Testing Flex Applications in Workspace ES .....</b>	<b>31</b>
Deploying a Flex application.....	31
Testing a Flex application.....	32
Troubleshooting .....	33
<b>A Sample Code .....</b>	<b>34</b>

# About This Document

This document contains information that describes how to perform the following tasks:

- Create a Flex application to provide form-like capabilities for capturing data.
- 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 Adobe Flex™, ActionScript™ 3.0, MXML, XML, and using 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), Adobe LiveCycle Workbench ES, and are familiar with Workspace ES.

## Before you begin

You must have Flex Builder 2.0.1 with hotfix 2, the eclipse Flex 2.0.1 plug-in with hotfix 2, or the same version of Flex SDK used by LiveCycle ES installed on your computer. You should have access to the LiveCycle ES SDK directory on the LiveCycle ES server or on your computer.

message URL

[http://www.adobe.com/go/learn\\_lc\\_overview](http://www.adobe.com/go/learn_lc_overview)

## Additional information

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

For information about	See
A LiveCycle ES Overview	<i>LiveCycle ES Overview</i> , available at <a href="http://www.adobe.com/go/learn_lc_overview">http://www.adobe.com/go/learn_lc_overview</a>
End-to-end process of creating a LiveCycle ES application that includes a form and human-centric processes, configuring services, and testing within LiveCycle Workspace ES.	<i>Creating Your First LiveCycle ES Application</i> , available at <a href="http://www.adobe.com/go/learn_lc_firstApplication">http://www.adobe.com/go/learn_lc_firstApplication</a>
LiveCycle Workspace ES.	<i>LiveCycle Workspace ES Help</i> , available within LiveCycle Workspace ES.
The ActionScript classes and properties included with LiveCycle ES.	<i>LiveCycle ES ActionScript Language</i> , available at <a href="http://www.adobe.com/go/learn_lc_fgActionScript">http://www.adobe.com/go/learn_lc_fgActionScript</a>
Rendering and deploying form guides using processes created in Workbench ES.	<i>LiveCycle Workbench ES Help</i> , available within LiveCycle Workbench ES

<b>For information about</b>	<b>See</b>
Detailed information about the classes and methods included with LiveCycle ES.	<i>LiveCycle ES Java API Reference</i> , available at <a href="http://www.adobe.com/go/learn_lc_JavaAPI">http://www.adobe.com/go/learn_lc_JavaAPI</a>
LiveCycle ES terminology	<i>LiveCycle ES Glossary</i> , available at <a href="http://www.adobe.com/go/learn_lc_glossary">http://www.adobe.com/go/learn_lc_glossary</a>
Installing Flex Builder.	<a href="http://www.adobe.com/go/flex2_installation">http://www.adobe.com/go/flex2_installation</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>

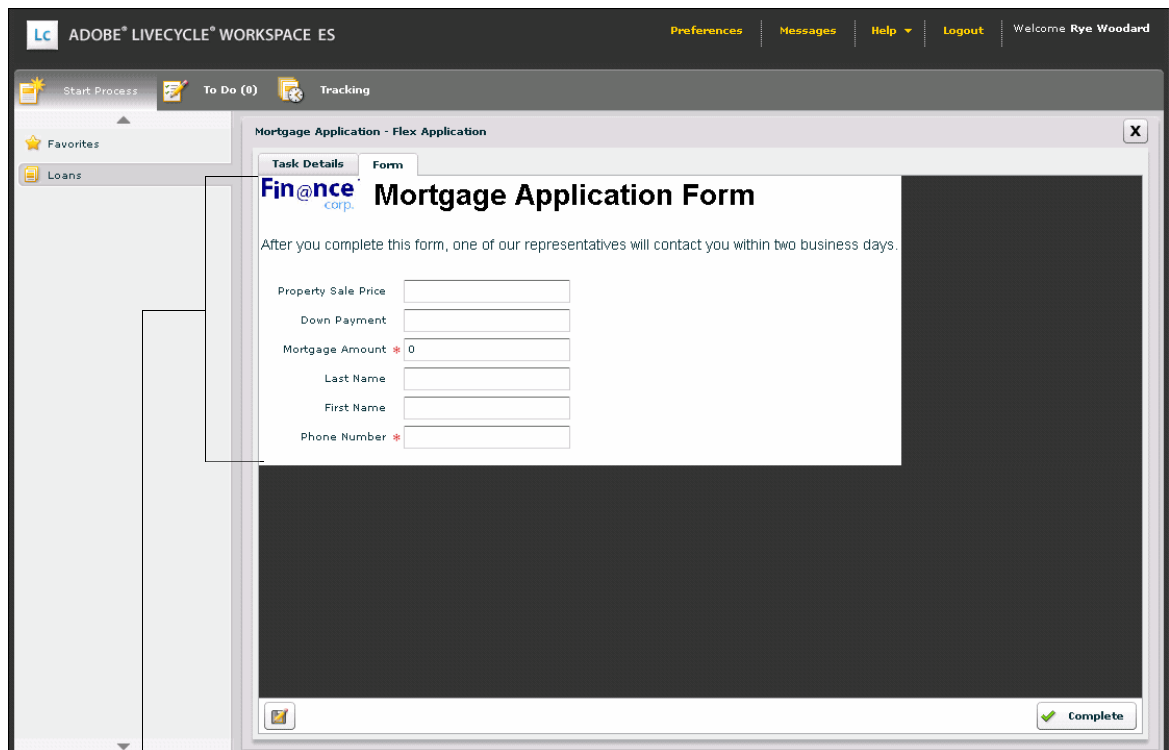
# 1

## 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®, and then enable it for use within LiveCycle Workspace ES. If you have an existing Flex application, you can also enable it for use within LiveCycle Workspace ES. Flex applications, such as 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 an enriched user experience by providing different visualizations and integrations for tasks to be created that go beyond those offered by PDF forms, HTML forms, and form guides. For instance, dashboards can be presented to users, audio and video can be provided for training or Help purposes, and more advanced user-interface components can be incorporated.

For example, when the Flex application is displayed within Workspace ES, the Flex application will respond to user actions such as clicking the Complete button 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.



Flex application

A Flex application that is enabled to communicate with Workspace ES functions as if it were part of Workspace ES, which itself is also a Flex application. Flex applications must be configured with human-centric processes to appear in Workspace ES. For information about configuring human-centric processes, see *LiveCycle Workbench ES Help* at [http://www.adobe.com/go/learn\\_lc\\_workbench](http://www.adobe.com/go/learn_lc_workbench) and navigate to Creating Processes > Designing Human-Centric Processes.

## 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. Implementing the application logic for the Flex application. (See [Creating a Flex application as a form](#).)
3. Enabling the Flex application for Workspace ES. (See [Enabling a Flex Application for Workspace ES](#).)
4. 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 development environment to develop Flex applications enabled for Workspace ES. (See [Configuring Your Development Environment](#).)
2. Enabling the Flex application for Workspace ES. (See [Enabling a Flex Application for Workspace ES](#).)
3. 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, which is available when you install the LiveCycle ES Samples. The sample process is located at `[install directory]\LiveCycle_ES_SDK\samples\LiveCycleES\SimpleMortgageLoan-Flex`, where `[install directory]` represents the location where you installed the LiveCycle ES server or Workbench ES.

## 2

# Configuring Your Development Environment

---

Before you can begin creating Flex applications enabled for Workspace ES, you must locate and copy the required SWC file from LiveCycle ES to access the Workspace ES API. You must ensure that your Flex development environment uses the same Flex SDK version as LiveCycle ES.

To use the Workspace ES APIs, the Workspace ES SWC files must be in your build path. These SWC files are located in the LiveCycle ES SDK directory and allow you to use components, interfaces, and classes from the Workspace ES API. When using Flex Builder, you simply build your project; however, when you build your Flex application using the Flex SDK, you need to 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 have been completed:

- Flex Builder 2.0.1 (or the plug-in to Eclipse) with hotfix 2 or Flex SDK 2.0.1 with hotfix 2 is installed on your computer.

**Note:** Flex Builder Hotfix 2 is available at

<http://kb.adobe.com/selfservice/viewContent.do?externalId=kb402000>.

- The LiveCycle ES SDK directory is accessible and all the samples are installed. You can access this directory on the LiveCycle ES server or from a computer where LiveCycle Workbench ES is installed.

## Summary of steps

The following summary outlines the steps to take to configure your development environment with Flex Builder 2.0.1:

1. If you do not have Flex Builder 2.0.1 hotfix 2 applied, you must synchronize with the Flex SDK version used by LiveCycle ES. (See [Synchronizing with the Flex SDK version used by LiveCycle ES](#).)
2. Locate and copy the SWC file from the LiveCycle ES SDK directory to the computer on which you are developing your Flex application. (See [Accessing Workspace ES component library](#).)
3. Create a Flex project in Flex Builder to include the Workspace ES API. (See [Creating Flex projects to use the Workspace ES API](#).)

## Synchronizing with the Flex SDK version used by LiveCycle ES

The version of the Flex SDK used with LiveCycle ES must be the same as the Flex SDK that you use to build a Flex application. If they are not the same, you cannot create a Flex application that functions properly within Workspace ES. The proper Flex SDK version is included on your LiveCycle ES DVD.

**Caution:** Complete the following procedure only if you do not have Flex Builder 2.0.1 hotfix 2 installed on your computer.

► **To synchronize the local Flex SDK with the Flex SDK used by LiveCycle ES:**

1. In the `lifecycle_dataservices` directory located on your LiveCycle ES DVD, copy the `flex_sdk_2.zip` file to the computer where you have installed Flex Builder 2.0.1.
2. Close Flex Builder if you have it running.
3. Back up the Flex SDK 2 directory located in `C:\Program files\Adobe\Flex Builder 2`.
4. Unzip `C:\lids\resources\flex_sdk\flex_sdk_2.zip` into the Flex Builder SDK folder (`C:\Program files\Adobe\Flex Builder 2\Flex SDK 2\`). The Flex Builder SDK folder must be named Flex SDK 2.
5. Make a copy of the Flex Builder lib folder that has all the JARs that we use for compilation. The lib folder is in `C:\Program Files\Adobe\Flex Builder 2\plugins\com.adobe.flexbuilder.flex_2.0.155577\` (or if you are using the plug-in version of Flex Builder, it is in the plug-ins folder of your Eclipse installation).
6. Copy the lib folder located in `C:\Program files\Adobe\Flex Builder 2\Flex SDK 2\` and paste it into the Flex Builder plug-in folder located in `C:\Program Files\Adobe\Flex Builder 2\plugins\com.adobe.flexbuilder.flex_2.0.155577\`.
7. In the lib folder copy you made in step 3, copy the `flex-compiler-oem.jar` file and paste it into the new lib folder you created in step 4.

**Tip:** If you have existing projects that were created or compiled using the previous Flex SDK version, it is recommended that you close, reopen, clean, and then rebuild them.

## Accessing Workspace ES component library

To use a Flex application as a form within Workspace ES, the Flex application must use the Workspace ES API. To access the Workspace ES API, you must copy the appropriate SWC files to your computer.

► **To access the Workspace ES component library:**

1. Create a folder on your computer to store SWC files that you use for development.
2. Access the LiveCycle ES SDK directory at one of the following locations:

**Note:** 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 SWC file when you have Workbench ES installed on your computer, browse to `[installdir]\LiveCycle ES\Workbench ES\LiveCycle_ES_SDK\misc\Process_Management\Workspace`, where `[installdir]` represents where Workbench ES was installed on your computer.
- To access the SWC file from the LiveCycle ES server, you must access the server and browse to `[installdir]\LiveCycle_ES_SDK\misc\Process_Management\Workspace` where `[installdir]` represents the location where LiveCycle ES is installed on a server.

For example, using a turnkey installation, browse to `\Adobe\LiveCycle8\LiveCycle_ES_SDK\misc\Process_Management\Workspace`.

3. Copy the `workspace-runtime.swc` file to the folder you created in step 1.

4. Access one of the locale folders for your preferred language at one of the following locations:
  - To access the SWC file when you have Workbench ES installed on your computer, browse to `[installdir]\LiveCycle ES\Workbench ES\LiveCycle_ES_SDK\misc\Process_Management\Workspace\locale`, where `[installdir]` represents where Workbench ES was installed on your computer.
  - To access the SWC file from the LiveCycle ES server, you must access the server and browse to `[installdir]\LiveCycle_ES_SDK\misc\Process_Management\Workspace\locale` where `[installdir]` represents the location where LiveCycle ES is installed on a server.  
For example, for an english locale, browse to  
`\Adobe\LiveCycle8\LiveCycle_ES_SDK\misc\Process_Management\Workspace\locale\en`.
5. Copy the `workspace_rb.swc` file to the folder you created in step 1.
6. Access the application server where LiveCycle ES was deployed and browse to the `adobe-workspace-runtime-exp.war` folder that contains the required Flex Data Services SWC file.  
For example, for a turnkey installation, browse to `[installdir]\Adobe\LiveCycle8\jboss\server\all\tmp\deploy\tmp15107adobe-livecycle-jboss.ear-contents\adobe-workspace-runtime-exp.war\WEB_INF\flex\libs`, where `[installdir]` represents the location where LiveCycle ES is installed on a server.
7. Copy the `fds.swc` file to the folder you created in step 1.

## Creating Flex projects to use the Workspace ES API

For a Flex application to use the Workspace ES API, you must include the `workspace-runtime.swc` file in your Flex project and also create a namespace in your MXML file to access classes, components, and interfaces from the Workspace ES API.

### ► To use the Workspace ES API in a Flex Builder project:

1. In Flex Builder, select **File > New > Flex Project**.
2. In the **New Flex Project** dialog box, select **Basic** (for example, XML or web service from PHP/JSP/ASP.NET) and click **Next**.
3. In the **New Flex Project** dialog box, in the **Project Name** box, type a name for the Flex project.
4. Perform one of following tasks in the Project Contents area, and then click **Next**.
  - To save your project to a default location, select **Use Default Location**.
  - Click **Browse** and select the folder to save your project to, and then click **OK**.
5. In the **New Flex Project** dialog box, click the **Library Path** tab, and then click **Add SWC**.
6. In the Add SWC dialog box, click **Browse** and select the `workspace-runtime.swc` file that you copied to your computer and click **Open**.
7. Repeat step 6 for the `workspace_rb.swc` and `fds.swc` files.
8. When you return to the New Flex Project dialog box, click **Finish**.
9. In the MXML file that contains the Application object for your project, add the LiveCycle namespace 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:

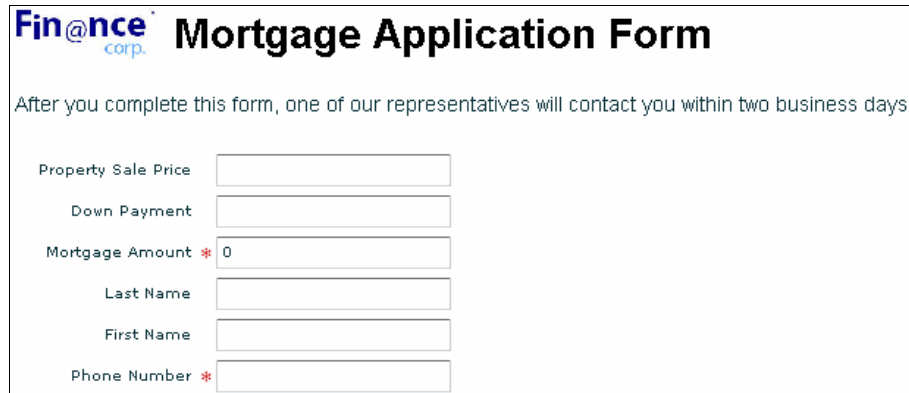
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:lc="http://www.adobe.com/2006/livecycle"
  layout="absolute">
</mx:Application>
```

**Caution:** The SWC file will be automatically included each time you compile using Flex Builder. If you compile using the command line compiler, you must reference the workspace-client.swc file when compiling. For more information about including SWCs as compile options, see the Flex documentation at [http://www.adobe.com/go/learn\\_lc\\_Flex](http://www.adobe.com/go/learn_lc_Flex).

# 3

## Creating a Flex Application for Data Capture

You can create a variety of sophisticated Flex applications as forms. For the purposes of this document, the steps to create a simple form will be described. After completing the tasks described in this section, you will have a Flex application as a form. For example, a simple Flex application may look like the following illustration:



**Fin@nce<sup>™</sup>**  
corp.

### Mortgage Application Form

After you complete this form, one of our representatives will contact you within two business days.

Property Sale Price

Down Payment

Mortgage Amount \* 0

Last Name

First Name

Phone Number \*

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

- A Fin@nce Corp. logo on the upper-left corner of the form.
- A Mortgage Application Form title.
- Instructions for filling out the form.
- Six fields arranged horizontally that include 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 right 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.
- Reduces the likelihood that an error occurs during the execution of your business process.

### Summary of steps

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

1. Create a Flex application by using the `mx:Form` and `mx:FormItem` components. (See [Creating a Flex application as a form.](#))
2. Add a data model to the Flex application. (See [Adding a data model.](#))
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 a Flex application as a form

A Flex application can be created as a form by using `mx:Form` and `mx:FormItem` components. You are not limited to using only the components described in this document. 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 *Flex 2 Developer's Guide* at [http://www.adobe.com/go/learn\\_lc\\_Flex](http://www.adobe.com/go/learn_lc_Flex) and navigate to Flex 2 Developer's Guide > Building User Interfaces for Flex applications > Using Layout Containers > Form, FormHeading, and FormItem layout containers.

For the purposes of this document, the basic steps to building a form are described. Specifically, the steps describe how to use the `mx:Form` and `mx:FormItem` containers that are used 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, will be routed to a specific person to review and approve the loan. The example code in this document 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 tasks:

1. If you have not already done so, add 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 1, 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 by adding a `mx:Label` control.
4. Within the `mx:VBox` container from step 1, add an `mx:Form` container.
5. In the `mx:Form` container, configure properties such as `width`, `height`, `backgroundColor`, and `foregroundColor` for the form.

6. (Optional) Within the `mx:Form` container from step 3, add an `mx:FormHeading` container.
  - Add an `mx:FormHeading` container so that it is nested within the `mx:Form` container.
  - Set the heading for the form by setting the `label` attribute.
  - Set other attributes you want, such as `fontSize`, `fontWeight`, and `color`. For more information about the attributes available for the `mx:FormHeading` container, see *LiveCycle ES ActionScript Language*, available at [http://www.adobe.com/go/learn\\_lc\\_fgActionScript](http://www.adobe.com/go/learn_lc_fgActionScript).
7. Within the `mx:Form` container from step 3, 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. Each `mx:FormItem` container is to be nested within the `mx:FormHeading` 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">
  <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"/>
        </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:Form>
    </mx:HBox>
  </mx:VBox>
</mx:Application>
```

```
</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>
```

## Adding a data model

In your Flex application, you 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. One way to accomplish this is by using the `mx:Model` component.

For more information about adding a data model, see *Flex 2 Developer's Guide* at [http://www.adobe.com/go/learn\\_lc\\_flex](http://www.adobe.com/go/learn_lc_flex) and navigate to Flex 2 Developer's Guide > Flex Data Features > Storing Data > Defining a data model.

The `mx:Model` component is convenient for creating two-way binding between the data model you use and each element within the `mx:Form` container; two-way binding is not possible with XML class. The `mx:Model` component provides an identity mapping between the XML and a data model, and is used for overloading and saving the data to a model from the `mx:FormItem` container.

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

1. Add an `mx:Model` component:
  - Add an `mx:Model` 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:Model` component element tags.
2. Map form data to the data model defined in the `mx:Model` 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:Model` component.
    - For each XML element, map it to an entry in the form you created.

### **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:Model id="mydata">
    <data>
```

```
<mortgageInfo>
  <propertyPrice> {propertySalePrice.text} </propertyPrice>
  <downPayment> {downPayment.text} </downPayment>
  <mortgageAmount> {mortgageAmount.text} </mortgageAmount>
</mortgageInfo>
<contactInfo>
  <lastName> {lastName.text} </lastName>
  <firstName> {firstName.text} </firstName>
  <phoneNum> {phoneNum.text} </phoneNum>
</contactInfo>
</data>
</mx:Model>
<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}"/>
      </mx:FormItem>
      <mx:FormItem label="Down Payment" required="false">
        <mx:TextInput id="downPayment"
          text="{mydata.mortgageInfo.downPayment}" />
      </mx:FormItem>
      <mx:FormItem label="Mortgage Amount" required="true">
        <mx:TextInput id="mortgageAmount"
          text="{mydata.mortgageInfo.propertyPrice -
            mydata.mortgageInfo.downPayment}" />
      </mx:FormItem>
      <mx:FormItem label="Last Name" required="false">
        <mx:TextInput id="lastName"
          text="{mydata.contactInfo.lastName}" />
      </mx:FormItem>
      <mx:FormItem label="First Name" required="false">
        <mx:TextInput id="firstName"
          text="{mydata.contactInfo.firstName}"/>
      </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>
</mx:Application>
```

## Adding form validation

The form data must be stored and passed as XML to be used in a process within LiveCycle ES. The `mx:Model` component creates two-way bindings between the `mx:Model` component and each element within an `mx:Form` container, which is not possible with XML in ActionScript. The two-way binding provides an identity mapping between the XML and a Model object that allows for overloading and saving of data directly from the `mx:FormItem` container.

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 sure that the form data is marked as invalid.

For more information about handling form validation, see *Flex 2 Developer's Guide* at [http://www.adobe.com/go/learn\\_lc\\_Flex](http://www.adobe.com/go/learn_lc_Flex) and navigate to Flex 2 Developer's Guide > Flex Data Features > 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, a numeric value using the `mx:NumberValidator` component, or the `mx:PhoneNumberValidator` component to verify whether a phone number exists. Validation classes are in the `mx.validators` package and depending 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 set it to a value that represents the field you want to validate in your 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 that 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">
    ...
    ...

    <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>
</mx:Application>
```

```
<mx:HBox>
  <mx:Form backgroundColor="white">
    <mx:FormItem label="Property Sale Price" required="false">
      <mx:TextInput id="propertySalePrice"
        text="{mydata.mortgageInfo.propertyPrice}"/>
    </mx:FormItem>
    <mx:FormItem label="Down Payment" required="false">
      <mx:TextInput id="downPayment"
        text="{mydata.mortgageInfo.downPayment}" />
    </mx:FormItem>
    <mx:FormItem label="Mortgage Amount" required="true">
      <mx:TextInput id="mortgageAmount"
        text="{mydata.mortgageInfo.propertyPrice -
          mydata.mortgageInfo.downPayment}"
        change="lcConnector.setDirty();" />
    </mx:FormItem>
    ...
  </mx:Form>
</mx:HBox>
</mx:VBox>
<!-- The validator for the field that stores the mortgage amount -->
<mx:NumberValidator id="mortgageValidator"
  source="{mortgageAmount}"
  required="true"
  minValue="0"
  lowerThanMinError="Mortgage must be greater than 0." />
<mx:PhoneNumberValidator id="phoneNumValidator"
  source="{phoneNum}"
  requiredFieldError="You must provide a phone
  number." />

...
...

</mx:Application>
```

# 4

## Enabling a Flex Application for Workspace ES

---

When a Flex application is enabled for LiveCycle Workspace ES, the Flex application responds to actions in Workspace ES like any other form. Before you start this section, you should have a Flex application written as a form. For information about creating a Flex application as a form, see [Creating a Flex Application for Data Capture](#). The Flex application you are enabling for Workspace ES must have a data model, and you can optionally validate any data that it stores.

To enable a Flex application for Workspace ES, you use Flex's 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.

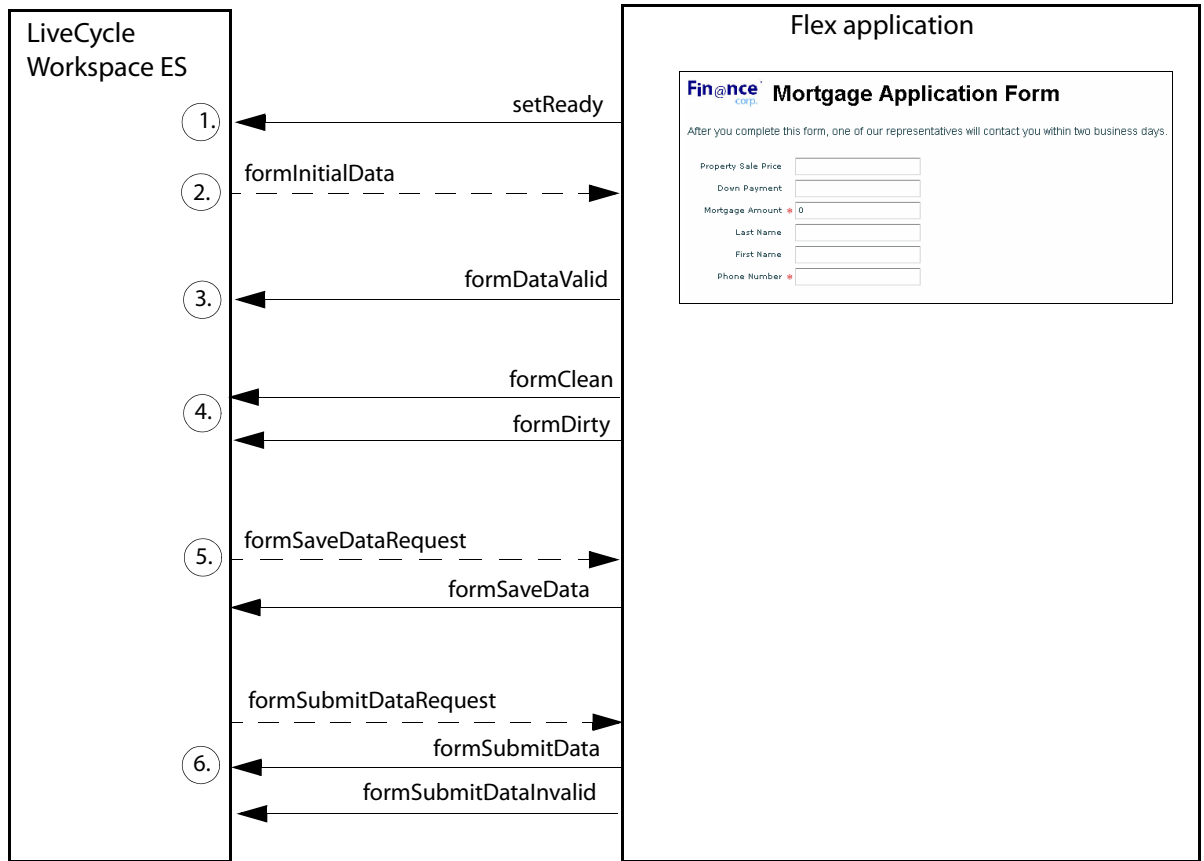
You may want to write the code to provide the communication between the Flex application and Workspace ES by using Flex event-handling APIs; however, it is recommended that you use the `lc:FormConnector` component, which is available from the Workspace ES API, to enable a Flex application for Workspace 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. The `flash.events.DataEvent` type is used for events that carry data and the `flash.data.Event` type is for all other events.

**Note:** Simple Flash types 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:

Each item in the following list corresponds to the numbering 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 very step that must occur because, otherwise, Workspace ES will not dispatch any events.

2. Workspace ES dispatches a `formInitialData` event to the Flex application with data, 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 Workbench ES when starting a process. For information about developing processes, see *LiveCycle Workbench ES Help* at [http://www.adobe.com/go/learn\\_lc\\_workbench](http://www.adobe.com/go/learn_lc_workbench) and navigate to Creating Processes > Designing Human-Centric Processes.

3. At minimum, the Flex application must dispatch the `formDataValid` event to enable the Complete or route buttons in Workspace ES. In Workspace ES, the Complete button is not available to a user until after a `formDataValid` event is received from the Flex application.

Optionally, both `formDataValid` and `formDataInvalid` events can be sent to indicate the validity of the data that the Flex application stores. In addition, instead of dispatching separate `formDataValid` and `formDataInvalid` events as the Flex application runs, you can optimize the performance of your Flex application by dispatching a single `formDataValid` event in the event listener for the `formInitialData` event. In this situation, your Flex application needs to 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 to save a draft form when they leave a task by default 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 will prompt a user only to save a draft form 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` 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 the data stored in the Flex application](#).)
3. (Optional) Dispatch events to indicate that data has changed. (See [Indicating that form data stored in the Flex application has changed](#).)
4. Create event listeners to handle events dispatched by Workspace ES. (See [Creating event listeners to handle events from Workspace ES](#).)

## Adding the FormConnector component

Before you use the `lc:FormConnector` component, you must configure your development process and include the `workspace-runtime.swc` file in your Flex project. (See [Configuring Your Development Environment](#).)

By using the `lc:FormConnector` component, which is available from the Workspace ES API, you can use events and methods that ease the communication between your Flex application and Workspace ES. After the Flex application has completed loading and is ready to communicate with Workspace ES, the `setReady` event must be dispatched to Workspace ES. The `setReady` event is used to notify to Workspace ES that the Flex application is ready to start receiving events. This event must be dispatched as the first event 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 `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 has completed 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 Form Connector object -->
  <lc:FormConnector id="lcConnector" />

...
...

</mx:Application>
```

## Validating the data stored in the Flex application

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 because the data should all 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` from the `lc:FormConnector` component to dispatch the `setDataInvalid` 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"
                       valid="connector.setDataValid()"
                       invalid="lcConnector.setDataInvalid()" />
    <mx:PhoneNumberValidator id="phoneNumValidator"
                            source="{phoneNum}"
                            requiredFieldError="You must provide a phone number."
                            valid="connector.setDataValid()"
                            invalid="lcConnector.setDataInvalid()" />

    ...
    ...

</mx:Application>
```

## Indicating that form data stored in the Flex application has changed

In Workspace ES, by default, each time a user leaves a form associated with a task, they are prompted to save the form data regardless of whether changes have been made. To change this behavior so that the user is only prompted to save the form when changes occur to the form data, the Flex application must dispatch an event to indicate whether the form data that it stored has 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 has changed. If you choose to use events to indicate when to save changes to the form data, your Flex application must also 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:

1. In your Flex component, locate each component or control where data is entered or changed. When a field changes its value, for most components, there is a change event that is available to bind to and add the attribute that specifies a change event. For example, for the `mx:TextInput` control, the `change` attribute was added.
2. Bind the `change` attribute to the `setDirty()` method from the `lc:FormConnector` component.
3. Add the `change` attribute and set it to dispatch a `formDirty` event. For example, for the `mx:TextInput` control, add the `change` attribute and bind it 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="lcConnector.setDirty();" />
        </mx:FormItem>
      </mx:Form>
    </mx:HBox>
  </mx:VBox>
</mx:Application>
```

```

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

...
...

</mx:Application>

```

To indicate that no changes have occurred on the form, perform this task:

- In the areas where you first load form data into a Flex application or save the form data from a Flex application, dispatch a `formClean` event to indicate that the form no longer needs to be saved.

**Example: Dispatching the `formClean` event after the Flex application receives the `formInitialData` event.**

```

private function formInitDataListener(event: DataEvent): void
{
  if ((event.data != null) || (event.data != " "))
  {
    //Map the XML data back to the Model object.
    var myXML:XML = new XML(event.data);
    data.mortgageInfo.propertyPrice = myXML.mortgageInfo.propertyPrice;
    lcConnector.setClean();
  }
}

```

## 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 the form data 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 prepopulate 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 design time of an automated process created within Workbench ES. The XML data should be formatted based on the same schema used by the Flex application (for example, the schema defined by the `mx:Model` component).

When Workspace ES first loads a Flex application, if initial data exists, it can be populated into your Flex application. The initial data that appears in your Flex application when you first start a process is configured in the automated business process using LiveCycle Workbench ES. For more information about configuring initialization data, see *LiveCycle Workbench ES Help* at [http://www.adobe.com/go/learn\\_lc\\_workbench](http://www.adobe.com/go/learn_lc_workbench) and navigate to Creating Processes > Designing Human-Centric Processes > Prepopulating forms with default data.

**formSaveDataRequest:** Specifies that the user has 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 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.

These events are the summarized events that your Flex application can dispatch 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 as XML data and populate the fields in the Flex application. If you have an instance of the `Model` object, the values can be populated using the `mx:Model` component.
4. (Optional) Dispatch the `formClean` event by using the `setClean()` method from the `lc:FormConnector` component if you choose to indicate to Workspace ES whether data has changed in your form.
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()">
    ...
    ...
    <lc:FormConnector id="lcConnector"
                    ...
                    ...
                    formInitialData="formInitDataHandler(event)" ...
    </lc:FormConnector>

    <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 != " "))
                {
                    //Extract the initialization data as XML
                    var myXML:XML = new XML(event.data);
                    //Assign the data to the Model object
                    data.mortgageInfo.propertyPrice =
                                                                myXML.mortgageInfo.propertyPrice;
                }
            }
        ]]>
    </mx:Script>

    ...
    ...

</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, nested within the `mx:Script` component, with one parameter of type `Event`, that returns `void` to handle the `formSaveDataRequest` event. For the function, perform the tasks that follow.
3. Extract the form data from the `mx:Model` component and save it in an object of type `XML`.
4. Dispatch the `formSaveData` event by calling the `setSaveData` method from the `lc:FormConnector` component and send the XML data back to Workspace ES.

**Note:** If you are using the `mx:Model` component to store your data, you must define an XML object by using the `mx:Model` component bound with the form data.

5. 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()" >
    ...
    ...
    <lc:FormConnector id="lcConnector"
        ...
        ...
        formSaveDataRequest="formSaveDataListener(event)" ...
    </lc:FormConnector>

    <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 valid.
            private function formSaveDataListener (event:Event):void
            {
                var myXML:XML =
                XML (<data>
                    <mortgageInfo>
                        <propertyPrice> {propertySalePrice.text}
                        </propertyPrice>
                        <downPayment> {downPayment.text} </downPayment>
                        <mortgageAmount> {mortgageAmount.text}
                    </mortgageInfo>
                </data>
            }
        ]]>
    </mx:Script>

```

```
        </mortgageAmount>
        </mortgageInfo>
        <contactInfo>
            <lastName> {lastName.text} </lastName>
            <firstName> {firstName.text}</firstName>
            <phoneNum> {phoneNum.text} </phoneNum>;
        </contactInfo></data>;
        lcConnector.setSaveData(myXML);
    }

    ...
    ...

    ]]>
</mx:Script>

...
...

</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, extract form data from the `mx:Model` component, store the form data as an XML object, and 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()">
    ...
    ...

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

```
        formSubmitDataRequest="formSubmitDataListener(event)"/>
<mx:Script>

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 ))
    {
        var myXML:XML = XML (<data>
            <mortgageInfo>
                <propertyPrice> {propertySalePrice.text}
            </propertyPrice>
            <downPayment>{downPayment.text}
            </downPayment>
            <mortgageAmount> {mortgageAmount.text}
            </mortgageAmount>
            </mortgageInfo>
            <contactInfo>
                <lastName> {lastName.text} </lastName>
                <firstName> {firstName.text}</firstName>
                <phoneNum> {phoneNum.text} </phoneNum>;
            </contactInfo>
        </data>);
        lcConnector.setSubmitData(myXML);
    }
    // Data was not valid, dispatch the formSubmitDataInvalid
    // event using the setSubmitDataInvalid method.
    else
    {
        // Data was not valid, dispatch the formSubmitDataInvalid
        // event using the setSubmitDataInvalid method.
        lcConnector.setSubmitDataInvalid();
    }
}
]]>
</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 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 include it in a process.

**Caution:** You can only test Workspace ES-enabled Flex applications within Workspace ES because the communication that occurs between the Flex application 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 configure the process, you must configure the process to be accessible from Workspace ES. For information about deploying your Flex application, see [Deploying a Flex application](#).

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

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 a Flex application

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 into the LiveCycle ES repository by using Workbench ES. For information about importing a SWF file into LiveCycle ES, see the *LiveCycle Workbench ES Help* at [http://www.adobe.com/go/learn\\_lc\\_workbench](http://www.adobe.com/go/learn_lc_workbench) and navigate 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 the Flex application. Save and activate the process as a service.
  - For information about creating a human-centric processes, see *LiveCycle Workbench ES Help* at [http://www.adobe.com/go/learn\\_lc\\_workbench](http://www.adobe.com/go/learn_lc_workbench) and navigate to Creating Processes > Designing Human-Centric Processes.
  - For information about configuring a Flex application in a process, see “Leverage Flex applications in LiveCycle Workspace ES” Quick Start in *LiveCycle Workbench ES Help*.
2. Configure a TaskManager endpoint with the proper permissions in Archive Administration within LiveCycle Administration Console.

For information about configuring a process to be invoked from Workspace ES, see *Archive Administration Help* at [http://www.adobe.com/go/learn\\_lc\\_adminAAC](http://www.adobe.com/go/learn_lc_adminAAC) and navigate to Managing Endpoints > Adding an endpoint.

## Testing a Flex application

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, then 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 have not made changes to the Flex application and have exited the task.

## Troubleshooting

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

<b>Problem</b>	<b>Proposed solution</b>
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 <code>Form</code> variable 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 <code>XML</code> object.

# A

## Sample Code

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

```
<?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()">

    <!-- Build the form -->
    <!-- The following is a sample "form" written in Flex -->
    <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="lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Down Payment" required="false">
                <mx:TextInput id="downPayment"
                    text="{mydata.mortgageInfo.downPayment}"
                    change="lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Mortgage Amount" required="true">
                <mx:TextInput id="mortgageAmount"
                    text="{mydata.mortgageInfo.propertyPrice -
                    mydata.mortgageInfo.downPayment}"
                    change="lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Last Name" required="false">
                <mx:TextInput id="lastName"
                    text="{mydata.contactInfo.lastName}"
                    change="lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="First Name" required="false">
                <mx:TextInput id="firstName"
                    text="{mydata.contactInfo.firstName}"
                    change="lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Phone Number" required="true">
```

```
        <mx:TextInput id="phoneNum"
                    text="{mydata.contactInfo.phoneNum}"
                    change="lcConnector.setDirty();" />
    </mx:FormItem>
</mx:Form>
</mx:HBox>
</mx:VBox>

<!-- Define the model -->
<!-- The data model for the form -->
<mx:Model id="mydata">
    <data>
        <mortgageInfo>
            <propertyPrice> {propertySalePrice.text} </propertyPrice>
            <downPayment> {downPayment.text} </downPayment>
            <mortgageAmount> {mortgageAmount.text} </mortgageAmount>
        </mortgageInfo>
        <contactInfo>
            <lastName> {lastName.text} </lastName>
            <firstName> {firstName.text} </firstName>
            <phoneNum> {phoneNum.text} </phoneNum>
        </contactInfo>
    </data>
</mx:Model>

<!-- The Form Connector object from the Workspace ES API -->
<lc:FormConnector id="lcConnector"
                  formInitialData="formInitDataListener(event)"
                  formSaveDataRequest="formSaveDataRequestListener(event)"
                  formSubmitDataRequest="formSubmitDataRequestListener(event)" />

<!-- 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." />
<mx:PhoneNumberValidator id="phoneNumValidator"
                          source="{phoneNum}"
                          requiredFieldError="You must provide a phone number." />

<mx:Script>
    <![CDATA[
        import mx.events.ValidationResultEvent;

        // The formInitDataListener function responds to the formInitialData
        // event that is dispatched by Workspace ES after a setReady event is
        // received. The formInitial event provides any initialization
        // data specified by the automated business process.
        private function formInitDataListener(event: DataEvent): void
        {
            if ((event.data != null) || (event.data != " "))
            {
```

```
        //Map the XML data back to the Model object.
        var myXML:XML = new XML(event.data);
        data.mortgageInfo.propertyPrice =
            myXML.mortgageInfo.propertyPrice;
        lcConnector.setClean();
    }
}

// The formSaveDataRequestListener responds to formSaveDataRequest
// event, which is triggered when a user clicks the Save As Draft button
// within Workspace ES. Validation of data is not necessary to allow
// the user to save in-progress data.
private function formSaveDataRequestListener (event:Event):void
{
    var myXML:XML = XML (<data>
        <mortgageInfo>
            <propertyPrice> {propertySalePrice.text}
        </propertyPrice>
            <downPayment> {downPayment.text}
        </downPayment>
            <mortgageAmount> {mortgageAmount.text}
        </mortgageAmount>
        </mortgageInfo>
        <contactInfo>
            <lastName> {lastName.text} </lastName>
            <firstName> {firstName.text}</firstName>
            <phoneNum> {phoneNum.text} </phoneNum>;
        </contactInfo>
    </data>);
    lcConnector.setSaveData(myXML);
    lcConnector.setClean()
}

// The formSubmitDataRequestListener responds to the
// formSubmitDataRequest event.The formSubmitDataRequest
// 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 formSubmitDataRequestListener(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 ))
    {
        var myXML:XML = XML (<data>
```

```
        <mortgageInfo>
            <propertyPrice> {propertySalePrice.text}
        </propertyPrice>
        <downPayment> {downPayment.text}
        </downPayment>
        <mortgageAmount> {mortgageAmount.text}
        </mortgageAmount>
        </mortgageInfo>
        <contactInfo>
            <lastName> {lastName.text} </lastName>
            <firstName> {firstName.text}</firstName>
            <phoneNum> {phoneNum.text} </phoneNum>;
        </contactInfo>
        </data>;
        lcConnector.setSubmitData(myXML);
    }
    // Data was not valid, dispatch the formSubmitDataInvalid event
    // using the setSubmitDataInvalid method.
else
{
    // Data was not valid, dispatch the formSubmitDataInvalid event
    // using the setSubmitDataInvalid method.
    lcConnector.setSubmitDataInvalid();
}
}
]]>
</mx:Script>
</mx:Application>
```