



Creating Flex Applications Enabled for LiveCycle Workspace ES2

March 2010

Adobe® LiveCycle® ES2

Version 9

© 2010 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle® ES2 (9.0) Creating Flex® Applications Enabled for LiveCycle Workspace ES2 for Microsoft® Windows®, Linux®, and UNIX® Edition 3.0, March 2010

This developer document is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the document for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the document; and (2) any reuse or distribution of the document contains a notice that use of the document is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>.

Adobe, the Adobe logo, Adobe Reader, Acrobat, Distiller, Flash, Flex, FrameMaker, LiveCycle, PageMaker, Photoshop, and PostScript 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.

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

Contents

About This Document

Who should read this document?	4
Before you begin	4
Additional information	5

Introduction

Sample files	7
--------------------	---

Configuring Your Development Environment for LiveCycle ES2

Install the LiveCycle ES2 version of the Flex SDK	8
Configure Flex Builder to use the LiveCycle ES2 version of the Flex SDK	9

Configuring Flex projects to use the Workspace API

Creating Flex Applications for Data Capture

Create a Flex application as forms	13
Add a data model to a Flex application	15
Add validation to a form	16

Enabling Flex Applications for Workspace ES2

Understanding Workspace API events and event handling	18
Add the SwfConnector component	22
Validate data and send events in the Flex applications	23
Indicate changes to the form data stored in Flex applications	24
Handle events from Workspace ES2	25

Deploying and Testing Flex Applications in Workspace ES2

Deploy Flex applications	29
Test Flex applications in Workspace ES2	30
Troubleshooting	31

Sample Code of Flex application enabled for Workspace ES2

About This Document

This document provides a walk-through of these tasks:

- Creating an application using Adobe Flex® to provide capabilities for capturing data from human-centric processes. Applications developed using Flex are referred to as Flex applications in this document and often informally called Flex forms.
- Enabling a Flex application for use within Adobe® LiveCycle® Workspace ES2.
- Deploying and testing a Flex application.

Each task is divided into subtasks at the end of which you find the MXML or ActionScript code. Subsequent sections build on the code provided in previous sections. **Bolded courier font** appears in the examples to indicate the new code that is incrementally added to each subtask. The complete example code appears at the end of the document.

Who should read this document?

This document is intended for developers who are familiar with the Flex SDK, ActionScript™ 3.0, MXML, and Adobe Flex® Builder™. Flex Builder is the tool used for the purposes of this document.

It is beneficial that developers have exposure to developing and testing human-centric LiveCycle ES2 applications. You develop LiveCycle ES2 applications using Adobe LiveCycle Workbench ES2 and test them using Workspace ES2.

Before you begin

Before creating Flex applications enabled for use in Workspace ES2, you must have access to the following items:

- An installation of Flex Builder 3.x. When you use the Flex plug-in for Eclipse, steps in this document do not correspond exactly because the menu commands differ.
- The LiveCycle ES2 DVD to install a LiveCycle ES2 version of the Flex SDK.
- The LiveCycle ES2 SDK folder. The LiveCycle ES2 SDK folder is available from the LiveCycle ES2 server or from a computer where Workbench ES2 is installed.
- Access to a LiveCycle ES2 server, Workbench ES2, and Workspace ES2. It is necessary that the samples are installed on the LiveCycle ES2 server to have a testing environment.

It is beneficial if you are familiar with developing human-centric processes. If you are not, consider completing the [Creating Your First LiveCycle ES2 Application](http://www.adobe.com/go/learn_lc_firstApplication_9) tutorial at http://www.adobe.com/go/learn_lc_firstApplication_9.

Additional information

The resources in this table can help you learn more about LiveCycle ES2.

For information about	See
An overview of LiveCycle ES2	LiveCycle ES2 Overview at http://www.adobe.com/go/learn_lc_overview_9
The end-to-end process of creating a LiveCycle ES2 application	Creating Your First LiveCycle ES2 Application at http://www.adobe.com/go/learn_lc_firstApplication_9
Adobe LiveCycle Workspace ES2	LiveCycle Workspace ES2 Help at http://www.adobe.com/go/learn_lc_euWorkspace_9
ActionScript classes and properties included with LiveCycle ES2 and Flex	LiveCycle ES2 ActionScript Language Reference at http://www.adobe.com/go/learn_lc_actionScript_9
Rendering and deploying Flex applications using processes created in Workbench ES2	Application Development Using LiveCycle Workbench ES2 at http://www.adobe.com/go/learn_lc_workbench_9
LiveCycle ES2 terminology	LiveCycle ES2 Glossary at www.adobe.com/go/learn_lc_glossary_9
Installing and using Flex Builder	Flex Help and Support website at http://www.adobe.com/go/learn_lc_Flex
Patch updates, technical notes, and additional information on this product version	http://www.adobe.com/go/learn_lc_support

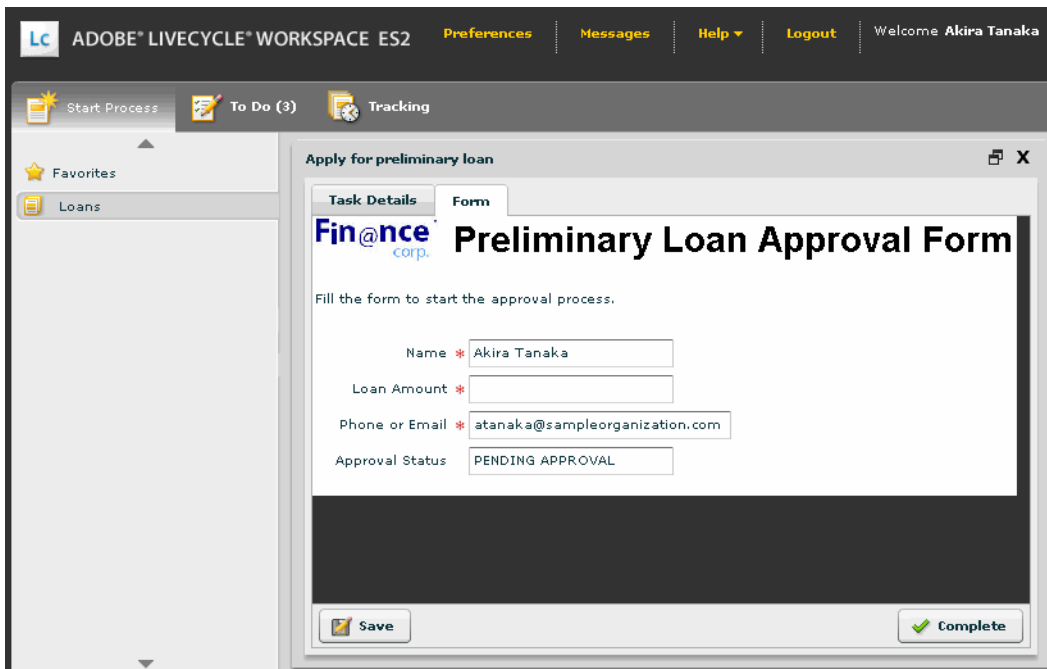
Introduction

You can create Flex applications that provide data capture capabilities in the same manner that forms created in Adobe® LiveCycle® Designer ES2 or Adobe® Acrobat® do. A Flex application that is enabled for Workspace ES2 functions as if it were part of Workspace ES2, which is itself a Flex application. Like PDF or HTML forms, you can use Flex applications along with human-centric processes to capture data from users that use Workspace ES2.

A Flex application uses events to communicate with Workspace ES2. The *communication* permits the Flex application to respond to user actions in Workspace ES2. For example, when the user clicks the Complete button, the form data stored by the Flex application is sent to Workspace ES2. In addition, a Flex application can use events to control aspects of the Workspace ES2 user interface and retrieve data about the task or the user. Some examples of the tasks are as follows:

- Displaying or hiding tabs
- Prompting for attachments
- Displaying the Flex application in full-screen mode
- Retrieving task instructions
- Retrieving the user's full name and email address.

The following illustration shows a sample Flex application within Workspace ES2. In this case, the Flex application looks like a traditional form.



The screenshot displays the Adobe LiveCycle Workspace ES2 interface. At the top, there is a navigation bar with 'ADOBE LIVECYCLE WORKSPACE ES2', 'Preferences', 'Messages', 'Help', 'Logout', and 'Welcome Akira Tanaka'. Below this, there are tabs for 'Start Process', 'To Do (3)', and 'Tracking'. A sidebar on the left shows 'Favorites' and 'Loans'. The main content area is titled 'Apply for preliminary loan' and contains a 'Form' tab. The form is for 'Fin@nce corp.' and is titled 'Preliminary Loan Approval Form'. It includes the instruction 'Fill the form to start the approval process.' and several input fields: 'Name' (filled with 'Akira Tanaka'), 'Loan Amount', 'Phone or Email' (filled with 'atanaka@sampleorganization.com'), and 'Approval Status' (filled with 'PENDING APPROVAL'). At the bottom of the form, there are 'Save' and 'Complete' buttons.

Note: You can create Flex applications that run as Web applications (runs in Adobe Flash® Player). AIR applications are not supported in LiveCycle Workspace ES2.

You can create a Flex application for data capture like the one that appears above and enable it for Workspace ES2. Alternatively, if you have an existing Flex application that has a data model, you can enable it for Workspace ES2 and then create a process that to use the Flex application.

Create a Flex application enabled for Workspace ES2

If you want to create a Flex application enabled for Workspace ES2, complete these tasks. The tasks described below leverage the human-centric process from the *Creating Your First LiveCycle ES2 Application* tutorial at http://www.adobe.com/go/learn_lc_firstApplication_9.

- 1 Set up your Flex development environment for LiveCycle ES2. (See “[Configuring Your Development Environment for LiveCycle ES2](#)” on page 8.)
- 2 Configure your Flex project to use the Workspace API. (See “[Configuring Flex projects to use the Workspace API](#)” on page 11.)
- 3 Implement the application logic for the Flex application. (See “[Creating Flex Applications for Data Capture](#)” on page 13.)
- 4 Enable the Flex application for Workspace ES2. (See “[Enabling Flex Applications for Workspace ES2](#)” on page 18.)
- 5 Deploy and test the Flex application. (See “[Deploying and Testing Flex Applications in Workspace ES2](#)” on page 29.)

Enable an existing Flex application for Workspace ES2

If your Flex application has a data model, you can enable a Flex application for use in a process by completing the following tasks:

- 1 Set up your Flex development environment for LiveCycle ES2. (See “[Configuring Your Development Environment for LiveCycle ES2](#)” on page 8.)
- 2 Configure your Flex project to use the Workspace API. (See “[Configuring Flex projects to use the Workspace API](#)” on page 11.)
- 3 Enable the Flex application for Workspace ES2. (See “[Enabling Flex Applications for Workspace ES2](#)” on page 18.)
- 4 Deploy and test the Flex application. (See “[Deploying and Testing Flex Applications in Workspace ES2](#)” on page 29.)

Sample files

The example code to implement the Flex application that is enabled for Workspace ES2 is available at the end of this document. (See “[Sample Code of Flex application enabled for Workspace ES2](#)” on page 32.)

You can copy the code into an MXML file and compile it using Flex Builder 3.x. Before you compile, ensure that you configure your Flex development environment for LiveCycle ES2 and configure the Workspace API in the project. (See “[Configuring Your Development Environment for LiveCycle ES2](#)” on page 8 and “[Configuring Flex projects to use the Workspace API](#)” on page 11.)

In addition, if you want to use the image in the source code, copy the `financeCorpLogo.jpg` file from the *Creating Your First LiveCycle ES2 Application* tutorial ZIP file. You can download the ZIP file at http://www.adobe.com/go/learn_lc_firstApplicationZip_9.

Configuring Your Development Environment for LiveCycle ES2

Complete these tasks to configure your development environment with the LiveCycle ES2 version of the Flex SDK:

- Install the LiveCycle ES2 version of the Flex SDK on your development computer. (See [“Install the LiveCycle ES2 version of the Flex SDK” on page 8.](#))
- Configure Flex Builder or the Flex project to compile with the LiveCycle ES2 version of the Flex SDK. (See [“Configure Flex Builder to use the LiveCycle ES2 version of the Flex SDK” on page 9.](#))

Before you can configure your development environment, ensure that you have access to the LiveCycle ES2 DVD and a LiveCycle ES2 server. Also ensure that Flex Builder is installed on your computer.

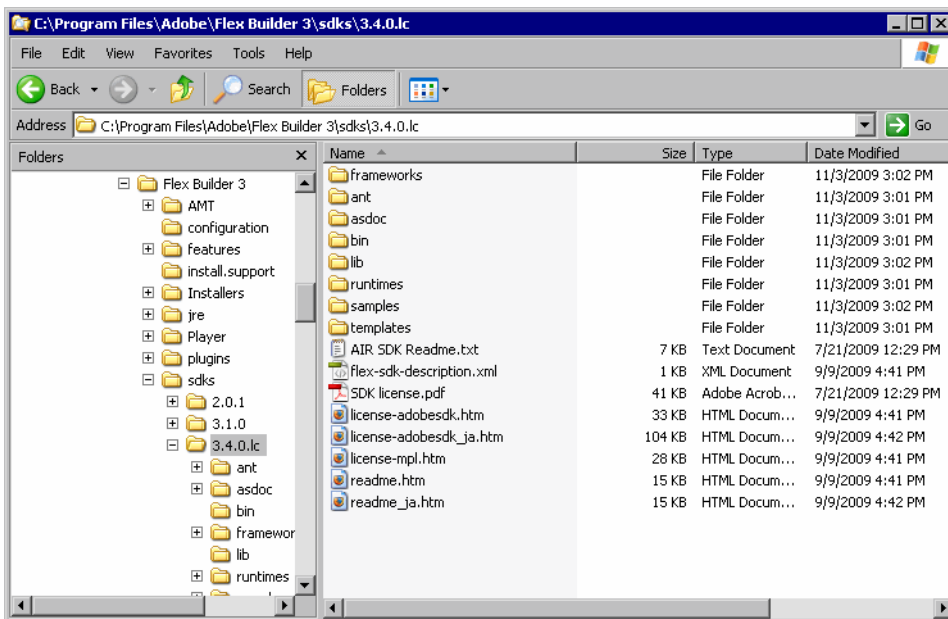
The LiveCycle ES2 version of the Flex SDK is available on the LiveCycle ES2 DVD includes these additional resources:

- Localization SWC files for proper localization support.
- Enhancements that ensure Flex applications function properly within a LiveCycle ES2 environment.

Install the LiveCycle ES2 version of the Flex SDK

- 1 Exit Flex Builder if you have it open.
- 2 On the LiveCycle ES2 DVD, navigate to the `additional\flex_sdk` folder and copy the `flex_sdk_3.zip` to your computer.
- 3 Navigate to the `sdk` folder located in the root folder of your installation of Flex Builder 3. For example, navigate to the `C:\Program Files\Adobe\Flex Builder 3\sdk` folder.
- 4 Create a folder, such as `3.4.0.lc`.
- 5 Using an archival and extraction tool, extract the `flex_sdk_3.zip` file to the folder created in the previous step.

After completing the task, the folder and its contents looks similar to the following illustration:



Configure Flex Builder to use the LiveCycle ES2 version of the Flex SDK

To develop Flex applications for LiveCycle ES2, set the default Flex compiler to use the LiveCycle ES2 version of the Flex SDK. Alternatively, configure specific projects when it is necessary for you to use multiple Flex SDK versions. (See [“Configure an individual Flex project to use LiveCycle ES2 version of the Flex SDK” on page 10.](#))

- 1 In Flex Builder, select **Window > Preferences**.
- 2 In the Preferences dialog box, select **Flex > Installed Flex SDKs**, and then click **Add**.
- 3 In the Add Flex SDK dialog box, perform the following steps:
 - Click **Browse**.
 - In the Browse For Folder dialog box, select the folder you created in step 3 in procedure [“Install the LiveCycle ES2 version of the Flex SDK” on page 8](#), and then click **OK**.
 - In the **Flex SDK name** box, type a name to reflect the specific Flex SDK version. For example, `Flex 3.4.0 - LCES2`.
 - Click **OK**.
- 4 Select the check box beside the Flex SDK version that was added in the previous step and click **Apply**.

Configuring Flex projects to use the Workspace API

To configure your Flex project to use the Workspace API, include the `foundation-runtime.swc`, `procmgmt-api.swc`, and `procmgmt.ui.swc` files in the build path for a Flex project. These SWC files are located in the LiveCycle ES2 SDK folder.

Note: Use the `-library-path` command-line options to compile your project to include the Workspace API SWC files. (See the *Flex 3 Developer Guide* at <http://livedocs.adobe.com/flex/3/html/help.html>.)

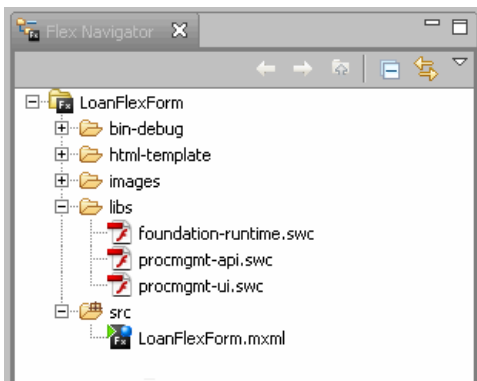
- 1 Select **File > New > Flex Project**.
- 2 In the Create a Flex project dialog box, in the **Project name** box, type a name, such as `LoanFlexForm`.
- 3 (Optional) If you want to select an alternative location to store the Flex project, deselect the **Use default location** check box, click **Browse**, and then navigate to the desired location.
- 4 Click **Next**.
- 5 (Optional) If you want to select an alternative Output folder, click **Browse**, and navigate to the desired folder.
- 6 Click **Next**.
- 7 Click **Finish**.
- 8 Navigate to the `[LC_SDK]/misc/Process_Management/Workspace/libs` folder, where `[LC_SDK]` is the location of the LiveCycle ES2 SDK folder. Copy the `foundation-runtime.swc`, `procmgmt-api.swc`, and `procmgmt-ui.swc` files to the `libs` folder in the Flex project.

You can access the required SWC files from the LiveCycle ES2 SDK folder at one of these locations:

- On a computer where LiveCycle Workbench ES2 is installed, navigate to `[installdir]/Adobe/Adobe LiveCycle Workbench ES2/LiveCycle_ES_SDK` folder where `[installdir]` represents the location where Workbench ES2 is installed. For example, navigate to the `C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK` on your computer.
 - On the LiveCycle ES2 server, navigate to the `[installdir]/LiveCycle_ES_SDK` folder where `[installdir]` represents where LiveCycle ES2 is installed. For example, using a JBoss® Turnkey installation, navigate to the `C:\Adobe\Adobe LiveCycle ES2\LiveCycle_ES_SDK` folder on the LiveCycle ES2 server.
- 9 Verify that your Flex project uses the LiveCycle ES2 version of the Flex SDK by completing these steps:
 - Right-click the Flex project and select **Properties**.
 - In the Properties for `[flex project name]` dialog box, click **Flex Compiler**.
 - In the Flex SDK version pane, verify that the project uses the LiveCycle ES2 version of the Flex SDK. For example, `Flex 3.4.0 - LCES2`. (See “[Configuring Your Development Environment for LiveCycle ES2](#)” on page 8.)

Caution: In Flex Builder, the SWC files are included each time the project is compiled. When you compile using the Flex SDK command-line compiler, the `foundation-runtime.swc`, `procmgmt-api.swc`, and `procmgmt-ui.swc` files must be included in the compile options each time. (See “[Using mxmxml, the application compiler](#)” in *Flex 3 Developer Guide* at <http://livedocs.adobe.com/flex/3/html/help.html>.)

After you complete the steps, the Flex project looks similar to the following illustration:



Creating Flex Applications for Data Capture

You can create Flex applications for data capture by following these steps:

- Create a form in Flex by using the `mx:Form` and `mx:FormItem` components. (See “Create a Flex application as forms” on page 13.)
- Add a data model and bind it to the fields in the form. (See “Add a data model to a Flex application” on page 15.)
- Validate the data stored in the Flex application. (See “Add validation to a form” on page 16.)

If you complete the steps as described, the Flex application looks like the following illustration:

The screenshot shows a web form titled "Preliminary Loan Approval Form" for "Fin@nce corp.". Below the title is the instruction "Fill the form to start the approval process." There are four input fields arranged vertically, each with a red asterisk indicating a required field:

- Name: John Jacobs
- Loan Amount: (empty)
- Phone or Email: jjacobs@sampleorganization.com
- Approval Status: PENDING APPROVAL

The example creates a simple form in Flex that has these items:

- A Fin@nce Corporation logo in the upper-left corner of the form. You get the `financeCorpLogo.jpg` from the Creating Your First LiveCycle ES2 Application ZIP file at http://www.adobe.com/go/learn_lc_firstApplicationZip_9. Alternatively, you can choose an image that meets your requirements.
- A title of Preliminary Loan Approval Form.
- Instructions for completing the task.
- The Name, Loan Amount, Phone Number, or Email fields arranged vertically.
- An Approval Status field that is not editable.
- A data model bound to the objects that present and capture user information. A data model is necessary to pass form data from the Flex application to LiveCycle Workspace ES2. Typically, data is passed to the human-centric process.
- Data validation, although optional, is recommended. Validate the form data before you pass it to Workspace ES2 (and ultimately to a human-centric 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 of an error during the execution of your business process.

Create a Flex application as forms

To create a Flex application that provides form-like behavior, use the Flex components `mx:Form` and `mx:FormItem` to build a form layout for a preliminary loan approval form. In addition, you can use the following components to enhance the form's functionality:

- `mx:Image`: For adding a company logo to the Flex application.

- `mx:RadioButton`: For adding a list of options that users can select from.
 - `mx:Button`: For adding a button that performs calculations based on the input of the user.
- 1 In Flex Builder, drag the `financeCorpLogo.jpg` image to the `src` folder in the Flex project.
 - 2 Add a `mx:VBox` component within the `<mx:Application>` tags.
 - 3 In the `<mx:VBox>` tag, add the `background` property and set it to `white`.
 - 4 Add a `mx:HBox` component after the opening `<mx:VBox>` tag. Within the `<mx:HBox>` tags, add and configure these components:
 - A `mx:Image` component for the logo and set these properties:
 - `source` to `@Embed('financeCorpLogo.jpg')`
 - `scaleContent` to `true`
 - `width` to `100`
 - `height` to `50`
 - A `mx:Label` component for the form title and set these properties:
 - `text` to `Preliminary Loan Approval Form`
 - `fontSize` to `28`
 - `color` to `Black`
 - `fontWeight` to `bold`
 - `fontFamily` to `Arial`
 - 5 Add another `mx:HBox` component after the closing `</mx:HBox>` tag from previous step.
 - 6 Within the `<mx:HBox>` tag, add a `mx:Label` component and set these properties:
 - `id` to `textinstructions`
 - `fontSize` to `10`
 - `text` to `Fill form`
 - 7 Add another `<mx:HBox>` component after the previous closing `</mx:HBox>` tag.
 - 8 After the opening `<mx:HBox>` tag, add a `mx:Form` component.
 - 9 Within the `<mx:Form>` tag, add the `backgroundColor` property and set it to `white`.
 - 10 Add four `mx:FormItem` components within the `<mx:Form>` tags. Nest each `<mx:FormItem>` tag within the `<mx:Form>` tag. Configure the `mx:FormItem` components as follows:
 - Set the `label` property for the `mx:FormItem` tags as follows:
 - First `mx:FormItem` tag: set the `label` property to `Name` and the `required` property to `true`.
 - Second `mx:FormItem` tag: set the `label` property to `Loan Amount` and the `required` property to `true`.
 - Third `mx:FormItem` tag: set the `label` property to `Phone or Email` and the `required` property to `true`.
 - Fourth `mx:FormItem` tag: set the `label` property to `Approval Status` and the `required` property to `false`.
 - Add a `mx:TextInput` component within each of the `<mx:FormItem>` tags and configure them as follows:
 - First `<mx:TextInput>` tag: add the `id` property and set it to `firstLastName`.
 - Second `<mx:TextInput>` tag: add the `id` property and set it to `loanAmount`.
 - Third `<mx:TextInput>` tag: add the `id` property and set it to `phoneOrEmail`.
 - Fourth `<mx:TextInput>` tag: add the `id` and `editable` properties and set them to `approvalStatus` and `false`, respectively.

Code for creating a Flex application as a form

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    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="Preliminary Loan Approval Form" fontSize="28"
      color="black" fontWeight="bold" fontFamily="Arial"/>
  </mx:HBox>
  <mx:HBox>
    <mx:Label id="taskinstructions" fontSize="10" text="" />
  </mx:HBox>
  <mx:HBox>
    <mx:Form backgroundColor="white">
      <mx:FormItem label="Name" required="true">
        <mx:TextInput id="firstName"/>
      </mx:FormItem>
      <mx:FormItem label="Loan Amount" required="true">
        <mx:TextInput id="loanAmount"/>
      </mx:FormItem>
      <mx:FormItem label="Phone or Email" required="true">
        <mx:TextInput id="phoneOrEmail"/>
      </mx:FormItem>
      <mx:FormItem label="Approval Status" required="false">
        <mx:TextInput id="approvalStatus" editable="false"/>
      </mx:FormItem>
    </mx:Form>
  </mx:HBox>
</mx:VBox>
</mx:Application>

```

For more information about creating flex applications that have form-like behavior, see “[Form, FormHeading, and FormItem layout containers](http://livedocs.adobe.com/flex/3/html/help.html)” in *Flex 3 Developer Guide* at <http://livedocs.adobe.com/flex/3/html/help.html>.

Add a data model to a Flex application

In the Flex application, it is a recommended practice to persist data using a component such as `mx:XML`. Form data passed from the human-centric process can be extracted and stored in a model. The model is bound to fields in the Flex application so that the data model values are updated when the user makes changes. When the user is ready to submit the form, the entire contents from the data model can be submitted to the human-centric process as an XML value. (See “[Defining a data model](http://livedocs.adobe.com/flex/3/html/help.html)” in *Flex 3 Developer Guide* at <http://livedocs.adobe.com/flex/3/html/help.html>.)

When you want to create simple data models, use the `<mx:XML>` tags. However, if you plan on creating more complex data models, consider using ActionScript to create the data model using the `mx:XML` class (Top Level class).

- 1 Add a `mx:XML` component and assign a unique name using the `id` property. For example, `myData`.
- 2 Within the `<mx:XML>` tags, define an XML schema to store the data model, and then bind the values from the `<mx:TextInput>` tags.
- 3 For each of the `<mx:TextInput>` tags within the `<mx:FormItem>` tags, complete these steps:
 - Set the `text` property to a value that represents the XML element defined in the `mx:XML` component.
 - Set the `change` property to update the data in the data model each time the field is updated.

Code for creating a two-way binding between the defined data model and each field in the Flex application

```

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <!-- Define the model -->
  <mx:XML id="myData">

```

```

    <LoanApp>
      <Name>{firstLastName.text}</Name>
      <LoanAmount>{loanAmount.text}</LoanAmount>
      <PhoneOrEmail>{phoneOrEmail.text}</PhoneOrEmail>
      <ApprovalStatus>{approvalStatus.text}</ApprovalStatus>
    </LoanApp>
  </mx:XML>

<!-- Create the form. -->
<mx:VBox backgroundColor="white">
  <mx:HBox>
    <mx:Image id="banner"
      source="@Embed('../images/financeCorpLogo.jpg') "
      scaleContent="true" width="100" height="50"/>
    <mx:Label text="Preliminary Loan Approval Form" fontSize="28"
      color="black" fontWeight="bold" fontFamily="Arial"/>
  </mx:HBox>
  <mx:HBox>
    <mx:Label id="taskinstructions" fontSize="10" text="" />
  </mx:HBox>
  <mx:HBox>
    <mx:Form backgroundColor="white">
      <mx:FormItem label="Name" required="true">
        <mx:TextInput id="firstLastName" text="{myData.Name}"
          change="myData.Name = firstLastName.text;"/>
      </mx:FormItem>
      <mx:FormItem label="Loan Amount" required="true">
        <mx:TextInput id="loanAmount" text="{myData.LoanAmount}"
          change="myData.LoanAmount = loanAmount.text;"/>
      </mx:FormItem>
      <mx:FormItem label="Phone or Email" required="true">
        <mx:TextInput id="phoneOrEmail" text="{myData.PhoneOrEmail}"
          change="myData.PhoneOrEmail = phoneOrEmail.text;"/>
      </mx:FormItem>
      <mx:FormItem label="Approval Status" required="false">
        <mx:TextInput id="approvalStatus" text="{myData.ApprovalStatus}"
          change="myData.ApprovalStatus = approvalStatus.text;"
          editable="false"/>
      </mx:FormItem>
    </mx:Form> </mx:HBox>
  </mx:VBox>

</mx:Application>

```

Add validation to a form

To add validation, add a component from the `mx:validator` package to the Flex application. (See “[Validating Data](#)” in *Flex 3 Developer Guide* at <http://livedocs.adobe.com/flex/3/html/help.html>.) Validation components are in the `mx.validators` package. Different properties are used to perform the validation of data depending on which component used.

Validation helps to ensure that the data is correct before the data is submitted to a human-centric process. It is a recommended practice to add validation to the Flex application to ensure that the information provided is correct.

- 1 Identify the component to which to add validation. You can validate string values, numeric values, or phone numbers using the `mx:StringValidator`, `mx:NumberValidator`, or `mx:PhoneNumberValidator` components respectively.

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

Code for verifying that the loan amount is greater than zero

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <!-- Define the model -->
    <mx:XML id="myData">
        <LoanApp>
            <Name>{firstName.text}</Name>
            <LoanAmount>{loanAmount.text}</LoanAmount>
            <PhoneOrEmail>{phoneOrEmail.text}</PhoneOrEmail>
            <ApprovalStatus>{approvalStatus.text}</ApprovalStatus>
        </LoanApp>
    </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="Preliminary Loan Approval Form" fontSize="28"
                color="black" fontWeight="bold" fontFamily="Arial"/>
        </mx:HBox>
        <mx:HBox>
            <mx:Label id="taskinstructions" fontSize="10" text="Fill form." />
        </mx:HBox>
        <mx:HBox>
            <mx:Form backgroundColor="white">
                <mx:FormItem label="Name" required="true">
                    <mx:TextInput id="firstName" text="{myData.Name}"
                        change="myData.Name = firstName.text;"/>
                </mx:FormItem>
                ...
            </mx:Form>
        </mx:HBox>
    </mx:VBox>

    <!-- The validator for the field that stores the loan amount -->
    <mx:NumberValidator id="loanValidator"
        source="{loanAmount}"
        property="text"
        required="true"
        minValue="1"
        lowerThanMinError="Loan amount must be greater than 0."
        domain="real"
        listener="loanAmount"/>
</mx:Application>
```

Enabling Flex Applications for Workspace ES2

To enable Flex applications for LiveCycle Workspace ES2, code is added to send and handle events to facilitate communication between the Flex application and Workspace ES2. This communication enables users to submit data in a manner similar to PDF or HTML forms. To understand the required and optional events you can use, knowledge of how to send and handle events is important for understanding how to enable Flex applications for Workspace ES2. (See [“Understanding Workspace API events and event handling”](#) on page 18.)

Complete these tasks to enable a Flex application for Workspace ES2:

- Add the `lc:SwfConnector` component. (See [“Add the SwfConnector component”](#) on page 22.)
- Dispatch events to indicate whether data is valid. (See [“Validate data and send events in the Flex applications”](#) on page 23.)
- Dispatch events to indicate that data has changed. (See [“Indicate changes to the form data stored in Flex applications”](#) on page 24.)
- Create event listeners to handle events dispatched by Workspace ES2. (See [“Handle events from Workspace ES2”](#) on page 25.)

Understanding Workspace API events and event handling

Using events and event handling are necessary to enable a Flex application for Workspace ES2 for these reasons:

- Notify Workspace ES2 that the Flex application is ready to start receiving events and communicating.
- Exchange form data between the Flex application and Workspace ES2.
- Respond to button clicks that occur within Workspace ES2 to save or submit form data.
- Notify Workspace ES2 that data or the validity of the data has changed, so that Workspace ES2 can enable the Save and Complete buttons.

You can use additional events for the following purposes:

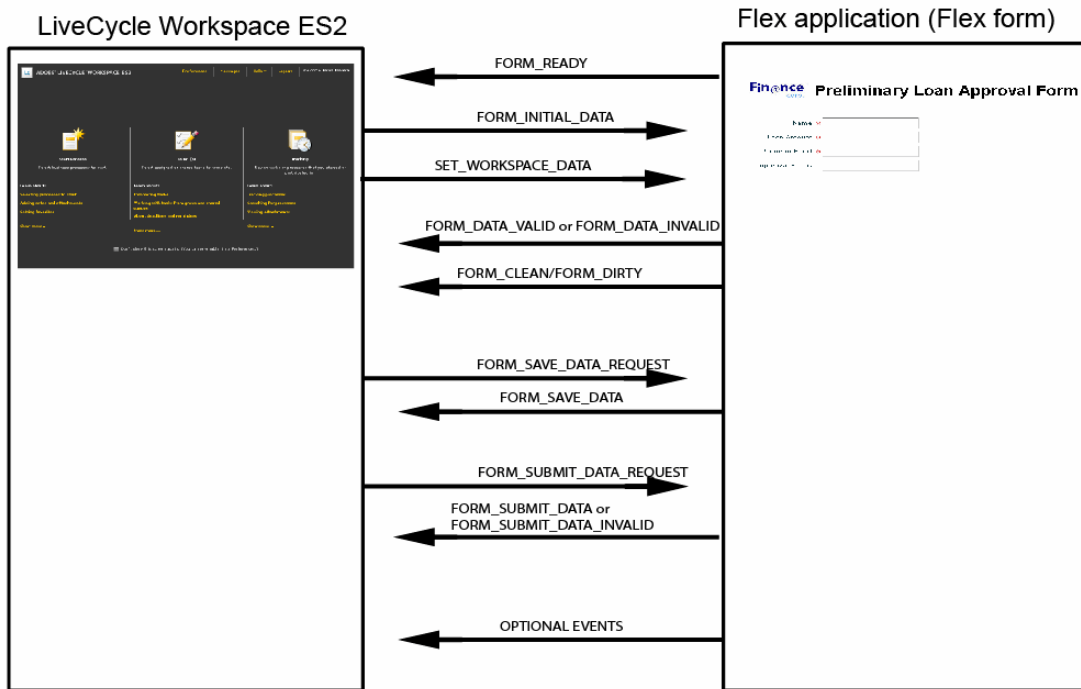
- Control the Workspace ES2 user interface. For example, hide or show tabs related to the task, or maximize the Flex application in the Workspace ES2 user interface window.
- Retrieve information about the user who is logged in to Workspace ES2.
- Retrieve task-specific information such as instructions.

For information about how events are passed between the Flex application and Workspace ES2, see [“Events sent between Workspace ES2 and a Flex application”](#) on page 19.

You can send and handle events using Flex event-handling APIs; however, it is a recommended practice to use the `lc:SwfConnector` component from the Workspace API to send events. For information about sending and handling events, see [“Events sent to Workspace ES2”](#) on page 21 and [“Event handling using the Workspace API”](#) on page 21.

Events sent between Workspace ES2 and a Flex application

The following illustration shows the events that are sent between a Flex application and Workspace ES2. Arrows indicate whether the event is sent to Workspace ES2 or to the Flex application.



The following table describes the events in the previous illustration:

Events	Description
FORM_READY	The event notifies Workspace ES2 that the Flex application is ready to start receiving events. After the Flex application has loaded, it must send a <code>FORM_READY</code> event to Workspace ES2.
FORM_INITIAL_DATA	The event provides form data from the human-centric process to the Flex application. After receiving the <code>FORM_READY</code> event, Workspace ES2 dispatches a <code>FORM_INITIAL_EVENT</code> event. The Flex application can use the data to populate field values. The data that is loaded into a Flex application is retrieved during the execution of an operation from the User service. The data can also be configured when a Workspace start point is used. (See “Working with captured data” in Application Development Using LiveCycle Workbench ES2 at http://www.adobe.com/go/learn_lc_workbench_9 .)
SET_WORKSPACE_DATA	The event provides data about the task and user who is logged in to Workspace ES2 to the Flex application. Optionally, use task and user-specific information to populate the Flex application. For example, you can retrieve the name of the user logged in to Workspace ES2 and task instructions to provide a personalized description of how to complete the task.
FORM_DATA_VALID and FORM_DATA_INVALID	The events notify Workspace ES2 to enable the Complete button or route buttons in Workspace ES2. The Flex application must send the <code>FORM_DATA_VALID</code> or <code>FORM_DATA_INVALID</code> events after validating the data. To reduce the number of events sent, optimization can be achieved by sending the <code>FORM_DATA_INVALID</code> event in the event listener for the <code>FORM_INITIAL_DATA</code> event handler. In this situation, the Flex application validates the data upon receipt of the data from Workspace ES2.

Events	Description
FORM_CLEAN and FORM_DIRTY	<p>The events notify Workspace ES2 whether changes have occurred to the form data to enable the Save button. In Workspace ES2, the user is always prompted to save the form when they leave a task. The prompting occurs because the form is assumed to be in a dirty status. A <i>dirty</i> status means that the data has changed since the last time it was saved. A <i>clean</i> status means that the data has not changed since the last time it was saved. A Flex application can track the status of the data by sending a FORM_CLEAN or FORM_DIRTY events to Workspace ES2.</p> <p>It is a recommended practice that the Flex application dispatch a FORM_CLEAN event after the FORM_INITIAL_DATA and FORM_SAVE_DATA events. When Workspace ES2 receives a FORM_DIRTY event, the Save button is enabled.</p>
FORM_SAVE_DATA_REQUEST and FORM_SAVE_DATA	<p>The events indicate that the user wants to save data in Workspace ES2. Workspace ES2 dispatches a FORM_DATA_SAVE_REQUEST event when the user clicks the Save button. In response, the Flex application sends the FORM_SAVE_DATA event with the form data specified in XML format.</p>
FORM_SUBMIT_DATA_REQUEST, FORM_SUBMIT_DATA, and FORM_SUBMIT_DATA_INVALID	<p>The events indicate that the user wants to complete the task by completing the task. Workspace ES2 sends a FORM_SUBMIT_DATA_REQUEST event when the user chooses to submit the form data. In response to the FORM_SUBMIT_DATA_REQUEST event, the Flex application must dispatch either a FORM_SUBMIT_DATA event with the valid form data or a FORM_SUBMIT_DATA_INVALID event.</p> <p>Workspace ES2 uploads the form data from a Flex application after a FORM_SUBMIT_DATA event is received in response to a FORM_SUBMIT_DATA_REQUEST event.</p>
OPTIONAL EVENTS	<p>After sending the FORM_READY event, optional events can control the Workspace ES2 user interface. For example, it is often useful to maximize the Flex application in full-screen mode, show or hide tabs and buttons in the Workspace ES2 user interface. For a complete list of the events and their descriptions, see "SwfConnector" in <i>LiveCycle ES2 ActionScript Language Reference</i> at http://www.adobe.com/go/learn_lc_actionScript_9. The following are examples of events that are commonly used:</p> <ul style="list-style-type: none"> • FULL_SCREEN • CHANGE_ROUTE_BUTTON_TOOLTIP • DISABLE_ROUTE_BUTTON • HIDE_ATTACHMENTS_VIEW • SHOW_ATTACHMENTS_VIEW • MINIMIZE_SCREEN • SHOW_TASK_DETAILS_VIEW • SHOW_TASK_FORM_VIEW

Note: Use the events defined in the *FormEvents*, *SwfAppButtonEvent*, *SwfAppEvent*, and *SwfDataEvent* classes (*lc.procmgmt.events* package) when using the Flex event-handling API to communicate with Workspace ES2. (See "[FormEvents](#)", "[SwfAppButtonEvent](#)", "[SwfAppEvent](#)", and "[SwfDataEvent](#)" in *LiveCycle ES2 ActionScript Language Reference* at http://www.adobe.com/go/learn_lc_actionScript_9.)

Events sent to Workspace ES2

You can use the `lc:SwfConnector` component to send events to Workspace ES2. The following table outlines the most commonly used methods:

lc:SwfConnector Method	Description
<code>setReady</code>	Sends the <code>FORM_READY</code> event, which notifies Workspace ES2 that the Flex application has loaded and is ready to start communication. No communication occurs until the Flex application sends the <code>FORM_READY</code> event.
<code>setDataValid</code>	Sends a <code>FORM_DATA_INVALID</code> event to notify Workspace ES2 that the form data is valid. The event must be dispatched to Workspace ES2 for the Complete button to be available.
<code>setDataInvalid</code>	Sends a <code>FORM_DATA_INVALID</code> event to notify Workspace ES2 that the form data is not valid.
<code>setClean</code>	Sends the <code>FORM_CLEAN</code> event that notifies Workspace ES2 that no changes have occurred since the last time the form was saved.
<code>setDirty</code>	Sends the <code>FORM_DIRTY</code> event that notifies Workspace ES2 that changes have occurred, which enables the Save button in Workspace ES2.
<code>setSaveData</code>	Sends the <code>FORM_SAVE_DATA</code> event and passes the form data stored as an XML from the Flex application to Workspace ES2.
<code>setSubmitData</code>	Sends the <code>FORM_SUBMIT_DATA</code> event and passes the form data stored as XML from the Flex application to complete the task or start a process in Workspace ES2.

Event handling using the Workspace API

The `lc:SwfConnector` component provides properties to handle events sent from Workspace ES2. The following table describes the `lc:SwfConnector` properties to bind functions you create to handle events sent from Workspace ES2:

lc:SwfConnector Property	Description
<code>formInitialData</code>	<p>Workspace ES2 sent the <code>FORM_INITIAL_DATA</code> event, which contains initialization data. As soon as Workspace ES2 receives the <code>FORM_READY</code> event, it sends initialization data to the Flex application.</p> <p>The event listener can use the initialization data provided from Workspace ES2 to pre-fill data stored by the Flex application. Pre-filling data is useful for pre-filling fields displayed in the Flex application. The initialization data is configured during the design of an automated process created within Workbench ES2. It is a recommended practice to send the <code>FORM_CLEAN</code> event to indicate that a save is not required. It is also recommended that you send <code>FORM_DATA_VALID</code> or <code>FORM_DATA_INVALID</code> event when you validate the data.</p> <p>The data must use a data schema that the Flex application understands. It is a recommended practice that the data model used by your Flex application and the human-centric process is the same. (See “Working with captured data” in <i>Application Development Using LiveCycle Workbench ES2</i> at http://www.adobe.com/go/learn_lc_workbench_9.)</p>
<code>formSaveDataRequest:</code>	<p>Workspace ES2 sent a <code>FORM_SAVE_DATA_REQUEST</code> event because the user clicked the Save button.</p> <p>The event listener created must send the <code>FORM_DATA_SAVE</code> event with the form data formatted as XML. It is a recommended practice to send the <code>FORM_CLEAN</code> event so that the Save button in Workspace ES2 is disabled. There is no requirement to ensure that the data is valid because it can represent in-progress form data.</p>
<code>formSubmitDataRequest</code>	<p>Workspace ES2 sent a <code>FORM_SUBMIT_DATA_REQUEST</code> because the user clicked the Complete button or one of the route buttons.</p> <p>The event listener created must send the <code>FORM_SUBMIT_DATA</code> event with the form data formatted as XML. It is a recommended practice that the data is validated before it is sent. Before sending the <code>FORM_SUBMIT_DATA</code> event, send the <code>FORM_DATA_VALID</code> event to indicate that the data is valid. If it fails validation, send the <code>FORM_SUBMIT_DATA_INVALID</code> event to indicate that the data is not valid and is not sent.</p>
<code>setWorkspaceData</code>	<p>Workspace ES2 sent a <code>SET_WORKSPACE_DATA</code> event with data. The data contains information about the task and user logged in to Workspace ES2.</p> <p>The optional event listener created can retrieve information about the task or user that is logged in to Workspace ES2. For example, the task instructions and name of the user can be retrieved to build a more personable message in the Flex application, such as “Good day John Jacobs, please fill in the form to complete the task”.</p>

Add the SwfConnector component

The `lc:SwfConnector` component is required to enable an application for Workspace ES2. The `lc:SwfConnector` component has methods and properties to send and handle events between a Flex application and Workspace ES2. (See “[SwfConnector](#)” in *LiveCycle ES2 ActionScript Language Reference* at http://www.adobe.com/go/learn_lc_actionScript_9.)

- 1 Add a new namespace URI and prefix to the `<mx:Application>` tag for accessing the Workspace API components. For example, add `xmlns:lc="http://www.adobe.com/2006/livecycle"`.
- 2 Add an instance of the `lc:SwfConnector` component and assign a name to the `id` property. For example, `lcConnector`.
- 3 Add a `mx:Script` component if one does not exist.
- 4 Add a `creationComplete` property to the `mx:Application` component and bind it to a function. In the function, use the `lc:SwfConnector.setReady()` method to notify Workspace ES2 that the Flex application is ready to receive events.

Code for adding a SwfConnector object

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

<mx:Script>
<![CDATA[
    /*****
    * Indicates to Workspace ES2 that the Flex application is
    * ready to start receiving events.
    *****/
    private function initApp():void
    {
        lcConnector.setReady();
    }
    ]]>
</mx:Script>

<!-- Enable communication with Workspace ES2 -->
<lc:SwfConnector id="lcConnector"/>

<!-- Define the model -->
<mx:XML id="myData">
    <LoanApp>
        <Name>{firstName.text}</Name>
        <LoanAmount>{loanAmount.text}</LoanAmount>
        <PhoneOrEmail>{phoneOrEmail.text}</PhoneOrEmail>
        <ApprovalStatus>{approvalStatus.text}</ApprovalStatus>
    </LoanApp>
</mx:XML>
...
...
</mx:Application>
```

Validate data and send events in the Flex applications

When the Flex application validates data, use the `lc:SwfConnector` component to indicate the validity of the data to Workspace ES2. Use components from the `mx.validators` package to verify the validity of fields values. Then, use the `setDataValid` and `setDataInvalid` methods from the `lc:SwfConnector` component to indicate the validity of form data in the Flex application. (See “Events sent to Workspace ES2” on page 21.)

In Workspace ES2, the Complete button and route buttons are necessary for users to complete tasks. For Workspace ES2 to enable the Complete button or route buttons, the `formDataValid` event must be sent from the Flex application. When the Flex application validates more than one value, it is a recommended practice to send one event to represent the validity of all the values.



Verify the validity of the data in the event listener used to handle the `FORM_SUBMIT_DATA_REQUEST` to validate multiple pieces of information. Ensure any data sent is valid and formatted correctly.

- 1 Add the `valid` property to the `<mx:NumberValidator>` tag (or `mx.validator` component being used in the Flex application), and set it to `lc:SwfConnector.setDataValid` method.
- 2 Add the `invalid` property to the `<mx:NumberValidator>` tag (or another `mx.validator` component) and bind it to the `lc:SwfConnector.setDataInvalid` method.

Code for dispatching a `FORM_DATA_INVALID` and `FORM_DATA_INVALID` events from a `mx.validator` component

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

<mx:Script>
<![CDATA[
  ...
  ...
  <!-- Enable communication with Workspace ES2 -->
  <lc:SwfConnector 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()"/>

<!-- The validator for the fields that store the loan amount -->
<mx:NumberValidator id="loanValidator"
  source="{loanAmount}"
  property="text"
  required="true"
  minValue="1"
  lowerThanMinError="Loan amount must be greater than 0."
  domain="real"
  valid="lcConnector.setDataValid();"
  invalid="lcConnector.setDataInvalid();"
  listener="loanAmount"/>
```

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

Indicate changes to the form data stored in Flex applications

When a user leaves a form associated with a task, Workspace ES2 prompts the user to save the form data regardless of whether changes were made. To prompt the user to save the form when form data changes, the Flex application must send events to notify Workspace ES2 that the data changed. It is a recommended practice that after the Flex application receives the `FORM_INITIAL_DATA` event, the Flex application sends the `FORM_CLEAN` event to Workspace ES2. In addition, whenever changes occur to the data, the Flex applications sends the `FORM_DIRTY` event to Workspace ES2.

Use the `setDirty` and `setClean` methods from the `lc:SwfConnector` component to indicate whether the data stored in the Flex application has changed. (See “Events sent to Workspace ES2” on page 21.)

Note: When the `formDirty` or `formClean` events are not used in the Flex application, users are always prompted to click Save. The message appears regardless of whether changes occurred to the data stored in the Flex application.

- 1 In your Flex component, locate each component where data is entered or changed. For most components, when a field changes its value, a change event is available.
- 2 Bind the `change` event to the `lc:SwfConnector.setDirty()` method. For example, for the `<mx:TextInput>` tag, bind the `change` property to the `lc.SwfConnector.setDirty()` method.

Code for sending the `formDirty` event that is bound to the `change` event for each `<mx:TextInput>` tag

```
<?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"
  creationComplete="initApp();"
  layout="absolute">

  <mx:Script>
  <![CDATA[
  ...
  ...

  <!-- 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="Preliminary Loan Approval Form" fontSize="28"
        color="black" fontWeight="bold" fontFamily="Arial"/>
    </mx:HBox>
    <mx:HBox>
      <mx:Label id="taskinstructions" fontSize="10" text="Fill form." />
    </mx:HBox>
    <mx:HBox>
      <mx:Form backgroundColor="white">
        <mx:FormItem label="Name" required="true">
          <mx:TextInput id="firstName" text="{myData.Name}"
            change="myData.Name = firstName.text;
              lcConnector.setDirty();"/>
        </mx:FormItem>
        <mx:FormItem label="Loan Amount" required="true">
          <mx:TextInput id="loanAmount" text="{myData.LoanAmount}"
            change="myData.LoanAmount = loanAmount.text;
              lcConnector.setDirty();"/>
        </mx:FormItem>
      </mx:Form>
    </mx:HBox>
  </mx:VBox>
  </mx:Script>
</mx:Application>
```

```

    </mx:FormItem>
    <mx:FormItem label="Phone or Email" required="true">
        <mx:TextInput id="phoneOrEmail" text="{myData.PhoneOrEmail}"
            change="myData.PhoneOrEmail = phoneOrEmail.text;
                lcConnector.setDirty();" />
    </mx:FormItem>
    <mx:FormItem label="Approval Status" required="false">
        <mx:TextInput id="approvalStatus" text="{myData.ApprovalStatus}"
            change="myData.ApprovalStatus = approvalStatus.text;"
            editable="false" />
    </mx:FormItem>
</mx:Form>
</mx:HBOX>
</mx:VBOX>
...
...
</mx:Application>

```

Handle events from Workspace ES2

Events and event listeners facilitate communication between a Flex application and Workspace ES2. Creating functions for event listeners is a significant part of enabling a Flex application for Workspace ES2. Event listeners respond to events sent from Workspace ES2. Specifically, provide functions to handle when the `FORM_INITIAL_DATA`, `FORM_SAVE_DATA_REQUEST`, and `FORM_SUBMIT_DATA_REQUEST` events are received by the Flex application. It is recommended that you bind custom functions to properties in the `lc:SwfConnector` component. (See “Event handling using the Workspace API” on page 21.)

You can use the following `lc:SwfConnector` methods to respond to various events from Workspace ES2: (See “Events sent to Workspace ES2” on page 21.)

- `setClean`
- `setDirty`
- `setFormValid`
- `setFormInvalid`
- `setSubmitData`

You can also control the Workspace ES2 user interface from the Flex application. For example, you can use the `SwfConnector.setFullScreen()` method to maximize the Flex application when it loads in Workspace ES2. (See “SwfConnector” in *LiveCycle ES2 ActionScript Language Reference* at http://www.adobe.com/go/learn_lc_actionScript_9.)

- 1 In the `mx:Application` component, in the function bound to the `creationComplete` property, add an event to maximize the Flex application so it appears in full-screen mode.
- 2 In the function `initApp()` function, add the `lc:SwfConnector.setFullScreen()` method after sending the `lc:SwfConnector.setReady()` method.
- 3 To create an event listener function to bind to the `formInitialData` property in the `lc:SwfConnector` component, complete these steps:
 - Add a private function within the `<mx:Script>` tags. For example, name the function `handleInitDataListener`.
 - Configure the function to return `void` take a parameter of type `DataEvent`.
 - In the function, if the initial data is passed from the `DataEvent` object, extract the data as XML data and populate the fields in the Flex application. You retrieve the XML values based on how the data is formatted from Workspace ES2. 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 in the data model or add default information.
 - Send the `FORM_CLEAN` event by using the `lc:SwfConnector.setClean()` method.

- Validate any data when using components for form validation. (See “Add validation to a form” on page 16.)
 - In the `lc:SwfConnector` object, add the `formInitialData` property and set it to the name of function you created in step 3.
- 4** To create an event listener function to bind to the `formSaveDataRequest` property of the `lc:SwfConnector` component, complete these steps.
- Add a private function within the `<mx:Script>` tags. For example, name the function `handleFormSaveDataRequestListener`.
 - Configure the function to return `void` take a parameter of type `DataEvent`.
 - Send the `FORM_SAVE_DATA` event, with the data, calling the `lc:SwfConnector.setSaveData` method.
 - Send the `FORM_CLEAN` event by calling the `lc:FormConnector.setClean()` method.
 - In the `lc:SwfConnector` component, add the `formSaveDataRequest` property and set it to the name of the function you created in this step.
- 5** To create an event listener function to bind to the `formSubmitDataRequest` property in the `lc:SwfConnector` component, complete these steps:
- Add a private function within the `<mx:Script>` tags. For example, name the function `handleSubmitFormDataRequestListener`.
 - Configure the function to return `void` and take a parameter of type `DataEvent`.
 - Validate any data stored by the Flex application and provide the following functionality:
 - If the form data is valid, send a `FORM_DATA_VALID` event using the `lc:SwfConnector.setSubmitDataValid()` method. Then send XML data using the `lc:SwfConnector.setSubmitData()` method.
 - If the data is not valid, send a `FORM_DATA_INVALID` event by using the `lc:SwfConnector.setSubmitDataInvalid()` method.
 - In the `lc:SwfConnector` component, add the `formSubmitDataRequest` property and set it to the name of the event listener that you created in this step.
- 6** To create an event listener function to bind to the `setWorkspaceData` property of the `lc:SwfConnector` component, complete the following steps:
- Add a private function within the `<mx:Script>` tags. For example, name the function `handleWorkspaceData`.
 - Configure the function to return `void` and take a parameter of type `SwfDataEvent`.
 - Retrieve the `Task` object from the `SwfDataEvent.task` property. Save the `Task` object to the `SwfConnector.task` property.
 - Retrieve the task instructions to display in the Flex application using `SwfConnector.task.taskinstructions` property.
 - Retrieve the `User` object from the `SwfConnector.getAuthenticatedUser()` method.
 - If the `firstName` in the `mx:TextField` component is empty, copy the full name and email of the user logged in to Workspace ES2 using the `User.displayname` and `User.email` properties.

Code for handling events from Workspace ES2

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

<mx:Script>
<![CDATA[
import mx.events.ValidationResultEvent;
import mx.validators.ValidationResult;
import lc.foundation.domain.Message;
import lc.procmgmt.events.SwfDataEvent;
import lc.foundation.domain.User;
import mx.controls.Alert;

/*****
* Indicates to Workspace ES2 that the Flex application is
* ready to start receiving events.
*****/
private function initApp():void
```

```

{
    lcConnector.setReady();
    lcConnector.setFullScreen();
}

/*****
 * Handles when initialization data is sent.
 *****/
private function handleInitDataListener(event:DataEvent):void
{
    //Data is being received as XDP data.
    if (event.data != null && event.data != "") {
        var wsData:XML = new XML(event.data);
        firstLastName.text = wsData..Name ;
        loanAmount.text = wsData..LoanAmount;
        phoneOrEmail.text = wsData..PhoneOrEmail;
        approvalStatus.text = wsData..ApprovalStatus;
    }
    else {
        //Presuming when there is no form data to set default information.
        approvalStatus.text = "PENDING APPROVAL";
    }
    //Form data is provided and nothing has been modified yet.
    lcConnector.setClean();

    //Validate form data
    var validationLoanAmount:ValidationResultEvent = loanValidator.validate(null, true);
    if (validationLoanAmount.type == ValidationResultEvent.INVALID){
        lcConnector.setDataInvalid();
    }
}

/*****
 * Get the Workspace User and Task data sent from Workspace ES2.
 *****/
private function handleWorkspaceData(event:SwfDataEvent):void {
    //Get the task information
    lcConnector.task = event.task;

    //Add task instructions to the form.
    taskinstructions.text = lcConnector.task.instructions;

    //Retrieve information of the user logged in to Workspace ES2 if none is provided
    var currentUser:User = lcConnector.getAuthenticatedUser();
    if (firstLastName.text == ""){
        firstLastName.text = currentUser.displayName;
        phoneOrEmail.text = currentUser.email;
    }
}

/*****
 * Handles when Save button is clicked in Workspace ES2.
 *****/
private function handleFormSaveDataRequestListener(event:Event): void {
    // Do not check the data because it is only being saved.
    lcConnector.setSaveData(myData)
    lcConnector.setClean();
}

/*****1
 * Handles when Complete button is clicked in Workspace ES2.

```

```
*****/
private function HandleSubmitFormDataReaderListener(event:Event):void{
    // Validate the data before submitting it.
    var validationLoanAmount:ValidationResultEvent = loanValidator.validate(null, true);
    if (validationLoanAmount.type == ValidationResultEvent.VALID){
        lcConnector.setDataValid();
        lcConnector.setSubmitData(myData);
    }
    else {
        Alert.show("You must specify a loan amount greater than zero",
            "Attention", Alert.OK);
        lcConnector.setDataInvalid();
    }
}
]]>
</mx:Script>

<!-- Enable communication with Workspace ES2 and define methods for handling communication
with Workspace ES2. -->
<lc:SwfConnectorid="lcConnector"
    formInitialData="handleInitDataListener(event) "
    formSaveDataRequest="handleFormSaveDataRequestListener(event) "
    formSubmitDataRequest="handleSubmitFormDataReaderListener(event) "
    setWorkspaceData="handleWorkspaceData(event) "/>

<!-- Define the model
...
...
</mx:Application>
```

Deploying and Testing Flex Applications in Workspace ES2

Deploy and configure the Flex application enabled for LiveCycle Workspace ES2 using a LiveCycle ES2 application to test it. (See “[Enabling Flex Applications for Workspace ES2](#)” on page 18.) Testing of Flex applications within Workspace ES2 cannot occur in a web browser alone. It is necessary that you create a human-centric process using LiveCycle Workbench ES2 and configure it to use a SWF file. The SWF file is compiled from your Flex project and located in the output folder, such as bin-debug.

Consider testing the Flex application using the process from the [Creating Your First LiveCycle ES2 Application](#) tutorial at http://www.adobe.com/go/learn_lc_firstApplication_9. The steps in the following tasks use the LiveCycle ES2 application from the tutorial to deploy and test the Flex application.

Complete these tasks to deploy and test the Flex application:

- Deploy the Flex application to a LiveCycle ES2 application using Workbench ES2. (See “[Deploy Flex applications](#)” on page 29.)
- Test the Flex application in Workspace ES2. (See “[Test Flex applications in Workspace ES2](#)” on page 30.)

Deploy Flex applications


To deploy a Flex application, you must import it into LiveCycle ES2 and configure it in a human-centric process.

Import the Flex application to LiveCycle ES2

- 1 In Flex Builder, save your Flex project and build the Flex application. A SWF file appears in the **bin-debug** folder in the Flex project.
- 2 In Workbench ES2, import the SWF file to the LiveCycle ES2 application version, by dragging the SWF file from Windows Explorer to the Applications view. For example, drag the **LoanFlexApp.swf** file to the FirstApp > FirstApp/1.0 folder in the Applications view.
- 3 In Workbench ES2, right-click the SWF file you imported in the previous step, and select **Check In**. For example, right-click **LoanFlexForm.swf** and select **Check In**.

For more information about importing a SWF file into LiveCycle ES2. (See “[Working with assets](#)” in *Application Development Using LiveCycle Workbench ES2* at http://www.adobe.com/go/learn_lc_workbench_9.)

Configure the Flex application in a human-centric process

- 1 In Workbench ES2, create a simple human-centric process. To pass data from Workspace ES2 to the process, the input variable in the process must be an XML data type.
- 2 To enable the Flex application to start a process, edit the process of the imported SWF file by completing these steps:
 - In the Applications view, right-click the process and select **Check Out**. For example, right-click PreLoanProcess.
 - In the process diagram, select the Workspace start point. For example, in PreLoanProcess, select **Apply for preliminary loan**.
 - In the Process Properties view, in the Presentation and Data property group, click the **ellipsis**  button beside the **Asset** property.
 - In the Select Form Assets dialog box, select the SWF file, such as **LoanFlexForm.swf**, and click **OK**.
- 3 Save and deploy the LiveCycle ES2 application by completing these steps:
 - Select **File > Save** to save the process.
 - Right-click the process, and select **Check In**.

- Right-click the application version and select **Deploy**.

For more information about human-centric processes, see “[Designing human-centric processes](#)” in *Application Development Using LiveCycle Workbench ES2* at http://www.adobe.com/go/learn_lc_workbench_9.

Test Flex applications in Workspace ES2

Use Workspace ES2 to test the following functionality:

- In Workspace ES2, interaction with the Flex application is possible.
- Validation checks work properly when the data is submitted to Workspace ES2.
- The Save button is enabled only when changes occur to the data.
- The Flex application saves the data correctly as draft.
- You can fill the Flex application with data. The tasks to fill the Flex application are specific to its application logic and the requirements of the process.
- The Flex application submits the data correctly to complete the human-centric process.

For a list of common problems and suggested resolutions, see “[Troubleshooting](#)” on page 31.

- 1 Log in to Workspace ES2. For example, log in using `jjacobs` and `password` as the user ID and password. (See “Testing a Service Using LiveCycle Workspace ES2” in *Creating Your First LiveCycle ES2 Application* at http://www.adobe.com/go/learn_lc_firstApplication_9.)
- 2 Click **Start Process**.
- 3 Click the category and then click the card representing your process. For example, click **Loans** and click **Apply for preliminary loan**.
- 4 When your Flex application appears, validate that the application displays correctly. For example, the name, *John Jacobs* and, email, *jjacobs@sampleorganization* are displayed in the Flex application.
- 5 Provide the information to complete the form. For example, in the **Loan Amount** field, type a value of `100000`.
- 6 Verify that the Flex application can save the data by completing these steps:
 - Click **Save**.
 - Click **To Do**, click **Drafts**, and then click **Apply for preliminary loan**.
 - Verify that the Flex application contains the information you provided in step 5.
- 7 Click **Complete** and verify that the next operation in the process executes.



Use *Record and Playback* to verify that the next operation is executed. (See “[Recording and playing back process](#)” in *Application Development Using LiveCycle Workbench ES2* at http://www.adobe.com/go/learn_lc_workbench_9.)

For more information about testing human-centric processes, see “Testing a Service Using LiveCycle Workspace ES2” in *Creating Your First LiveCycle ES2 Application* at http://www.adobe.com/go/learn_lc_firstApplication_9.)

Troubleshooting

This table contains common problems and proposed solutions when you deploy and test a Flex application in Workspace ES2.

Problem	Proposed solution
Compile problem: SwfConnector not found or events not found.	Verify that you loaded the Workspace API SWC files in the Flex project. When compiling the command-line options in the Flex SDK, ensure that you included the SWC files in the compiler options.
Flex application does not pre-populate with information.	Verify that the event listener for the <code>FORM_INITIAL_DATA</code> event receives the data from Workspace ES2. In the event listener for the <code>FORM_INITIAL_DATA</code> event, verify the values being received are retrieved correctly from the <code>mx:XML</code> object.
Complete button does not become enabled in Workspace ES2.	Verify that you did not inadvertently send a <code>formDataInvalid</code> event.
The Flex application displays in Workspace ES2 but nothing happens when the Submit button or Save button is clicked.	Verify that the <code>FORM_READY</code> event has been sent to Workspace ES2 from your Flex application. For example, use the <code>lc:SwfConnector.setReady</code> method from the <code>creationComplete</code> property from the <code><mx:Application></code> tag. Verify that the data is valid and send the <code>FORM_DATA_VALID</code> event was sent when using a class from the <code>mx.validator</code> package.
Workspace ES2 cannot display your form.	Verify that Flex application is checked-in and deployed with the LiveCycle ES2 application.
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>lc:SwfConnector.setSubmitData</code> method is used to respond to a <code>FORM_SUBMIT_DATA_REQUEST</code> . In addition, verify that the form data is provided as an XML object.

Sample Code of Flex application enabled for Workspace ES2

The following example represents the content from the MXML file that is created from completing the tasks in this document. The example creates a Flex application that can be used to start the human-centric process created in the [Creating Your First LiveCycle ES2 Application](http://www.adobe.com/go/learn_lc_firstApplication_9) tutorial at http://www.adobe.com/go/learn_lc_firstApplication_9:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
* This is a sample Flex application (Flex form) that you can use with a human-centric process created in
* Adobe LiveCycle Workbench ES2. For more information on creating human-centric processes, see
* Application Developing Using LiveCycle Workbench ES2 at http://www.adobe.com/go/learn_lc_workbench_9.

* To build the the SWF file, use the LiveCycle ES2 version of the Flex SDK and import the foundation-runtime.swc,
* procmgmt-api.swc, and procmgmt-ui.swc files from the LiveCycle ES2 SDK folder.
*
* Copyright 2010 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="initApp();">

<!-- ActionScript for Event listener functions-->
<mx:Script>
    <![CDATA[
        import lc.foundation.domain.User;
        import lc.procmgmt.events.SwfDataEvent;
        import mx.events.ValidationResultEvent;
        import mx.validators.ValidationResult;
        import mx.controls.Alert;

        /*****
         * Indicates to Workspace ES2 that the Flex application is
         * ready to start receiving events.
         *****/
        private function initApp():void
        {
            lcConnector.setReady();
            lcConnector.setFullScreen();
        }
        /*****
         * Indicates to Workspace ES2 that the Flex application is
         * ready to start receiving events.
         *****/
        private function handleInitDataListener(event:DataEvent):void
        {
            if (event.data != null && event.data != "") {
                var wsData:XML = new XML(event.data);
                firstLastName.text = wsData..Name ;
            }
        }
    ]]>

```

```

        loanAmount.text = wsData..LoanAmount;
        phoneOrEmail.text = wsData..PhoneOrEmail;
        approvalStatus.text = wsData..ApprovalStatus;
    }
    else {
        //Presuming when there is no form data to set default information.
        approvalStatus.text = "PENDING APPROVAL";
        //Form data is provided and nothing has been modified yet.
        lcConnector.setClean();

        //Validate form data
        var validationLoanAmount:ValidationResultEvent = loanValidator.validate(null,false);
        if (validationLoanAmount.type == ValidationResultEvent.INVALID){
            lcConnector.setDataInvalid();
        }
    }
}
}
}
/*****
 * Handles when the Save button is clicked in Workspace ES2.
 *****/
private function handleFormSaveDataRequestListener(event:Event): void {
    lcConnector.setSaveData(myData)
    lcConnector.setClean();
}
/*****1
 * Handles when Complete button is clicked in Workspace ES2.
 *****/
public function handleSubmitFormDataRequestListener(event:Event):void{
    var validationLoanAmount:ValidationResultEvent = loanValidator.validate(null, true);
    if (validationLoanAmount.type == ValidationResultEvent.VALID){
        lcConnector.setDataValid();
        lcConnector.setSubmitData(myData);
    }
    else {
        Alert.show("You must specify a loan amount greater than zero", "Attention", Alert.OK);
        lcConnector.setDataInvalid();
    }
}
/*****
 * Retrieves the Workspace User and Task data sent from
 * Workspace ES2.
 *****/
private function handleWorkspaceData(event:SwfDataEvent):void {
    //Get the task information
    lcConnector.task = event.task;
    //Add task instructions to the form.
    taskInstructions.text = lcConnector.task.instructions;
    //Get information of the full name and email of the user logged in to
    //Workspace ES2 when no name is provided.
    var currentUser:User = lcConnector.getAuthenticatedUser();
    if (firstName.text == ""){
        firstName.text = currentUser.displayName;
        phoneOrEmail.text = currentUser.email;
    }
}
}
]]>
</mx:Script>

<!-- Enables communication with Workspace ES2 and binds methods for handling communication
with Workspace ES2. -->
<lc:SwfConnector id="lcConnector"

```

```

        formInitialData="handleInitDataListener(event) "
        formSaveDataRequest="handleFormSaveDataRequestListener(event) "
        formSubmitDataRequest="handleSubmitFormDataRequestListener(event) "
        setWorkspaceData="handleWorkspaceData(event) "/>

<!-- Defines the model -->
<mx:XML id="myData">
    <LoanApp>
        <Name>{firstLastName.text}</Name>
        <LoanAmount>{loanAmount.text}</LoanAmount>
        <PhoneOrEmail>{phoneOrEmail.text}</PhoneOrEmail>
        <ApprovalStatus>{approvalStatus.text}</ApprovalStatus>
    </LoanApp>
</mx:XML>
<!-- Creates 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="Preliminary Loan Approval Form" fontSize="28"
            color="black" fontWeight="bold" fontFamily="Arial"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Label id="taskinstructions" fontSize="10" text="" />
    </mx:HBox>
    <mx:HBox>
        <mx:Form backgroundColor="white">
            <mx:FormItem label="Name" required="true">
                <mx:TextInput id="firstLastName" text="{myData.Name}"
                    change="myData.Name = firstLastName.text;
                    lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Loan Amount" required="true">
                <mx:TextInput id="loanAmount" text="{myData.LoanAmount}"
                    change="myData.LoanAmount = loanAmount.text;
                    lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Phone or Email" required="true">
                <mx:TextInput id="phoneOrEmail" text="{myData.PhoneOrEmail}"
                    change="myData.PhoneOrEmail = phoneOrEmail.text;
                    lcConnector.setDirty();" />
            </mx:FormItem>
            <mx:FormItem label="Approval Status" required="false">
                <mx:TextInput id="approvalStatus" text="{myData.ApprovalStatus}"
                    change="myData.ApprovalStatus = approvalStatus.text;"
                    editable="false" />
            </mx:FormItem>
        </mx:Form>
    </mx:HBox>
</mx:VBox>
<!-- Validates the field that stores the loan amount -->
<mx:NumberValidator id="loanValidator"
    source="{loanAmount}"
    property="text"
    required="true"
    minValue="1"
    lowerThanMinError="Loan amount must be greater than 0."
    domain="real"
    listener="loanAmount"/>
</mx:Application>

```