Revised edition of *Flex 3 in Action*

FLEX 4 IN ACTION

Tariq Ahmed
Dan Orlando
WITH John C. Bland II
AND Joel Hooks

SAMPLE CHAPTER

MANNING

Flex 4 in Action

by Tariq Ahmed
and Dan Orlando

with John C. Bland II and Joel Hooks

Chapter 15

brief contents

15 Working with data services

This chapter covers

- Data-centric development with Flash Builder
- Connecting to web services
- Using `HTTPService` and `WebService` components
- Understanding Action Message Format (AMF)
- Communication with Java EE using BlazeDS
- ColdFusion communication
- Communicating with PHP via Zend_AMF
- Setting up your development environment for a seamless client/server workflow

Integrating an RIA client application with its corresponding server-side application services can be a daunting task the first time you do it, so if you don't have much experience in this area, strap in tight because you're going to be spending about the next hour or so getting the crash course in RIA data communications. By the time you've finished with this chapter, you'll have a level of confidence that's felt by experts in the field, and you'll be considered a top resource in your workplace for enterprise data communications for RIA.

Client/server RIA communication gets easier the more you do it. Understanding the inner workings of how a Flex application communicates with data services will provide you with the knowledge to be able to connect a Flex or AIR application to any server-side technology, even if you don't know the server-side language being used. This is a highly sought-after skill set that you'll be able to add to your arsenal upon completing this chapter.

15.1 Accessing server-side data

RIA architects often find themselves bewildered when faced with the dilemma of choosing a web service infrastructure. Knowing the technology that's available in this area is critical, because the wrong decision could be catastrophic. Table 15.1 is a high-level overview of the services that are available to you and can be used as a reference when making infrastructure design decisions.

Table 15.1 Web service protocol matrix

Communication Server support	Application	Benefits
HTTP (includes REST and RPC hybrids)		
- All	Simple widget-based applications; speed and real-time UI updates aren't required.	Easy implementation via the `HTTPService` object; RPC hybrid protocols can be invoked using `RemoteObject`.
SOAP/WSDL		
- All	Data aggregation from external web services.	Easy implementation; pull data from multiple outside resources regardless of platform.
AMF		
- BlazeDS & LiveCycle Data Services (LCDS): Java, - .NETZend: PHP - AMFPHP: PHP - WebOrb: .NET, Ruby, PHP	Approaching enterprise level; speed is important; data is usually pulled from server by polling.	Binary data compression makes communications 12 times faster; strong data typing; multiplatform support.
RTMP		
- LiveCycle Data Services (LCDS), - Flash Media Server (FMS)	Enterprise level, messaging, instantaneous UI updates; data can be pushed to the client; streaming media content; data-intensive RIAs.	Integrates into existing J2EE infrastructure; document management, rapid data transfer, clustering, data tracking, syncing, paging, and conflict resolution.
Flash Remoting		
- Native to ColdFusion	Robust, enterprise platform for client/server Flex communications; native.	Seamless integration with the Flash platform; removes the need for an intermediate code library to do data type mapping and data serialization.

Table 15.1 Web service protocol matrix *(continued)*

Communication Server support	Application	Benefits
JSON		
- All (JavaScript data objects are serialized and transferred in binary form.)	AIR applications that use AJAX or Flex applications that use the `ExternalInterface` API.	Easy implementation with the `HTTPService` object; part of the AS3CorLib library.

TIP Note that you must place a cross-domain policy file at the root of the domain in order to use the HTTPService object. If you aren't familiar with the cross-domain policy, the Adobe Developer Connection has a good resource at the following web address: http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html.

Table 15.1 is a basic protocol matrix that provides a starting point for knowing when to use each of the technologies available for Flex data communications along with the benefits that can be gained for each of them.

15.1.1 Using the HTTPService object

Because the `HTTPService` object uses the same request-response paradigm as your Internet browser does when it displays web content, it can be used to invoke operations on any server-side technology with `GET` and `POST`. It doesn't require an intermediate code library for socket-level serialization and parsing as other data communication protocols do. Listing 15.1 illustrates how the `HTTPService` object is declared in MXML so it can be invoked later to run a search query on the Yahoo! search engine.

Listing 15.1 The `HTTPService` object declared in MXML notation

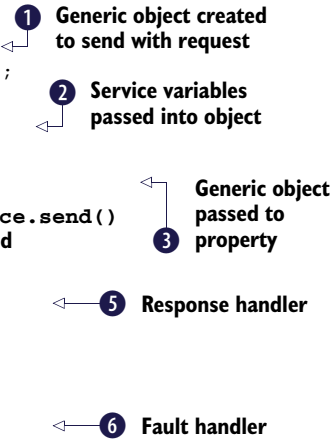
```
<mx:HTTPService
    id="yahooHTTPService"
    url="http://search.yahooapis.com/WebSearchService/V1/webSearch"
    method="GET"                          <-- Request can be GET or POST
    makeObjectsBindable="true"            <-- Makes result object bindable
    result="responseHandler(event)"       <-- Declare the method to handle the result
    fault="httpFaultHandler(event)"       <-- Declare a method to handle a fault
    showBusyCursor="true">                <-- Show busy cursor while in progress
</mx:HTTPService>
```

Before the service is invoked, a request object is set up so the necessary arguments can be sent with the service call; see the following listing.

Listing 15.2 Calling the `HTTPService` from ActionScript

```
<fx:Script>
  <![CDATA[

  import mx.rpc.events.ResultEvent;
  import mx.rpc.events.FaultEvent;
```

```
    public function sendHttpRequest():void {
       var requestObj:Object = new Object();
       requestObj.appid = new String("YahooDemo");
       requestObj.query =
          new String("Flex in Action");
       requestObj.results = new int(2);
       yahooHTTPService.request = requestObj;
       yahooHTTPService.send();
    }

    private function
       responseHandler(e:ResultEvent):void {
       trace("Received a result: " + e.result);
    }

    private function
       httpFaultHandler(e:FaultEvent):void {
       trace("Received a Fault: " + e.message);
    }

  ]]>
</fx:Script>
```



As shown in listing 15.2, the function `sendHTTPRequest()` invokes the `HTTPService` object. Before that happens, though, a generic object called `requestObj` is created ❶ and the variables `appid`, `query`, and `results` are passed to it ❷, which the Yahoo! service will expect to find on the generic object when it receives the request. The `requestObj` is then passed into the `HTTPService.request` property ❸, and finally the `HTTPService.send()` method is called ❹.

In addition, the functions named `responseHandler` and `faultHandler` (❺❻) are implemented to catch the event that's triggered by the service's response to the request. As you can see in the `Alert.show` that's called in the body of the `responseHandler` function, the data that's returned from the server can be accessed through the generic `ResultEvent.result` property. Any data type can be passed through this property, including value objects (or data transfer objects if you come from the world of Java or .NET development). You've learned all about the `HTTPService` object, which makes this an excellent opportunity to introduce you to the `WebService` object, as you'll see in the next section.

15.1.2 Consuming web services with the WebService component

Web Service Description Language (WSDL) is a standard format for describing SOAP- and RPC-based web services. This type of data consumption is great for mash-up-style applications, because many online services have adopted the Simple Object Access Protocol (SOAP) standard, but this is generally the slowest way to transfer data because it carries a lot of overhead with it. On the other hand, SOAP is similar to HTTP in that it's supported by just about every server-side platform.

Setting up web service integration in Flex is easy and takes only a few lines of code. For example, if you wanted to use the `<mx:WebService>` tag to connect to the weather web service, it would look like the code in the following listing.

Listing 15.3 Using the `WebService` component

```
<mx:WebService id="weatherService"                    Declare the WSDL document location (1)
   wsdl="http://www.webservicex.net/WeatherForecast.asmx?WSDL"
   fault="wsdlFault(event)">
     <mx:operation name="GetWeatherByZipCode"
        result="weatherResponse(event)"
        fault="weatherFault(event)"/>                 (2) Multiple operation declarations within a WebService
     <mx:operation name="GetWeatherByPlaceName"
        result="weatherResponse(event)"
        fault="weatherFault(event)"/>
</mx:WebService>
```

The `<mx:WebService>` tag contains the necessary information to point the Flex application to the WSDL document (1). When the `<mx:WebService>` tag is initialized, it parses the WSDL document and extracts the information it needs to generate Flex objects with which you can interact with the web service.

The `operation` tags (2) define the various operations that are used with this web service. In this case, the `GetWeatherByZipCode` and `GetWeatherByPlaceName` operations are defined and ready for you to invoke in the same way as you did earlier with the `HTTPService` object. Each operation has a handler defined to deal with the response from the service request.

> **TIP** In some cases, a web service you'd like to connect to may use methods that are reserved words in Flex. In these situations, you can use the `WebService.getOperation("nameOfOperation")` function to get a handle for the operation.

Flex removes the responsibility of translating SOAP XML packets into usable objects by abstracting the functionality for parsing SOAP packets within the `WebService` class. You can obtain the data object that's created after parsing is completed from the generic object `ResultEvent.result` after parsing of the data packets has completed and the `ResultEvent` is fired and captured by your result handler function.

15.2 Action Message Format in action

As you saw back in table 15.1 at the beginning of the chapter, the Action Message Format is a robust communication protocol that's quickly becoming the preferred method of communication among RIA developers because of its open technology, speed, and native support for the Flash virtual machine.

To show how fast AMF is, Adobe senior technical evangelist James Ward wrote a Census RIA Benchmark application. He's been updating it over the course of the last couple of years with new features and has included as many ways of transferring data as the community has been able to throw at him. Despite stiff competition from some serious community challengers, the AMF protocol remains the reigning champion. Figure 15.1 provides a screenshot of the application, which can be found at http://www.jamesward.com/census/.

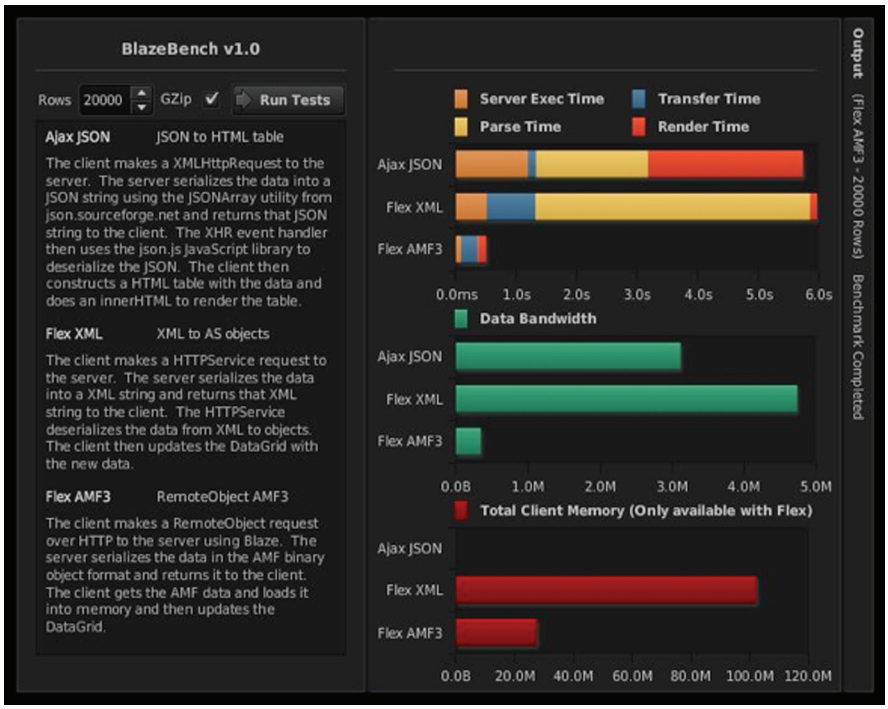


Figure 15.1 **James Ward's benchmark illustrates how impressive AMF is for data communication.**

Comparing data transfer times is truly a dose of reality. In figure 15.1, I set the number of rows to retrieve to a minimum of 500 (from 5,000) and turned off GZIP compression, and then I ran benchmarks only on data transfer mechanisms that are comparable to AMF3.

Upon running the same benchmark on any of the cases that involved SOAP, my browser usually hung after about 90 seconds. It then displayed a message that the server was taking a long time to respond, prompting me to tell it whether I wanted to continue waiting or not. This was appalling, and yet I found it to be quite humorous at the same time. Spend a few minutes with the Census RIA Benchmark application, and I guarantee that you'll think twice about your choice of communication protocol the next time you build an enterprise Flex application.

AMF presents you with a number of choices with regard to its implementation, but the options are usually narrowed by your choice of server-side technology. In the next sections, we'll do a brief roundup of the technologies available at the time of this writing.

15.2.1 Open-source AMF

In December of 2007, Adobe announced that AMF would become an open protocol, which immediately led to the development of a number of code libraries, many of which mimicked the data transfer capabilities with ColdFusion via Flash Remoting.

Opening up the protocol was the icing on the cake for most Flash platform developers. It was already speedier than anything else, and its support is native to the Flash virtual machine. The number of open source code libraries that were developed soon after is evidence of this. Getting to know the tools that are available for data communication with AMF is simple.

15.2.2 AMF with PHP

The open source AMFPHP project began in January of 2008, right after AMF went open source. It included a fairly steep learning curve for developers who were new to Flex at the time, and there was little documentation to go by, resulting in a lot of trial and error for determined PHP RIA programmers. Thankfully, Adobe formed a strategic alliance with Zend Technologies in Q308, and the Zend_AMF module was developed and integrated into the PHP Zend framework by a friend of mine, Wade Arnold.

The Zend_AMF module is easy to work with and is currently the ideal solution for data communication between Flex and PHP, especially if you're using Flash Builder for your Flex application development. One of the greatest new features of Flash Builder 4 is its ability to turbocharge your workflow by automating the process of hooking up data services, as you'll see in section 15.3.

15.2.3 AMF and ColdFusion

At the time of this writing, the beta of ColdFusion 9 was just released along with the first-ever ColdFusion development tool made by Adobe, called CFBuilder. Not surprisingly, AMF is most effective when working with ColdFusion because it has native support for the AMF format. That means that when running a Flash platform RIA application on the client with ColdFusion data services on the server, you have native support for the fastest available data-transfer mechanism on both the client and the server.

The result is a rich, web-based application that operates as if it's running native to the local desktop. More specifically, massive amounts of data can be captured, filtered, organized, processed, and displayed in a human-readable graphical summation in less than a second. It was merely a few years ago that this kind of query would be run by a research technician, who would then get up and go have lunch and come back 45 minutes later only to find that the query was not complete yet.

15.2.4 BlazeDS

Adobe open-sourced the AMF specification in tandem with the release of BlazeDS, an open source Java application that can be used for integrating Java and Flex via AMF. With BlazeDS, developers can invoke methods on preexisting plain old Java objects (POJOs), Spring Services, EJBs, and other Enterprise Java implementations. Blaze can also be hooked into JMS and Hibernate for applications that require messaging. Java application servers supported by BlazeDS include Tomcat, WebSphere, WebLogic, JBoss, and ColdFusion.

15.2.5 LiveCycle Data Services

BlazeDS was an offspring of LiveCycle Data Services ES. It's logical to assert that BlazeDS is the baby sibling of LCDS. The target market for LCDS is primarily large-scale enterprise environments consisting of large server farms. Considering the cost of a single LCDS license, it's no wonder that most small businesses and entrepreneurial developers view LCDS as out of reach. As with BlazeDS, LCDS is a Java-based implementation of AMF and offers additional advantages that are conducive to the needs of the enterprise. The cost is typically justified by the level of support that comes with the package.

15.2.6 Additional technologies

Other technologies that have surfaced over the last couple of years include WebOrb, AMF.NET, AMFPHP, and RubyAMF. But the three that stand out among the rest in the RIA arena are Zend_AMF, ColdFusion Remoting, and BlazeDS for Java EE. We'll be focusing on those technologies for the remainder of this chapter.

You're now armed with the background knowledge that you'll need to embark on your next mission: learning to build data-centric AMF applications using Flash Builder.

15.3 Building data-centric applications with Flash Builder

The data-centric development (DCD) features that ship with Flash Builder were created to shave a significant amount of development time for data-driven Flex applications and are undoubtedly my personal favorite of all the new Flash Builder features.

The first thing to point out with regard to the highly anticipated data-centric development features included in Flash Builder 4 is that the workflow generally remains the same regardless of the server-side technology being used. In a moment we'll walk through this workflow. I'll point out differences in the workflow that are dependent on the server-side technology being used as we walk through the examples.

Development time is cut down by implementing a series of DCD wizards that guide you through the process of connecting to your server-side code and transferring strongly typed data objects, otherwise known as *value objects*. While you're guided through the wizards, code is being generated in the background for you. The following example demonstrates how you can truly streamline your workflow for connecting to data services without even leaving the design view of the Flash Builder IDE!

15.3.1 Setting up the right environment

When developing RIA applications, your integrated development environment (IDE) should reflect both the client and server-side platforms you're working with. Table 15.2 provides a list of recommendations for how you can configure your environment based on your choice of server-side platform.

The concept of the IDE matrix shown in table 15.2 is to be able to do both client and server-side development without leaving the Eclipse environment. This will streamline your workflow and save you a lot of time.

Table 15.2 Eclipse IDE configuration matrix for data-driven Flex 4 RIA development

PHP	Eclipse PDT + Flash Builder 4 plug-in	Zend Studio + Flash Builder 4 plug-in
J2EE/Blaze/LCDS	Eclipse for Java + Flash Builder 4 plug-in	Eclipse for Java EE + Flash Builder plug-in
ColdFusion	CFEclipse + Flash Builder 4 plug-in	CFBuilder + Flash Builder 4 plug-in
WSDL	Flash Builder 4 + WDT Eclipse plug-in	WDT Eclipse + Flash Builder 4 plug-in
.NET	Flash Builder 4 + WDT Eclipse plug-in	WDT Eclipse + Flash Builder 4 plug-in

HTTP, RPC, AMF, and SOAP, oh my!

With Flex and Flash Builder 4, there's no need to be overwhelmed by the choice in protocol for your data communications. Flash Builder 4 takes care of most of this for you behind the scenes when you use the DCD wizards and tries to use Action Message Format for data communications whenever possible because of its superior transfer speeds, among other things.

15.3.2 Establishing connection to the server

Because the Flex framework is based on a Model-View-Controller architecture, it's usually in your best interest to maintain consistency by using MVC methodology in your application as well. In this section, we'll focus on using the Flash Builder GUI to generate the code for the model layer of your application. But first you need to establish a connection to your server-side application layer.

> **TIP** The data-centric development workflow built into the Flash Builder 4 IDE is almost identical between integrating with PHP and integrating with ColdFusion services.

You have two scenarios for connecting to a data service: Either you're trying to connect from a project that already exists, or you're starting fresh with a new project.

In the first scenario, you have an existing project that you'd like to connect to a data service. This is easily done by selecting the Connect to Data/Service link from the Data/Services panel in Flash Builder, as shown in figure 15.2.

In the second scenario, you can set up the connection to the server during creation of a new Flash Builder project. As shown in figure 15.4, the New Project Setup Wizard includes a drop-down menu that allows you to select an application server for configuration during the project setup process. The options are None/Other, .NET, J2EE, ColdFusion, and PHP.

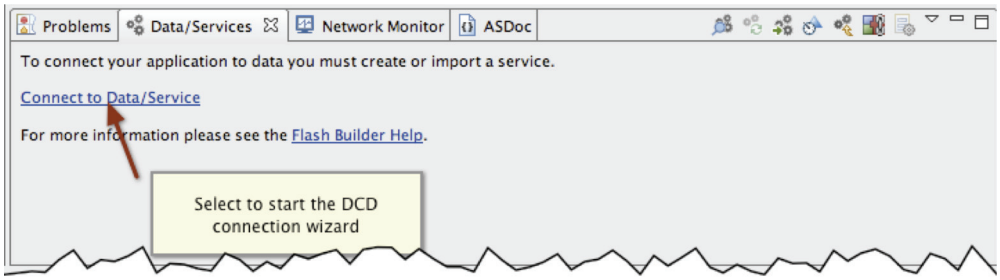


Figure 15.2 **Select Connect to Data/Service from the Data/Services panel to start the DCD wizard. You're then greeted with the window shown in figure 15.3, which lists all of the service types that you can connect to.**

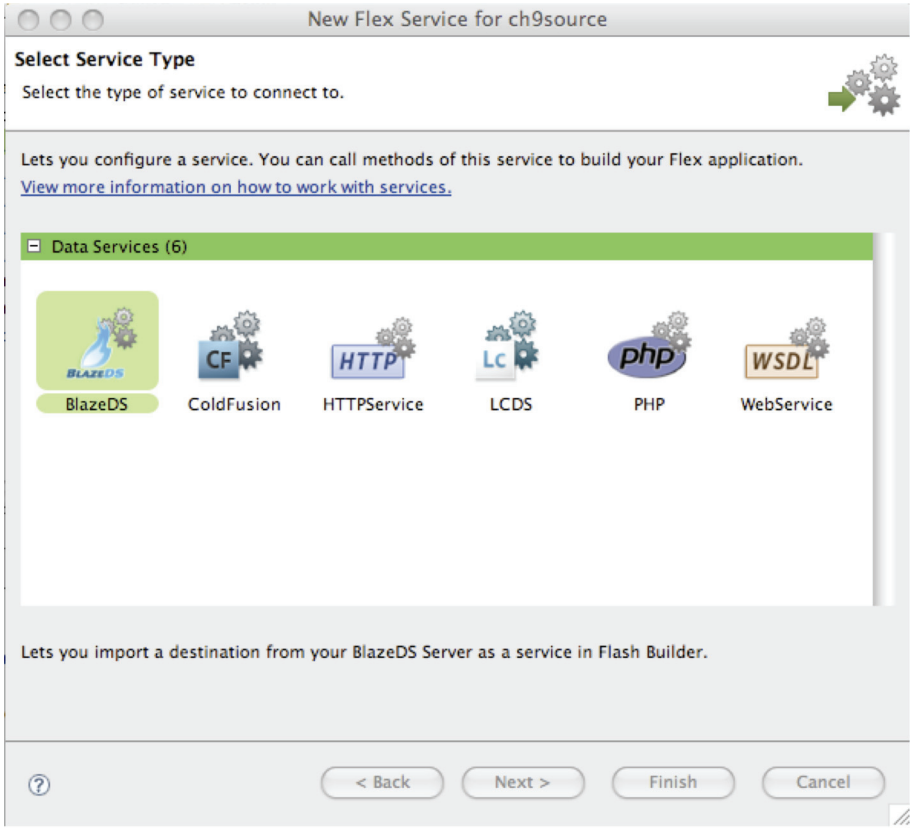


Figure 15.3 **Select from the list of available service types.**

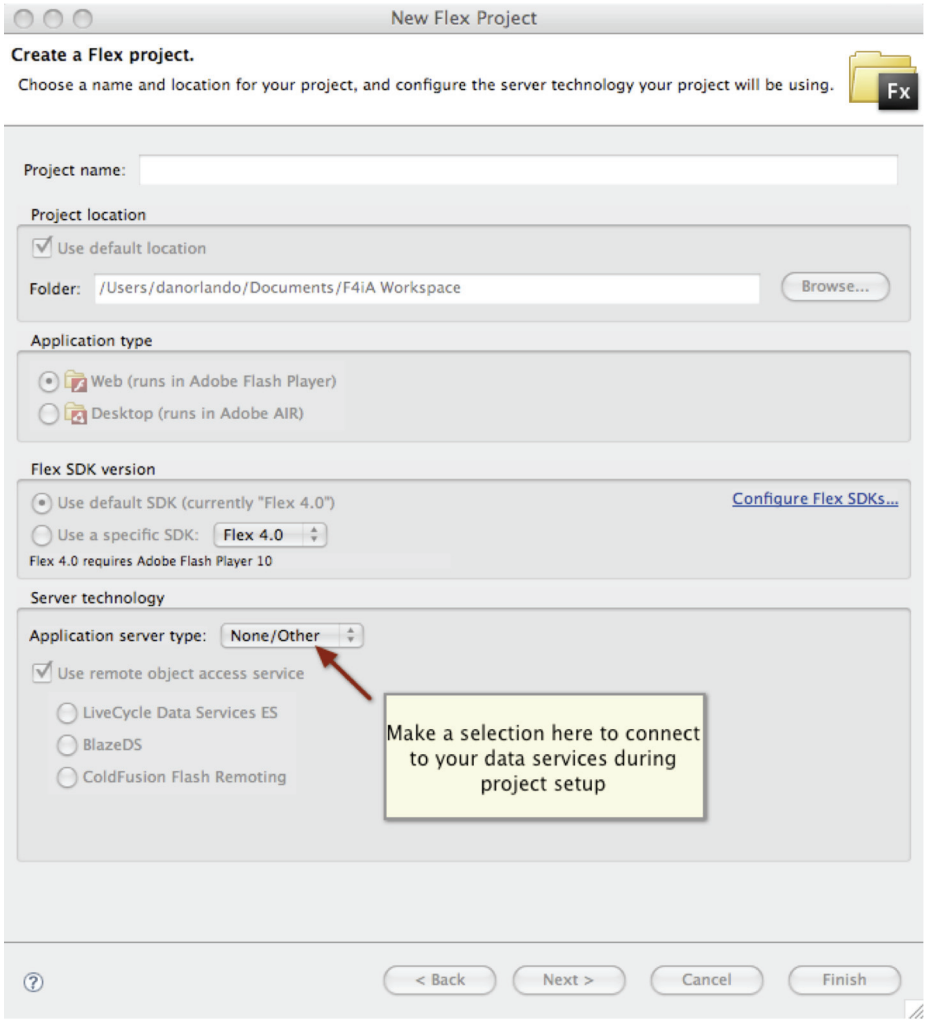


Figure 15.4 Use the drop-down menu to set up the application server when creating a new project in Flash Builder.

Regardless of the scenario, you'll end up in the same place, the server configuration dialog box. Figure 15.5 demonstrates the server configuration dialog window after I selected PHP as the application server type.

Regardless of the server type, Flash Builder is concerned about two things here:

1. The endpoint URI that it should connect to
2. Validating that it's able to connect successfully

After you enter the necessary parameters in the configuration fields, the button labeled Validate Configuration will be enabled. The wizard won't allow you to continue until you successfully validate your server configuration. You'll know when

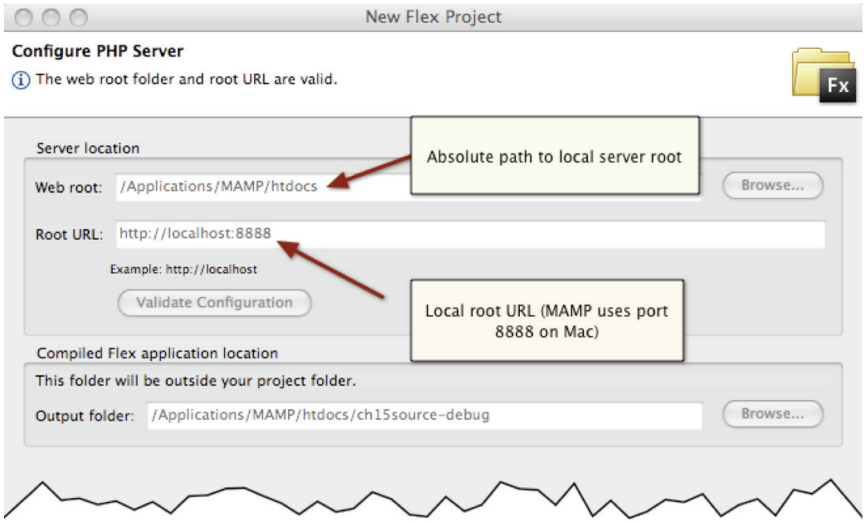


Figure 15.5 **Configuring a local PHP server for DCD in Flash Builder**

validation is successful because you'll see a friendly message at the top of the window, as shown in figure 15.5.

> **NOTE** If you're working with PHP on the server side, and Flash Builder sees that you don't yet have the Zend Framework installed, it will ask you if you want it to be installed. This is done so you can take advantage of the benefits of AMF through the Zend_AMF module. Make sure you select Yes if you're prompted with this message.

Now you're ready to have some fun as you learn how to autogenerate your services right from Flash Builder!

GENERATING SERVICES

One of the neatest things about the DCD features in Flash Builder 4 is the ability for the application to generate basic services for you containing standard CRUD operations for a specified database table. It also includes other handy operations such as retrieving paged result sets (20 at a time, for example).

If you've already created your database schema, Flash Builder can read the schema of a specified table and generate all the code you should need for a basic service from it. If that table contains many fields, the amount of time that this can save is immeasurable.

The process of generating a service stub begins by selecting the Click Here to Generate a Sample link under the PHP Location field, as shown in figure 15.6. If the code for your service has already been written, however, you can use this window to specify the name and location of the service so it can be introspected by Flash Builder.

The dialog box that appears next in the sequence is displayed in figure 15.7. To take full advantage of the built-in DCD features, make sure you've first created a data-

Figure 15.6 **Service stubs can be generated right from the service configuration dialog box.**

base and a table with fields inside your database. Then, make sure that Generate from Database is already selected when the sample service generator window opens. Fill in the rest of the parameters according to your database configuration, and click the Connect to Database button. You'll begin to see the magic happen.

Assuming all of your input parameters were correct and Flash Builder connected to the database you specified, you'll notice that all of a sudden the Table drop-down menu populates with all of the tables in your database. The next thing you need to do is select the table that you want to use to generate the service stub. Now click OK, and let the games begin!

The first thing you might think is that Flash Builder took on a mind of its own, and you may wonder what just happened when it's finished. Here's a rundown of what happened in those few tenths of a second:

- A package was added to your src folder.
- A services folder appeared under the libs folder.
- The service class was generated and opened up in the main editing pane of the Eclipse IDE or the default editor on your system if it isn't Eclipse for that file type.
- The class was introspected by Flash Builder, and all of its methods were displayed in the Data/Services panel.
- An abstract class was created in the project that contains methods to easily call each of the service's operations.
- An empty class was also created that makes the call to the super (abstract) class. The empty class is where you should place any custom code you might need.

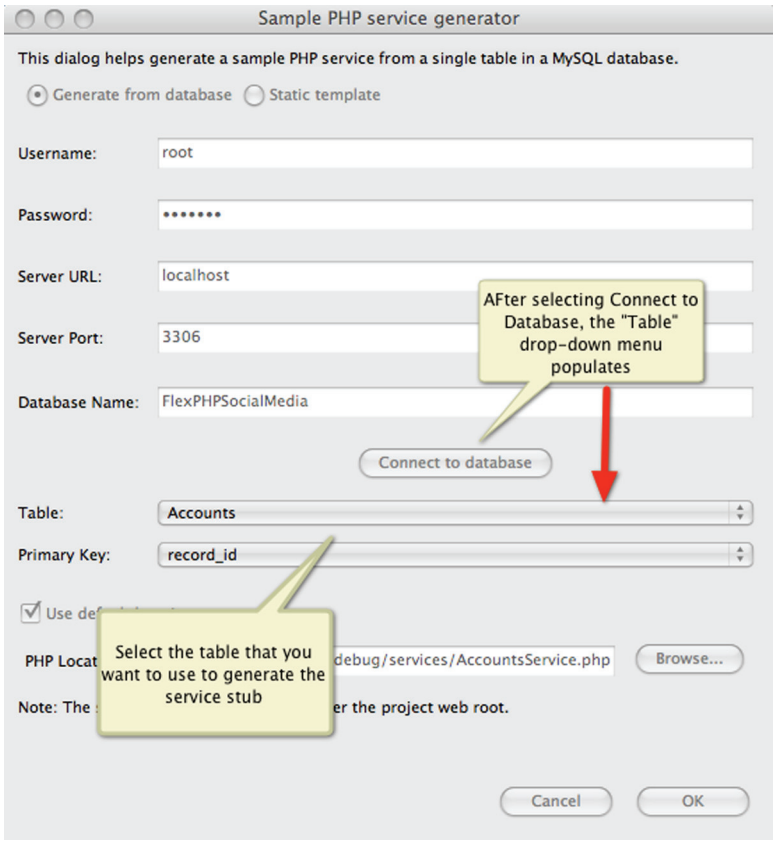


Figure 15.7 **The database table names appear in a drop-down menu after the Connect to Database button is clicked.**

If you're working in the full version of Flash Builder without having installed the necessary editor for your service code into the IDE, Flash Builder opens the generated service with whatever the default application is for that file type (most likely Adobe Dreamweaver, if it's installed on your local machine).

> **MIGRATION TIP** The important thing to understand is that the old paradigm that was used with previous versions of the Flex SDK—where value objects were created on the server side to correspond with value objects on the client side—has changed significantly. More specifically, you don't need to create a strongly typed PHP object that corresponds with your ActionScript objects (unless you want to!). With that said, for basic CRUD services, little coding is needed in the autogenerated PHP service other than what's already there. Configure your database connection parameters and table names accordingly.

Now that you've generated a service stub, it's time to configure your send and return types. Keep in mind that you've accomplished all of this, and you still haven't left the Flash Builder design view!

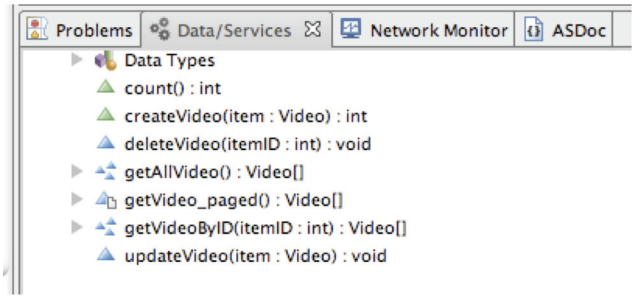


Figure 15.8 Flash Builder automatically sets the send and return data types upon service introspection.

CONFIGURING DATA SEND AND RETURN TYPES

Before you even start configuring data types manually, look at the data typings in your Data/Services panel. If you set the type in your database fields properly, then Flash Builder probably already set your data types for you, as it did for the project that's shown in figure 15.8. This is yet another incredible time-saver!

Your Flash application must know what to expect for the data object types that will be sent and returned for each operation. To configure your data types, either right-click an operation from the Data/Services pane and then click Configure Return Type, or select the method and then click the configure send/return type icon from the toolbar of the Data/Services pane. You should then see a window that looks similar to figure 15.9.

As previously mentioned, you don't have to code your own value objects anymore. You can autogenerate the necessary code when you invoke the `getAllItems()` method. In the past, when an array of strongly typed objects came back in the response from the server, it was standard practice to type the data as a bindable `ArrayCollection` declared at the top of the class and then bind the list or data grid to that `ArrayCollection` variable. You can still do things this way if you want to, but it's worth letting Flash Builder handle this for you because it makes the workflow so much faster. Figure 15.10 demonstrates how to enable autodetection of data return types.

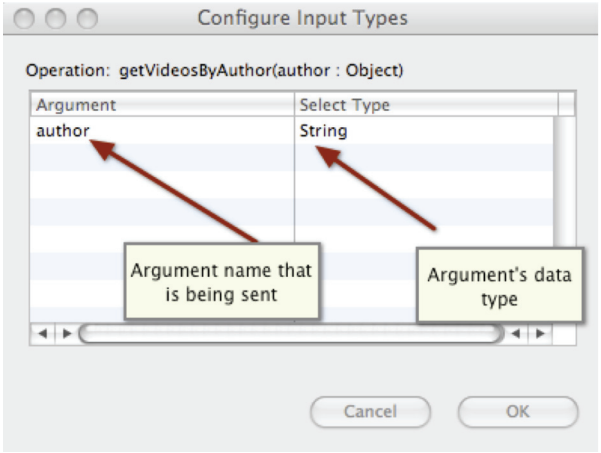


Figure 15.9 In the Configure Input Types window, you configure the parameters that will be sent with the request when the operation is invoked.

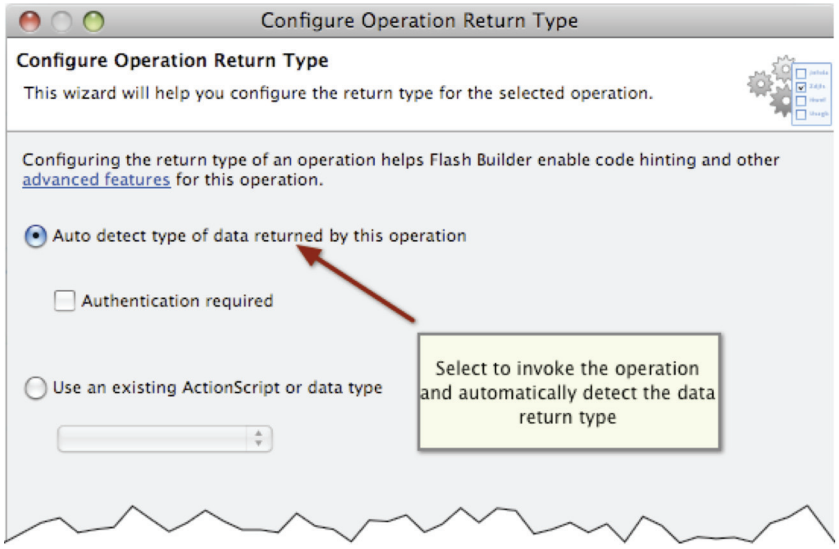


Figure 15.10 Return types can be automatically detected by letting Flash Builder invoke the operation.

The value objects that Flash Builder generates are more like value objects on steroids, in that they do a whole lot more than wrap a bunch of values for strong data typing between the client and server. For example, the return type set for `getAllItems()` is a `User` object, even though it's a collection of `User` objects. This is because the autogenerated result handlers are smart enough to know the difference between a single `User` object that's returned versus a collection of `User` objects. You don't have to type the result to an `ArrayCollection` and go through the whole typing, binding, and object-mapping process as you would have done with the old Flex Builder. Things like class mapping also made the process even more difficult. Luckily, that's all in the past.

So far you've been creating this data-centric application without leaving the design view of the Flash Builder IDE, which is pretty neat. Continuing with this theme, you'll learn how to perform drag-and-drop data binding in a moment. But before we move into drag-and-drop data binding, we'll take this opportunity to build on your data services skill set and knowledge base by showing you how to work with some of the other server technologies available.

It's important to know how to work with a diverse range of server-side technologies because you never know what you'll be working with on the server side for your next project. For example, in the last three months alone, the projects I've been involved with include WebOrb AMF for C#.NET, BlazeDS, LCDS, Tomcat/JBoss, Zend_AMF for PHP, AMFPHP, AMF.NET/C#, and a custom AMF server framework written in C++ for its increased multithreading capabilities. The more server-side technologies you can integrate your Flex applications with, the less you'll ever have to worry about finding work.

15.4 Data-centric Flex with ColdFusion

If you're already a PHP developer and are curious about ColdFusion, Flash Builder 4 makes it especially easy to get started. Adobe has finally added a fully supported development tool specifically for ColdFusion development to its IDE palette, called CFBuilder. From a Flex development standpoint, the coolest thing about CFBuilder is that, like Flash Builder, CFBuilder is built on Eclipse. This means that CFBuilder can be installed right into your Flash Builder IDE as a plug-in, as shown in figure 15.11, so you can handle both the client-side and server-side development of your RIA applications without leaving your primary IDE. There's never been a better time to start using ColdFusion with Flex than now.

To set up a new Flex project for use with ColdFusion services, select ColdFusion for the Application Server Type in on the first screen of the New Project Setup Wizard, and choose the Flash Remoting radio button. You'll then be prompted to set up the server configuration as you did in the previous example.

The purpose of figure 15.12 is to reiterate the point that the process of getting set up with data services is generally the same regardless of server technology. Figure 15.12 should look familiar because it's nearly identical to figure 15.5, where you configured and validated a PHP server. In this case, however, you're configuring the project for use with ColdFusion data services.

After completing the New Project Setup Wizard, select Connect to Data/Service, as you did earlier. The next pop-up window will look like figure 15.13, where you're presented with the option of setting up a ColdFusion service (Flash Remoting, in this case), `HTTPService`, or `WebService`.

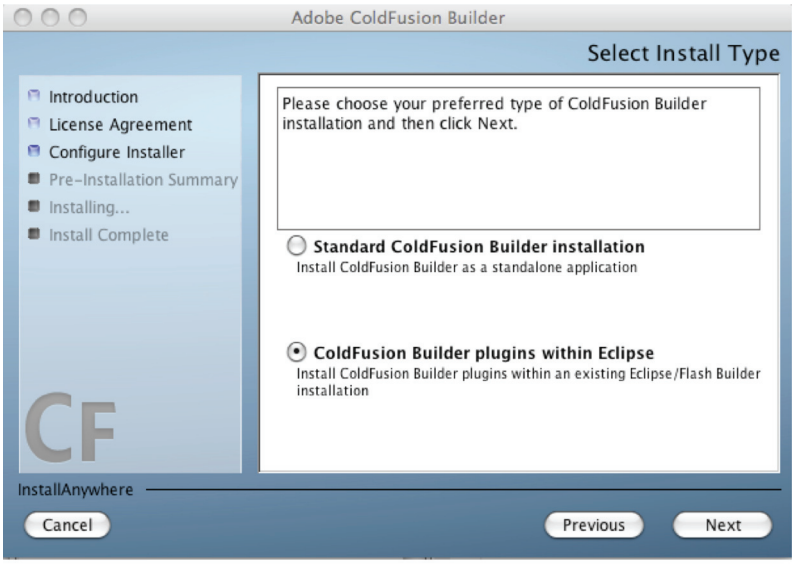


Figure 15.11 Adobe CFBuilder can be installed as a plug-in to your Flash Builder 4 IDE.

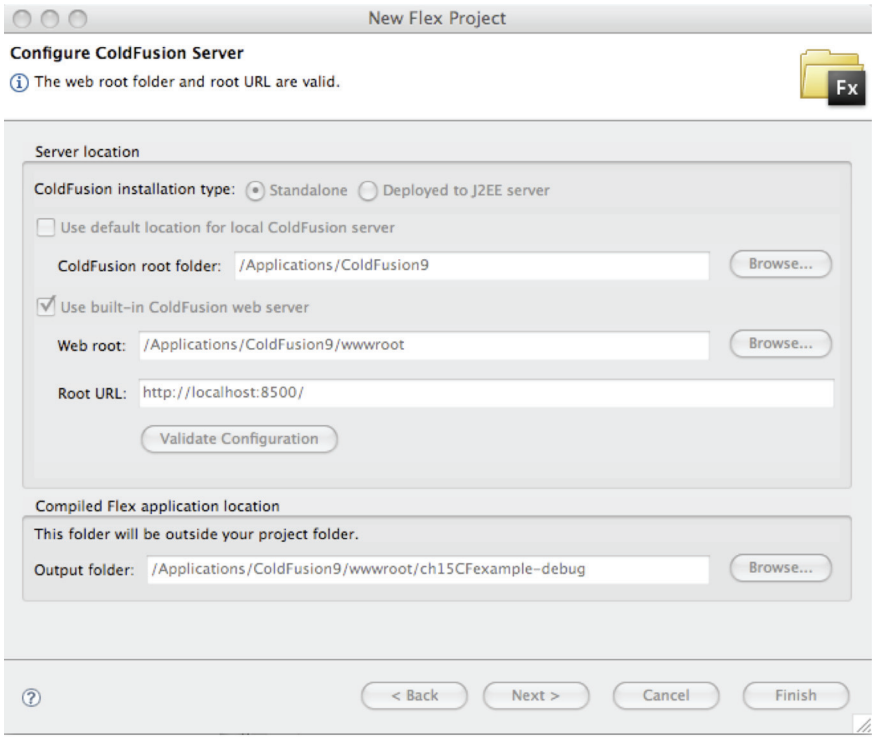


Figure 15.12 **Configuring a Flex project for ColdFusion data services**

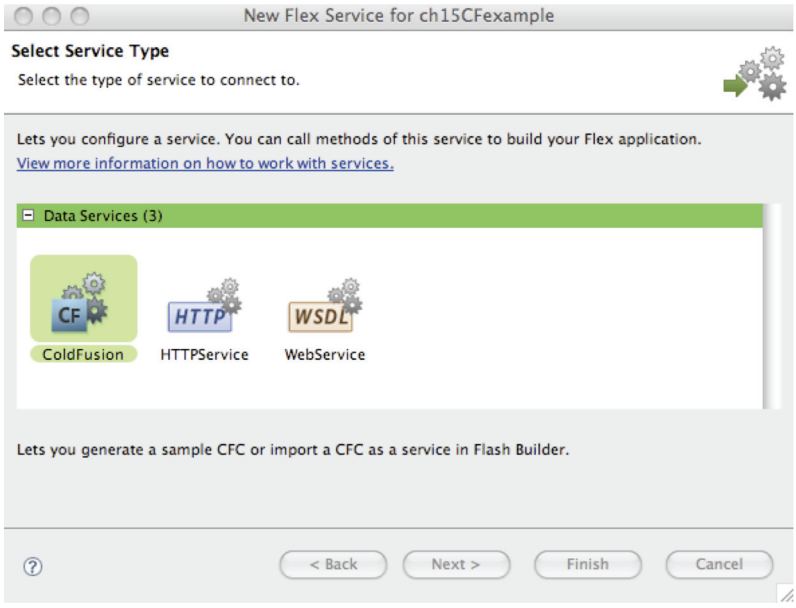


Figure 15.13 **Selecting ColdFusion in this case lets you take advantage of the built-in Flash Remoting.**

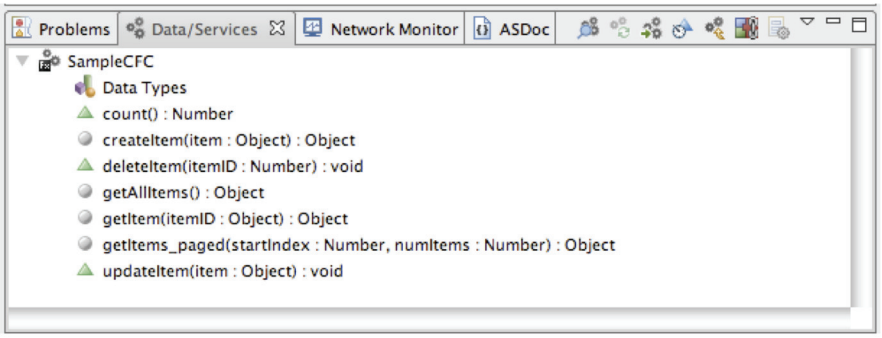


Figure 15.14 The stub operations show up in the Data/Services window, ready for drag-and-drop binding.

The rest of the service setup process is the same as for the previous example, and as shown in figure 15.14, the operations that are available once the service stub is generated are also the same.

Now that you've taken a look into ColdFusion-specific data-centric development with Flash Builder, let's take a quick look into data-centric Java EE with BlazeDS before we move on to drag-and-drop data binding.

15.5 Data-centric Flex with Java EE and BlazeDS

For development of the client-side Flex application, setting up a Flex project for use with J2EE web applications is as simple as selecting J2EE from the Application Server Type drop-down menu when prompted with the first window of the New Project Setup Wizard, as shown in figure 15.15.

After selecting J2EE, you're given the option of using either LiveCycle Data Services or BlazeDS. For this example, choose BlazeDS because it's powerful, the code is open source, and, best of all, it takes only about 15 minutes to get up and running with it.

SETTING UP BLAZEDS

Release builds of BlazeDS come in three flavors: Turnkey, Binary Distribution, and Source. The quickest way to get up and running is to deploy the Turnkey download. Release builds of BlazeDS are available from the Adobe Open Source website at http://opensource.adobe.com/wiki/display/blazeds/Release+Builds. In addition, Adobe evangelist Sujit Reddy G has a great post on setting up BlazeDS at http://sujitreddyg.wordpress.com/2009/04/07/setting-up-blazeds/. The BlazeDS documentation that's provided on the Adobe Open Source website also includes step-by-step instructions.

The moment you've been waiting for has finally come. It's time to tie together everything you've learned and make something useful!

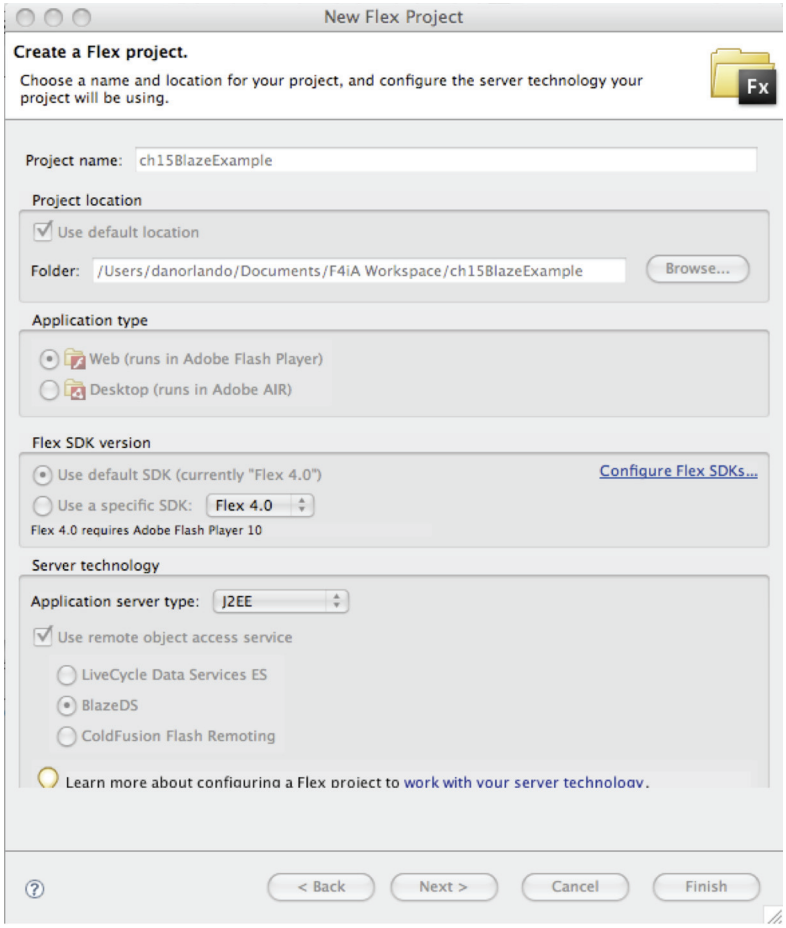


Figure 15.15 **Setting up a new Flex project for use with J2EE and BlazeDS**

15.6 Binding the model to the view

A data model is no good without a means of visualizing it. In the MVC design pattern, the model is the collection of data that's retrieved from the server side by invoking operations from a service, the view is the display of such data, and the controller is the code responsible for binding the model and the view together. In theory, this sounds great, but in practice, writing all of the code for this can be tedious and boring. Enter drag-and-drop data binding.

15.6.1 Drag-and-drop data binding

One of the coolest things about Flash Builder 4 is the ability to drag and drop operations from the Data/Services panel onto list-based components in design view,

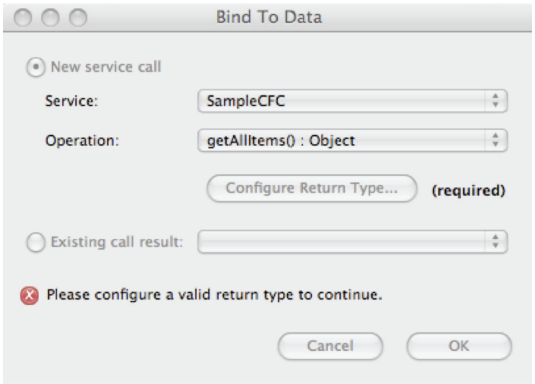


Figure 15.16 **If you see this window, you need to go back and configure your return types.**

effectively creating a binding between the respective operation and the component you dropped it on.

THREE SIMPLE STEPS TO DATA BINDING

Make sure you're in design view with your main application MXML file selected in the main window.

1. First, grab a `VDividedBox` and drag it to the stage. Set its `X` and `Y` values both to `0`, and set the `height` and `width` properties to `100%`.
2. Next, grab a `DataGrid` component from the components list and drag it to the stage so that it's inside the `VDividedBox`. Set its `X` and `Y` values both to `0` as well. Then set its `width` to `100%` and its `height` to `30%`.
3. Now select the `getAllItems()` method in the Data/Services window and drag it onto the `DataGrid`.

If you did not set the return type for your methods yet, you'll see a pop-up that looks similar to figure 15.16. Otherwise, you should immediately see the column headers autopopulate with the field names, and the number of columns should change based on the number of fields you have.

Assuming you followed the instructions in the previous section on connecting to a local database and you have data in your database, you should be able to run the application at this point. The `DataGrid` will display the data from your database when the application is initialized.

What if it didn't work?

A good way to pinpoint errors is to go back and invoke the methods through the return type configuration for each operation again. The error responses that come back are surprisingly detailed and will usually tell you exactly what the problem is and where it occurred (I managed to find and fix a couple of forgotten semicolons in PHP in less than a minute this way!). The Network Monitor will also come in handy during your experimentation endeavors, which we discuss shortly.

15.6.2 Generating a Master-Detail form

One thing is for sure: Drag-and-drop data binding is ultra cool, but generating a Master-Detail form without having to write a single line of code is even cooler!

Start by right-clicking your `DataGrid` component in design view, and select Generate Details Form. As demonstrated in figure 15.17, the window that's displayed should have Master-Detail selected as well as the Make Form Editable check box selected. Uncheck this box if you want the details of the selected item to be displayed as text fields, but you don't want the values of the selected item to be editable.

Next, select the check box labeled Make a New Service Call to Get Details. Make sure that the correct Service is selected, and for the Operation menu, select the `getObjectByID()` method, as shown in figure 15.17. Then click Next. The next window displayed should look similar to figure 15.18.

The Property Control Mapping window that's displayed (figure 15.18) provides you with the opportunity to unselect any fields that you don't want shown in the Master-Detail form. You can also leave a field selected and select Text for the control, which means it will display but won't be an editable item. A unique identifier field is a good example of this type of situation, which is illustrated in figure 15.18, where video_id is the unique identifier for each record. If you've been following along with a specific project of your own, you should be able to run it at this point with all the features and functionality available. Remember, *you accomplished all of this without even leaving the Flash Builder design view!*

Figure 15.17 Be sure to use the correct configuration settings for your Master-Detail form.

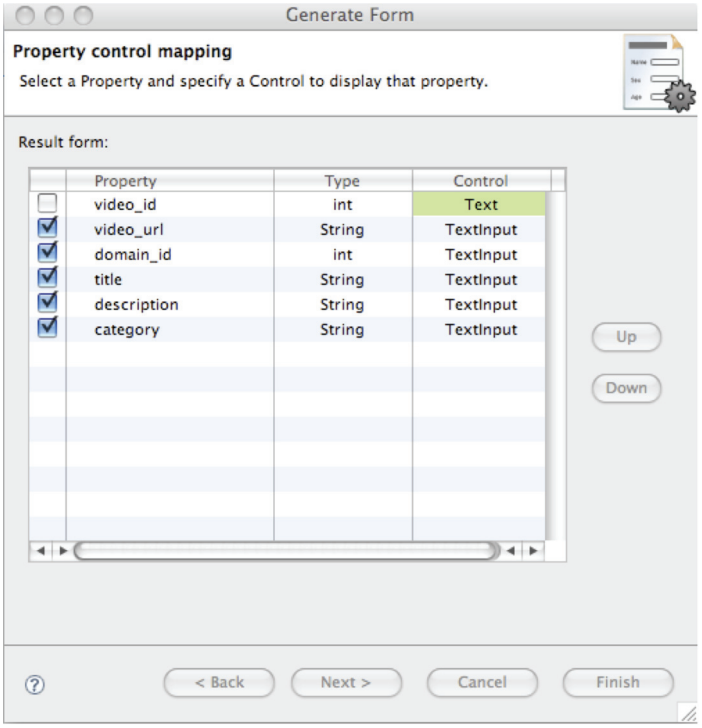


Figure 15.18 Property control mappings of the Generate Master-Detail Form Wizard

Those of you who are fairly new to Flex will certainly appreciate this, but veteran Flex programmers may want to dig a little deeper and find out what's going on behind the scenes, which is what we do in the next section.

15.6.3 Flash Builder code review

You accomplished a lot without looking at any of the code that was being generated behind the scenes, so now might be a good time to check up on Flash Builder and make sure it's still writing quality code. Listing 15.4 is the full set of code that should have been generated in the main MXML application file that you were working in this whole time.

Listing 15.4 DCD code generated by Flash Builder in main application file

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/halo"
               minWidth="1024" minHeight="768"
                xmlns:videoservice="services.videoservice.*"
                xmlns:valueObjects="valueObjects.*">
  <fx:Script>
    <![CDATA[
      import mx.events.ListEvent;
      import mx.events.FlexEvent;
```

```
    import mx.controls.Alert;

    protected function
       dataGrid_creationCompleteHandler(event:FlexEvent):void
    {
      getAllVideoResult.token = videoService.getAllVideo();
    }

    protected function
      dataGrid_changeHandler(event:ListEvent):void
    {
      getVideoByIDResult.token =
        videoService.getVideoByID(dataGrid.selectedItem.video_id);
    }

  ]]>
</fx:Script>
<fx:Declarations>
  <s:CallResponder id="getAllVideoResult"/>
  <videoservice:VideoService id="videoService"
    fault="Alert.show(event.fault.faultString + '\n'
    + event.fault.faultDetail)"
    showBusyCursor="true"/>
  <valueObjects:Video id="video"
    domain_id="{parseInt(domain_idTextInput.text)}"/>
  <s:CallResponder id="getVideoByIDResult"
     result="video = getVideoByIDResult.lastResult[0] as Video"/>
</fx:Declarations>
<mx:DataGrid x="10" y="10" id="dataGrid"
    creationComplete="dataGrid_creationCompleteHandler(event)"
    dataProvider="{getAllVideoResult.lastResult}"
    change="dataGrid_changeHandler(event)">
  <mx:columns>
    <mx:DataGridColumn headerText="video_id"
        dataField="video_id"/>
    <mx:DataGridColumn headerText="video_url"
        dataField="video_url"/>
    <mx:DataGridColumn headerText="domain_id"
        dataField="domain_id"/>
    <mx:DataGridColumn headerText="title"
        dataField="title"/>
    <mx:DataGridColumn headerText="description"
        dataField="description"/>
    <mx:DataGridColumn headerText="category"
        dataField="category"/>
  </mx:columns>
</mx:DataGrid>
<mx:Form>
  <mx:FormItem label="Video_url">
    <s:TextInput id="video_urlTextInput"
        text="@{video.video_url}"/>
  </mx:FormItem>
  <mx:FormItem label="Domain_id">
    <s:TextInput id="domain_idTextInput"
       text="@{video.domain_id}"/>
  </mx:FormItem>
```

Function for populating the `DataGrid`

Function for when `selectedItem` is changed

Manual code modification for selected item ❶

Service declarations placed within `<fx:Declarations/>`

Begin `DataGrid` code

Begin Master-Detail form code

```
    <mx:FormItem label="Title">
      <s:TextInput id="titleTextInput"
         text="@{video.title}"/>
    </mx:FormItem>
    <mx:FormItem label="Description">
      <s:TextInput id="descriptionTextInput"
         text="@{video.description}"/>
    </mx:FormItem>
    <mx:FormItem label="Category">
      <s:TextInput id="categoryTextInput"
         text="@{video.category}"/>
    </mx:FormItem>
  </mx:Form>

</s:Application>
```

The code generated by Flash Builder wasn't too shabby. One thing had to be manually changed though. The change handler function ❶ for the data grid needed the video ID value set manually from `itemID` to `dataGrid.selectedItem.video_id`.

If you're paying attention, you may notice something else missing as well. There isn't a control or a function for updating the selected item! You should still be able to run the application at this point, but changing values in the Master-Detail form won't have any effect on the database record.

To address this, switch back to design view, drag a button component to the stage, then drag the `updateVideo()` method from the Data/Services panel to it, and voila! Congratulations, you've completed your Flash Builder DCD CRUD service application!

15.7 Summary

In this chapter you learned about the many ways of communicating with the outside world from your Flex 4 applications. You also gave the new Flash Builder 4 data-centric development features a spin and learned how to speed up your workflow ten-fold by doing so.

Although the code-generating features for data-centric development with Flash Builder are incredibly useful and can save huge amounts of time during development, you should be aware of a couple of caveats as a conscientious developer. First, at the time of this writing, the DCD features don't support any of the microarchitecture frameworks for enterprise Flex development (discussed in chapter 19). It's worth noting, however, that there's a good possibility we'll see third-party plug-ins that add DCD support for use with the Swiz and Mate frameworks in the not-so-distant future. The second caveat is the general idea of generated code. In theory, it sounds like the be-all end-all to programming in general, but in practice, this is hardly the case. It's important to analyze and review any code that has been generated for you by the IDE and make sure it's consistent with what you're trying to accomplish.

In the next chapter, you'll learn about the concepts of objects and classes, which will be a nice lead-in for chapter 17 on building custom components with Flex 4.

WEB DEVELOPMENT

FLEX 4 IN ACTION

Ahmed • Orlando • Bland • Hooks

Flex has grown from just a way to build Flash apps into a rich ecosystem, and Flex 4 introduces new UI components, better performance monitoring, and speed enhancements to the compiler.

Flex 4 in Action is a comprehensive tutorial that introduces Flex to web designers and developers. It starts with the basics—forms and data—and moves through core concepts like navigation, drag-and-drop, and events. Even if you're new to Flex, this book is all you'll need to make your apps pop using the new Spark components, data services, charting, special effects, and more.

What's Inside

- How to architect your applications
- Use charting to build interactive dashboards
- Improve productivity with network monitoring and unit testing
- Give your apps a unique look with themes and skins
- And much more

Readers of this book need basic development skills, but no previous experience with Flex.

Tariq Ahmed is an RIA engineer and Flex community evangelist. **Dan Orlando** is an RIA architect, specializing in Flex and AIR. **John C. Bland II** is an independent Flex, ColdFusion, and mobile developer. **Joel Hooks** is a Flash Platform developer and ActionScript expert.

For online access to the authors and a free ebook for owners of this book, go to manning.com/Flex4inAction

"*The* desk reference for all things Flex 4"
—John Griffin, Overstock.com

"No question is left unanswered, no facet unexplored."
—Peter Pavlovich, Kronos Inc.

"A great book for both beginners and experienced Flex developers."
—Kevin Schmidt
Adobe Systems, Inc.

"The lessons are memorable, witty, and very relevant."
—Zareen Zaffar, Amcom

"Completely demystifies building rich user interfaces."
—Rick Wagner, Acxiom Corp.

"What you need to be flexible 4 your job!"
—Rick Evans, SAS

ISBN 13: 978-1-935182-42-9
ISBN 10: 1-935182-42-0
54999
9 781935 182429

MANNING

$49.99 / Can $57.99 [INCLUDING eBOOK]