

# Using Remote Object to Send Data to the Server

---

## Install LiveCycle Data Services ES

1. Browse to [https://www.adobe.com/cfusion/entitlement/index.cfm?e=lcds26\\_td](https://www.adobe.com/cfusion/entitlement/index.cfm?e=lcds26_td)
2. Either create an Adobe account or sign in using your member account information. You may be presented with an account information page that you will need to update or fill out.
3. Select and download the LiveCycle Data Services ES (LCDS) product that matches your system.

**Note:** This exercise assumes a Windows operating system.

4. Install LiveCycle Data Services ES by running **lcds261-win.exe**.
  - At the Serial Number screen, use **1306-4038-4494-1343-9848-7117**.
  - At the Installation Location screen, accept the installation path of **c:\lcds**.
  - At the Installer Options screen, select **LiveCycle Data Services with Tomcat**.

**Note:** Tomcat is the integrated application server.
  - Complete the installation.

## Installing the Server Files

5. Unzip **Server.zip** to **C:\lcds\tomcat\webapps**.

You will see a new directory in **\webapps** named **odt** that contains the following files and folders:

  - **META-INF** – This contains the **manifest.mf** file for this LCDS instance. You will not need to do anything with this file.
  - **WEB-INF** – This directory contains the following elements:
    - **web.xml** – This file defines the path of the **services-config.xml** file to be used for this LCDS instance.
    - **classes** – This directory contains a folder named **reservation** with pre-compiled Java class files that will be used in this exercise to insert data from the Flex application into the database.
    - **db** – This folder contains the Jar file and license for the HSQLDB database engine, a property file for the database and a sub-folder called **xyz** that contains the script and property files that create and populate the XYZ database used in this exercise.

**Note:** See [www.hsqldb.org](http://www.hsqldb.org) for more information about this free, open-source database engine.

- **flex** – This folder contains some configuration files for the LCDS instance.
  - Note:** You may need to modify the **services-config.xml** file if you get port conflict errors when starting LCDS.
- **lib** – This folder contains all the jar files for the LCDS instance.
- **src** – This contains a folder called **reservation** with the source Java files for the compiled class files in the **classes** directory mentioned earlier.
- **index.html**– This is the default web page for the LCDS instance. You will view it in the next section.

## Starting and testing the installation

6. Select **Start > All Programs > Adobe > LiveCycle Data Services ES 2.6.1 > Start LiveCycle Data Services Server**.
7. Browse to this url: <http://localhost:8400/odt>.

You should see the LiveCycle Data Services ES welcome page.

**Note:** You do not need to start the samples database to run this example. It needs to be started only if you want to look at the other samples. In order to run the sample applications, select **Start > All Programs > Adobe > LiveCycle Data Services ES 2.6.1 > Start Samples Database**. You can then browse to <http://localhost:8400/lcds-samples>.

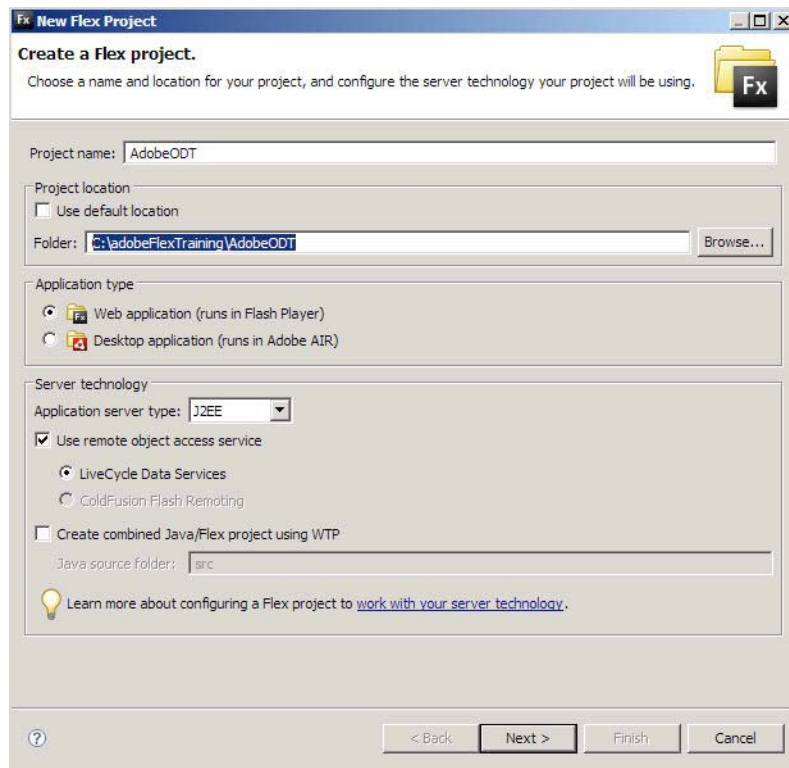
## Recreate the project using LCDS

8. Import **Ex9\_Starter.zip** into the **adobeFlexTraining** workspace to see the **AdobeODT** project.
  - Note:** You can alternatively start with the files you completed in Exercise 8.
9. In Flex Builder's **Navigator** view, right click on the **AdobeODT** project and select **Delete**. Make sure the **Do not delete contents** is selected.
  - Note:** In the following steps you will convert this Flex Builder project into an LCDS-enabled Flex Builder project.
10. Select **File > New > Flex Project**.
11. The project name is **AdobeODT**.
12. Uncheck the **Use default location** checkbox.
13. Select the project location for the **AdobeODT** project you just deleted. If you are using the **adobeFlexTraining** workspace for this series, the folder is **C:\adobeFlexTraining\AdobeODT**.
14. The application type is **Web Application**.
15. Under **Server Technology**, select **J2EE** as the application server type.
16. Check **Use remote object access service** and **LiveCycle Data Services** should be selected.

17. Make sure to uncheck the **Create combined Java/Flex project using WTP** checkbox.

**Note:** This option might not exist in some versions of Flex Builder. If you don't see the checkbox, ignore this step.

The dialog window should appear as follows:

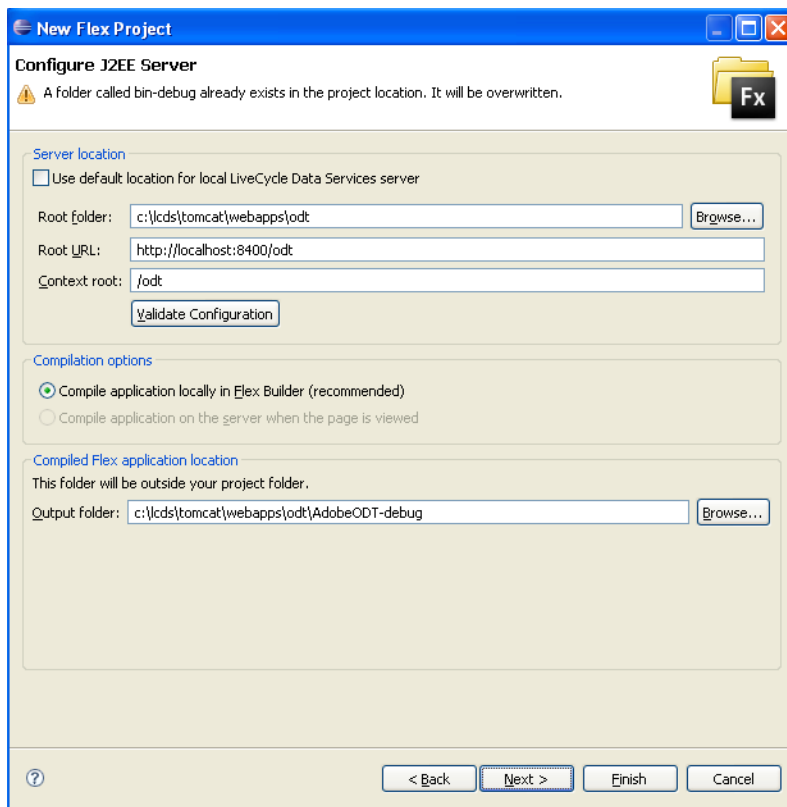


18. Click **Next**.
19. Under **Server location**, uncheck the **Use default location for local LiveCycle Data Service server** checkbox.
20. The root folder is **C:\lcds\tomcat\webapps\odt**.
21. The root url is **http://localhost:8400/odt**.
22. The context root is **/odt**.
23. Use the **Validate Configuration** button to validate your set up.

**Note:** Make sure the Live Cycle Data Services Server has been started.
24. Under **Compilation options**, make sure **Compile application locally in Flex Builder** is selected.
25. Under **Compile Flex application location**, the output folder should be **C:\lcds\tomcat\webapps\odt\AdobeODT-debug**.

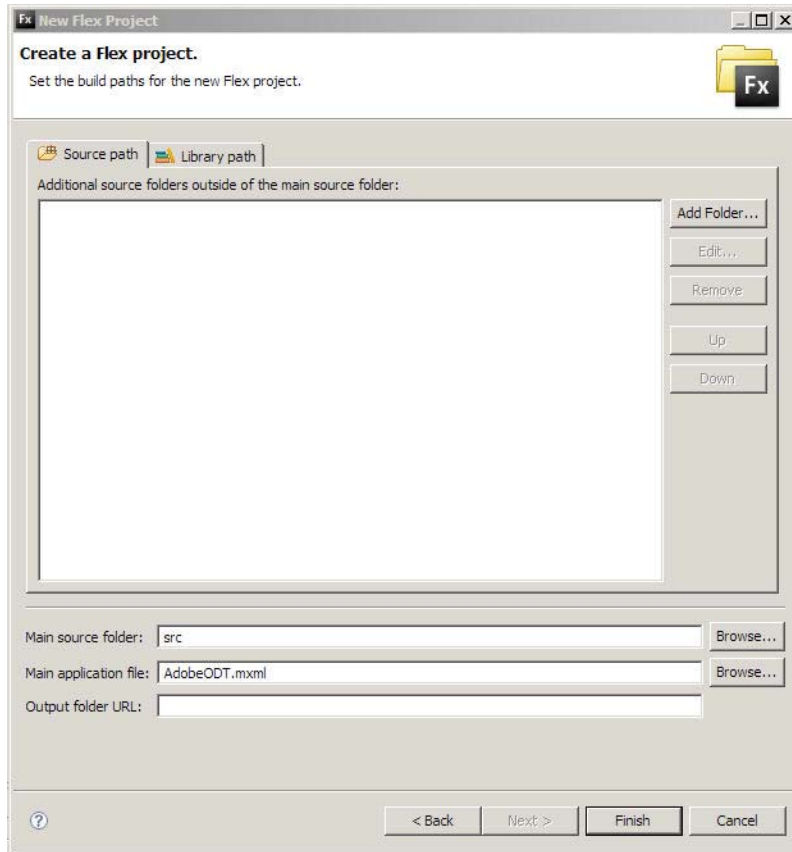
**Note:** This is where the compiled SWF and the associated wrapper files will be placed by Flex Builder. Unlike in previous exercises, this is outside the Flex Builder project directories.

The dialog box should appear as follows:



26. Click **Next**.
27. The main source folder is **src**.
28. The main application file is **AdobeODT.mxml**.

The dialog box should appear as follows:



29. Click **Finish**.
30. Run **AdobeODT.mxml**.
31. The browser url should display <http://localhost:8400/odt/AdobeODT-debug/AdobeODT.html>.

**Note:** You should see the XYZ Convention Center application as it appeared at the end of Exercise 8 of this series.

## Create a value object class

32. Return to Flex Builder.
33. In the **Navigator** view, right-click on the **src** folder of the **AdobeODT** project and select **New > Folder**.
34. Name the folder **vo**.
35. Click **Finish**.
36. Right-click on the **vo** folder and select **New > ActionScript Class**.
37. Name the class **Reservation**.
38. Keep all other defaults and click **Finish**.
39. Before the class definition, but after the opening curly brace of the package statement, add a `Bindable` metadata tag.

40. After the class definition but before the constructor, create a public variable named `reservationId` datatyped as `int` with a value of 0 (zero).

Your code should appear as follows:

```
public var reservationId:int = 0;
```

41. Create another public variable named `fullName` datatyped as `String` with an empty string value.

Your code should appear as follows:

```
public var fullName:String = "";
```

42. Create the following variables as public variables datatyped as `String` objects with empty values: `address`, `city`, `state`, `postalCode`, `phone`, `dateNeeded` and `options`.

Your class definition code should appear as follows:

```
[Bindable]
public class Reservation
{
    public var reservationId:int = 0;
    public var fullName:String = "";
    public var address:String = "";
    public var city:String = "";
    public var state:String = "";
    public var postalCode:String = "";
    public var phone:String = "";
    public var dateNeeded:String = "";
    public var options:String = "";

    public function Reservation()
    {
    }
}
```

43. Save the file.

## Populate value object

44. In the **ReservationForm.mxml** component in the **components** folder, locate the `Script` block.

45. After the last function in the Script block, create a new private function named `clearForm` that takes no parameters and has a `void` return type.
46. Within the function set the `text` property of the following controls to an empty string: `fullname`, `address`, `city`, `state`, `postalcode`, `phone` and `dateNeeded`.
47. Use the `removeAll()` method of the `selectedOptions ArrayCollection` instance.

Your code should appear as follows:

```
private function clearForm():void {
    fullname.text = "";
    address.text = "";
    city.text = "";
    state.text = "";
    postalcode.text = "";
    phone.text = "";
    dateNeeded.text = "";
    selectedOptions.removeAll();
}
```

48. Locate the `validateForm()` function.
49. After the code within the function, create a conditional statement to test if `vals.length` is equivalent to zero.

Your code should appear as follows:

```
if (vals.length == 0){
}
```

50. If the conditional statement tests true, create and instantiate a local variable named `dataObj` datatyped as `Reservation` class instance.

Your code should appear as follows:

```
var dataObj:Reservation = new Reservation();
```

51. Assign `fullname.text` to `dataObj.fullName`.
52. Assign `address.text` to `dataObj.adress`.
53. Assign `city.text` to `dataObj.city`.
54. Assign `state.text` to `dataObj.state`.
55. Assign `postalcode.text` to `dataObj.postalCode`.

56. Assign `phone.text` to `dataObj.phone`.
57. Assign `dateNeeded.text` to `dataObj.dateNeeded`.
58. Use the `toString()` method of `selectedOptions` and assign it to `dataObj.options`.

Your code should appear as follows:

```
private function validateForm():void {
    var vals:Array = new Array();
    vals = Validator.validateAll(myValidators);
    if (vals.length == 0) {
        var dataObj:Reservation = new Reservation();
        dataObj.fullName = fullname.text;
        dataObj.address = address.text;
        dataObj.city = city.text;
        dataObj.state = state.text;
        dataObj.postalCode = postalcode.text;
        dataObj.phone = phone.text;
        dataObj.dateNeeded = dateNeeded.text;
        dataObj.options = selectedOptions.toString();
    }
}
```

## Send data by RemoteObject

59. After the Script block, create a `RemoteObject` call with an `id` property set to a value of `hs` and a `destination` property with a value of `reservationService`.

Note: **reservationService** is the class in **C:\lcds\tomcat\webapps\odt\WEB-INF\classes\reservations**.

60. Add a `fault` property with a value of `faultHandler` and `pass event` as the only parameter.

Your code should appear as follows:

```
<mx:RemoteObject id="hs"
    source="reservations.ReservationService"
    destination="reservationService"
```

```
        fault="faultHandler(event)">
    </mx:RemoteObject>
```

61. Between the `RemoteObject` tags create a `method` tag with a `name` property set to a value of `doCreate` and a `result` property with a value of `resultHandler`. Pass `event` as the only parameter.

Your code should appear as follows:

```
<mx:RemoteObject id="hs"
    source="reservations.ReservationService"
    destination="reservationService"
    fault="faultHandler(event)">
    <mx:method name="doCreate"
        result="resultHandler(event)"/>
</mx:RemoteObject>
```

62. Locate the `Script` block.
63. After the last `import` statement, import the `ResultEvent` class.
64. Import the `vo.Reservation` class.
65. Import the `FaultEvent` class.
66. Import the `Alert` class.

Your code should appear as follows:

```
import mx.rpc.events.ResultEvent;
import mx.rpc.events.FaultEvent;
import vo.Reservation;
import mx.controls.Alert;
```

67. Before the end of the `Script` block, create a private function named `resultHandler` that takes one parameter named `event` datatyped as `ResultEvent`. The function returns `void`.
68. Within the function invoke the `show()` method of the `Alert` class and pass two parameters:
- Your form was sent successfully.
  - Confirmation

Your code should appear as follows:

```
private function resultHandler(event:ResultEvent):void {
    Alert.show("Your form was sent
successfully.", "Confirmation");
}
```

```
}
```

69. Before the end of the `Script` block, create a private function named `faultHandler` that takes one parameter named `event` datatyped as `FaultEvent`. The function returns `void`.
70. Within the function invoke the `show()` method of the `Alert` class and pass two parameters:
- Your form was not sent successfully.
  - Error

```
private function faultHandler(event:FaultEvent):void{  
    Alert.show("Your form was not sent  
successfully.", "Error");  
}
```

71. Locate the `validateForm()` function.
72. As the last line within the conditional `if` statement, invoke the `doCreate()` method of the `hs RemoteObject` instance and pass `dataObj` as the only parameter.

```
hs.doCreate(dataObj);
```

73. Call the `clearForm()` function.

```
clearForm();
```

74. Save the file.

75. Locate and open the **Reservation.as** file in the **vo** folder.

76. After the package code within the curly brace but before the `Bindable` metadata tag, add a `RemoteClass` metadata tag.

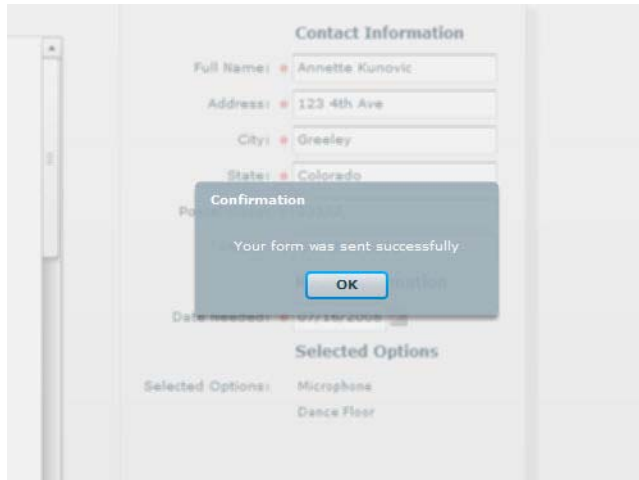
77. In parenthesis add an `alias` property with a value of `reservations.Reservation`.

```
package vo  
{  
  
    [RemoteClass(alias="reservations.Reservation")]  
    . . .  
}
```

78. Save the file.

79. Return to `AdobeODT.mxml` and run.

Enter data into each field, select two amenities and then submit. You should get an alert message with the status of your submission.



80. Open **C:\lcds\tomcat\webapps\odt\WEB-INF\db\xyz.log**.

**Note:** You will see the data that you just submitted to the database. The log file for an HSQLDB database contains all transactions made with the database.