Get More Refcardz! Visit refcardz.com

DZone Refcardz

Adobe

CONTENTS INCLUDE:

Getting Started with BlazeDS

By Shashank Tiwari

ABOUT BLAZEDS

Adobe BlazeDS is an open source software that facilitates effective integration of Flex and Java. It enables remote procedure calls and message exchanges between the two platforms thereby helping couple together rich and engaging Flash platform based interfaces and robust enterprise servers. Being open source, BlazeDS is freely available and can be downloaded from http://opensource.adobe.com/wiki/display/blazeds/BlazeDS/.

This Refcard provides a quick overview of BlazeDS. It attempts to illustrate some of the most important features of the software and therefore acts as a starting point for developers who are interested in the subject. More details, with exhaustive sets of examples, are available in the book *Professional BlazeDS* (Wiley/Wrox, 2009).

WHAT EXACTLY IS BLAZEDS?

Although it's mentioned upfront that BlazeDS helps connect Flash platform applications to Java, it's important to define it a bit further. Therefore both its behavioral and structural aspects are tersely listed in this section.

Behavioral Definition

BlazeDS enables and facilitates:

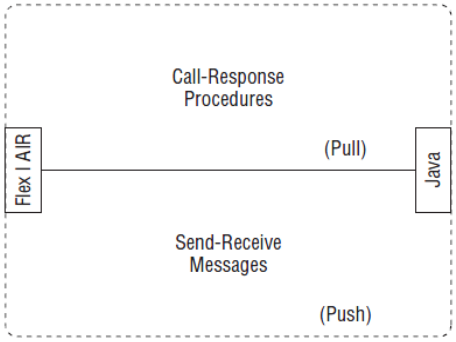
- Invocation of remote Java methods from a Flex application.
- Translation of Java objects returned from a server, in response to the remote method call, to corresponding AS3 objects.
- Translation of AS3 objects sent by a Flex application to corresponding Java objects for passing them in as method call arguments.
- Communication between a Flash Player instance and a Java server over a TCP/IP-based application-level binary protocol.
- Near real-time message passing between a Flex application and a Java server.
- Management of the communication channels between Flex and Java.
- Management of connection types between Flex and Java.
- Provision for adapters for communication with server-side Java artifacts like JMS queues, and persistence layers like Hibernate and JPA. (Some of these are in-built, and some can be obtained from open source projects or can be custom built.)
- Pushing data from the server to the client on the server's initiative and not as a response to a request.

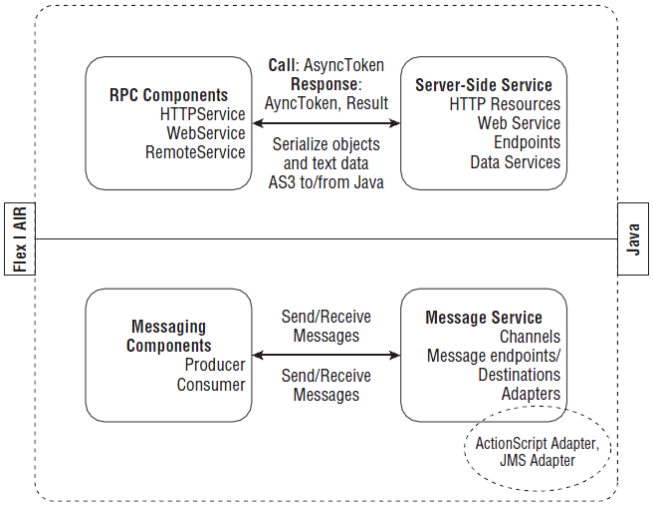
Structural Definition

BlazeDS is a:

- Java web application that leverages the Java Servlets specification.
- Web application that runs within a Java Servlet container or a Java application server, for example Apache Tomcat, JBoss AS, IBM Websphere or BEA (now Oracle) Weblogic.
- Set of services that can be managed using JMX agents.
- Remoting and messaging program that can be extended by using its Java API.
- Program that intercepts all communication between a Flash Player and a Java server.
- Configurable web application that can be clustered and used in cases that desire a higher than normal performance. (The in-built data push mechanism has a few limitations as far as high throughput and high volume goes but there are ways to get around this shortcoming.)

With the definition of BlazeDS firmly in place, it's worthwhile to explore Flex client and Java server integration in the larger context of combining the two platforms. Look at the next two figures for some insight into the context.





INSTALLING BLAZEDS

Installing BlazeDS is as simple or as complex as deploying a web application in a Java servlet container. BlazeDS works seamlessly in most mainstream open source and commercial Java servlet containers, including JBoss, Jetty, Tomcat, Oracle Weblogic and IBM Websphere.

You can get both compiled and source versions of the software. In addition, one of the binary versions comes in the form of a turnkey distribution that includes a configured copy of the Apache Tomcat Servlet container within the bundle. For beginners, it's convenient, appropriate and advisable to get the latest release version of the binary turnkey distribution.

CONFIGURING BLAZEDS

Simply put, at the heart of BlazeDS is a Servlet that bootstraps the infrastructure that intercepts all calls between a Flex client and the BlazeDS instance. This Servlet, called MessageBrokerServlet, uses artifacts like channels, endpoints, and adapters to enable proxy, remoting, and messaging services. A default configuration file, called services-config.xml, which lies in the WEB-INF/flex folder of the BlazeDS installation, defines these artifacts and their relationships in the context of the MessageBrokerServlet.

Channels and endpoints connect a Flex client to a BlazeDS server. They are the primary components that enable communication between these two entities. Endpoints reside at the BlazeDS end. Flex clients use channels to connect to these endpoints.

The BlazeDS endpoints are Servlet-based endpoints. Each endpoint defines a type and format of communication. For example, endpoints exist for simple AMF data exchange, polling AMF, and AMF data streaming.

Analogously, the Flex client defines a set of channels that vary depending on the type and format of communication. For example, the HTTPChannel facilitates communication over non-binary AMF format, AMFX (AMF in XML), and the AMFChannel enables standard binary AMF-based communication.

Matching endpoints and channels are paired, and that's when a Flex client and BlazeDS server talk to each other. The binding of channels and endpoints to their implementation classes and their pairing is done in the services-config.xml configuration file.

In addition to the endpoints, BlazeDS includes adapters that provide the critical compatibility between the core BlazeDS Servlet that talks to Flex and a server-side resource such as a JMS resource, a persistent data store, or an Object-Relational mapping layer. Adapters are also configured in services-config.xml.

The configuration file services-config.xml, in BlazeDS, is logically split into four configuration files.

The top level and the first of these four is services-config.xml itself. The other three are as follows:

- remoting-config.xml
- proxy-config.xml
- messaging-config.xml

Remoting-config, proxy-config and messaging-config contain configuration pertaining to remote procedure calls, proxy services and message services respectively.

Explaining every bit of the configuration is beyond the scope of this Refcard and is therefore not included. Only a couple of quick examples are shown to give you a flavor the typical configuration elements. Detailed explanations are available in the book entitled Professional BlazeDS (Wiley/Wrox, 2009), which I spoke about earlier.

A channel can be configured as follows:

```
<channel-definition id="my-amf" class="mx.messaging.channels.
AMFChannel">
<endpoint
url="http://{server.name}:{server.port}/{contex.root}/messagebroker/
amf"
class="flex.messaging.endpoints.AMFEndpoint"/>
</channel-definition>
```

An adapter can be configured as follows:

```
<adapters>
<adapter-definition id="java-object"
class="flex.messaging.services.remoting.adapters.JavaAdapter"
default="true"/>
</adapters>
```

With these brief configuration examples in place, let's explore BlazeDS's pull-based (or request-response based) communication abilities. The following sections include a few more in context configuration illustrations.

PULL-BASED COMMUNICATION

Off the shelf, the Flex framework includes three methods of pull-based communication and data interchange with external data sources:

- HTTP request-response
- Web services
- Remote procedure calls involving objects

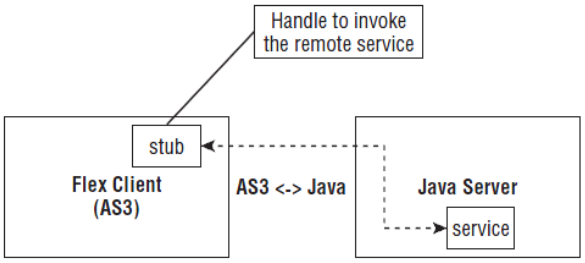
A possible remoting configuration could be as follows:

```
<service id="aRemotingService"
class="flex.messaging.services.RemotingService">
<adapters>
<adapter-definition id="java-object"
class="flex.messaging.services.remoting.adapters.JavaAdapter"
default="true"/>
</adapters>
<default-channels>
<channel ref="myAMFChannel"/>
</default-channels>
<destination id="myPOJODestination">
<properties>
<source>myPackage. MyPOJO</source>
<scope>session</scope>
</properties>
</destination>
</service>
```

Using BlazeDS's remoting you are enabled with automatic translation between the Java and ActionScript3(AS3) data types, marshalling, un-marshalling, serialization and de-serialization across the endpoints and channels.

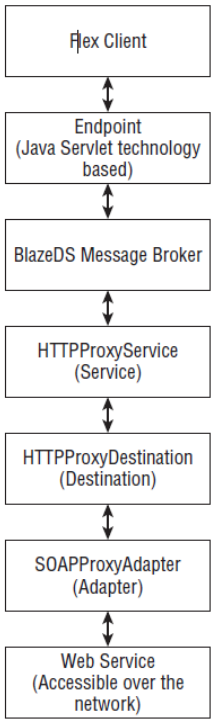
BlazeDS's remoting capabilities provide fast and efficient data transmission between a Flex client and a Java server with the help of the binary **Action Message Format (AMF)** protocol and the built-in endpoints, channels and adapters to support it. Adapters make it possible to hook-up specific server side entities. The JavaAdapter included in the configuration example, is a built-in adapter for plain Java objects, which are also sometimes referred to as POJOs.

For accessing managed entities like Spring Beans or Enterprise Java Beans (EJBs), you can use the JavaAdapter as the translator but you also need a custom factory to help you access these objects, as managed objects reside in a namespace separate from the one that BlazeDS uses to instantiate its own objects.



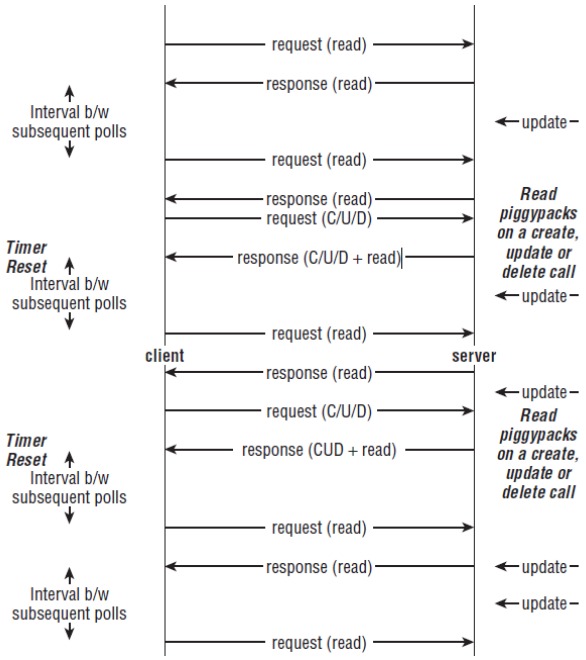
Besides, remoting BlazeDS can also act as a proxy server and help access data from domains that are not explicitly trusted via a crossdomain.xml security definition. Among others, the BlazeDS proxy capabilities have three important use cases including:

- Access control to proxy destinations
- Logging of all proxy traffic
- Handling of HTTP errors



COMMUNICATING IN REAL-TIME

Data pushing in a web application context implies the server sending data to a browser-based client without the client necessarily requesting it.

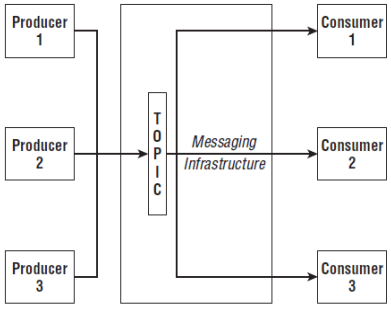


Long polling can provide near real-time data push by waiting till the response from the server is ready to be dispatched. It's not scalable though, as it blocks connections.

Direct connectivity over Real Time Messaging Protocol (RTMP) provides a better scalable connectivity over long polling but RTMP is not available with BlazeDS yet! The RTMP specification was proprietary until the beginning of this year and has most recently been opened up to public. Its inclusion into future versions of BlazeDS is anticipated.

Usage of Java NIO provides for scalable connections as blocking connections are replaced by non-blocking asynchronous counterparts. BlazeDS does not include Java NIO implementations for its communication channels but it's not very difficult to include one. The open source dsadapters project (http://code.google.com/p/dsadapters/) aims to provide extensions to BlazeDS that include java NIO based channels, specialized factories, filters and advanced adapters, a lot of which will be available in the near future.

Messaging systems involve two typical messaging domains: point-to-point and publish-subscribe. BlazeDS can logically support both messaging domains. It has first-class support for publish-subscribe messaging domain.



BlazeDS based message services leverage two types of built-in adapters, namely:

- JMSAdapters
- ActionScriptAdapters

The JMSAdapter connects with JMS server side messaging systems. The ActionScriptAdapter helps route messages between Flex clients via the server. Therefore JMSAdapter comes handy when Flex clients are wired up to send and receive messages to and from enterprise systems that use JMS, whereas ActionScriptAdapter is useful for building systems like chat applications to help communicate between two Flex clients.

For effective near real-time messaging use one of the following channels:

AMFChannel/Endpoint with Polling
The client sends a recurring request to the server at a pre-defined frequency. Polling is very resource and network intensive.

AMFChannel/Endpoint with Long Polling
The channel issues polls to the server to fetch data but if no data is available it waits until data arrives for the client or the configured server wait interval elapses. For high frequency updates this configuration has the overhead of a poll roundtrip for every pushed message and therefore messages can be queued between polls. The Servlet API, prior to its current version 3, utilizes blocking I/O in which case long polling soon exhausts connections as multiple concurrent clients utilize the channel.

Streaming AMFChannel/Endpoint
This channel opens an HTTP connection between the server and client, over which the server sends an unending response of messsages. This channel avoids the overhead of polling and keeps the connection open for the entire scope of communication between the client and the server. Like AMFChannel with long polling, the constraints of blocking Servlet I/O can exhaust connections as multiple concurrent clients utilize the channel.

JPA AND HIBERNATE WITH FLEX

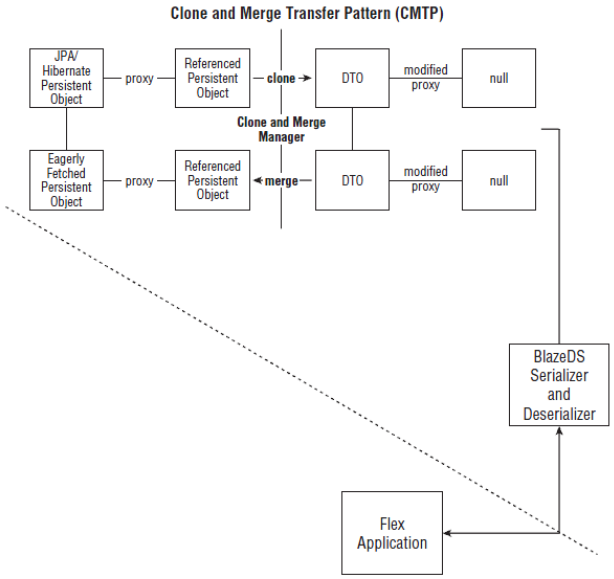
Object Relational Mapping (ORM) and persistence frameworks like JPA and Hibernate facilitate effective handling of relational data sources with Java applications. Extending the scope of these frameworks to include interactions with Flex clients allows seamless persistence management.

Off-the-shelf BlazeDS has no special features to support JPA and Hibernate. However a number of open source projects provide adapters to combine JPA and Hibernate with Flex. Gilead (http://noon.gilead.free.fr/gilead/) and dphibernate (http://code.google.com/p/dphibernate/) are two such open source adapters. The dsadapters project (http://code.google.com/p/dsadapters/) is also in the process of releasing a robust open source JPA/Hibernate adapter.

BlazeDS is capable of serializing and transforming Java-based objects to their AS3 counterparts so that may make you wonder why wiring up JPA and Hibernate entities and their collections needs any special handling. The reason for this special need arises because of the way BlazeDS serializes data across the wire.

BlazeDS has a set of endpoints where a Flex application channel sends requests up to BlazeDS that resides within a Servlet container or an application server. Responses from BlazeDS follow the route back up from the endpoint to the channel. On endpoints that support translation and serialization between AS3 and Java (or even web services), a serialization filter is defined to intercept calls to the endpoint. When an incoming or outgoing message hits the filter, serialization and deserialization occur. During serialization, the serializer eagerly fetches all the JPA and Hibernate persistent objects and sends them across the wire. The JPA and Hibernate proxies are replaced with the data that they stand in place of. This breaks the lazy loading semantics, rendering the idea of proxies useless. Therefore, any Hibernate adapter needs to preserve the proxy characteristics while keeping the standard behavior of the essential serialization and de-serialization mechanism between Flex and Java intact.

A common design pattern used to solve this problem is what I like to call the "Clone and Merge Transfer Pattern."



INTEGRATING WITH SPRING FRAMEWORK

The Spring BlazeDS integration project, also called Spring Flex, is a joint effort between SpringSource and Adobe to create a version of BlazeDS that works seamlessly with the Spring Framework. Prior to the existence of this project developers relied on custom factories to integrate Spring and BlazeDS.

BlazeDS is a Java Servlet based web application, so it integrates and works with the Spring framework facility that addresses the web layer. The core Spring framework web layer implements an MVC framework and is called Spring MVC. Spring BlazeDS integrates smoothly with Spring MVC.

A MessageBroker component sits at the heart of BlazeDS. All messages in BlazeDS are routed through the MessageBroker. In standard BlazeDS deployments, the MessageBrokerServlet is declared in WEB-INF/web.xml as follows:

```
<servlet>
<servlet-name>MessageBrokerServlet</servlet-name>
<servlet-class>flex.messaging.MessageBrokerServlet</servlet-class>
<init-param>
<param-name>services.configuration.file</param-name>
<param-value>/WEB-INF/flex/services-config.xml</param-value>
</init-param>
<init-param>
<param-name>flex.write.path</param-name>
<param-value>/WEB-INF/flex</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
<servlet-name>MessageBrokerServlet</servlet-name>
<url-pattern>/messagebroker/*</url-pattern>
</servlet-mapping>
```

In Spring MVC, all incoming requests are handled by a Front controller Servlet, called the DispatcherServlet. This means you must have a way that all requests coming from the Flex client and intended to be handled by the BlazeDS instance are routed by the DispatcherServlet to the MessageBroker. This is exactly what the Spring BlazeDS project implements.

In addition, the Spring BlazeDS project also defines an XML Schema and associated infrastructure so that BlazeDS artifacts can be configured within Spring configuration files as Spring managed objects.

As an example a default BlazeDS message broker configuration can then be like so:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:flex="http://www.springframework.org/schema/flex"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/flex
http://www.springframework.org/schema/flex/spring-flex-1.0.xsd">
<flex:message-broker/>
</beans>
```

More about the possible configurations can be learned online at http://www.springsource.org/spring-flex

With the help of this project it becomes easy to configure Spring Beans as server side remoting counterparts of Flex clients.

For example a service class configured as a Spring Bean as follows:

```
<bean id="myService" class="myPackage.MyService" />
```

Can easily be used as a remoting destination simply by specifying a configuration as follows:

```
<flex:remoting-destination ref="myService" />
```

There is plenty more, including use of annotations, possible for configuring Spring Beans as remoting destinations and you may want to learn more online from the Spring BlazeDS project site mentioned earlier in this paragraph.

Besides, remoting the Spring messaging and security benefits also get extended to Flex applications. In a Spring BlazeDS server, three types of message service components can interact with the Flex message service. The messaging service in Flex itself is agnostic to the messaging protocol used on the server side. Therefore, multiple server-side messaging alternatives easily work with Flex messaging. The three alternative server-side message services in Spring BlazeDS are:

- Native built-in BlazeDS AMF messaging
- JMS messaging using the Spring-spec JMS components
- Spring BlazeDS messaging integration

EXTENDING BLAZEDS

BlazeDS exposes a public Java API. You can leverage that API to customize the behavior of BlazeDS resources.

The flex.messaging.services.ServiceAdapter abstract class sits at the root of the hierarchy. All built-in adapter classes inherit from the ServiceAdapter abstract class. Abstract classes not only define a contract like interfaces do but also define behavior through partial method implementations. The ServiceAdapter class states the base behavioral characteristics of a BlazeDS adapter.

You can create custom BlazeDS adapters by extending either the ServiceAdapter or one of its subclasses like the JavaAdapter or the MessagingAdapter classes.

MAKING BLAZEDS APPLICATIONS SCALABLE

In web scale applications scalability is an important criteria for success. BlazeDS instances can be clustered and the following techniques can be applied to make applications scale better:

Clustering

Clustered instances of BlazeDS share data and messages across instances. Therefore, message producers connected to a clustered server instance can send messages to message consumers connected to a different server within the same cluster. Besides sharing state information and routing information through the clustered servers, clustering provides for channel failover support. Clustered of BlazeDS instances is powered by JGroups.

Data compression

BlazeDS is a Java Servlets based web application. As data from a BlazeDS server leaves for the Flex client, it can be intercepted using the familiar Java Servlet filter mechanism. Servlet filters can intercept requests to and responses from a Java Servlet. In order to zip the data transferred from the server to the client, a filter can be created to manipulate and compress the outgoing response.

Data format optimization

AMF3 facilitates a very efficient way of binary transmission of data between the server and the client. In addition, you can choose to go with a text-based format that could be well structured like XML or delimited like comma-separated or tab-delimited text. JSON (JavaScript Object Notation) and AMFX, where AMF is transmitted in XML, are also options.

Robust connection

Non-blocking channels allow for greater number of connections to be served provided they are not all active at the same time always. Life Cycle Data Services uses NIO channels for connection scalability. You can include the same robustness in BlazeDS as well.

Service orientation

Service oriented architecture (SOA) patterns that apply well when creating Flex and BlazeDS applications are:

- Concurrent contracts
- Cross-domain utility layer
- Functional decomposition
- Rules centralization
- State repository

Caching

If the accessed data is not changing during the course of its reuse it always makes sense to cache it. Caching is a time-tested way of increasing performance by avoiding data fetches across the network and using pre-fetched local data instead.

Resource Pooling

Many of these external systems and libraries, such as messaging infrastructure, database connections and stateless business services, lend themselves to pooling. When resources are pooled, they are shared over multiple clients. BlazeDS has acces to all resource pooling strategies that any Java EE web application running in an application server has.

Workload distribution

Distributing work optimally between a client and its server is an important challenge when architecting RIA. BlazeDS remoting services optimally combine a Flex client and a Java server and allow a developer to distribute workload across the wire in ways without necessarily imposing the overheads that loose coupling like XML based interactions over HTTP and web services do.

CONCLUSION

With no upfront costs and a great set of features BlazeDS is a compelling piece of software for most serious Rich Internet Application (RIA) stacks that involve Flex and Java.

ABOUT THE AUTHORS

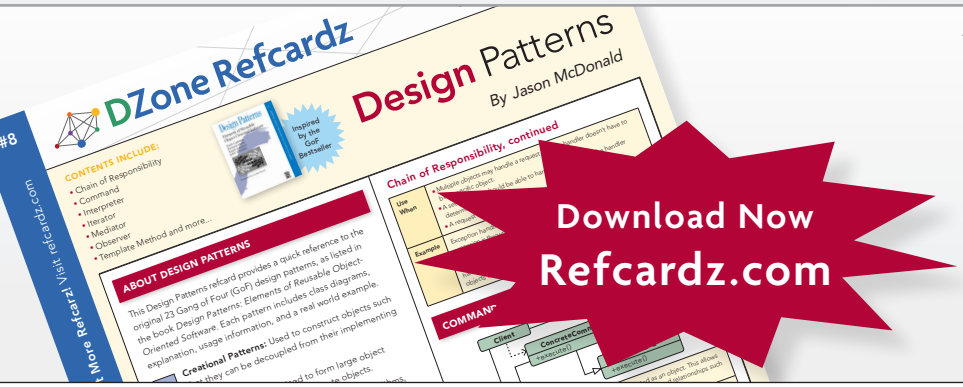
Shashank Tiwari is a Managing Partner & CTO at Treasury of Ideas, a technology driven innovation and value optimization company. As an experienced software developer and architect, he is adept in a multitude of technologies. He is an internationally recognized speaker, author and mentor. As an expert group member on a number of JCP (Java Community Process) specifications he has been actively participating in shaping the future of Java. He is also an Adobe Flex Champion and a common voice in the RIA community. Currently, he passionately builds rich high performance scalable applications and advises many on RIA and SOA adoption. His clients range from large financial service corporations to brilliant startups, whom he helps translate cutting edge ideas into reality. He is also actively engaged in training and mentoring developers and architects in leading edge technology. He is the author of a number of books and articles, including Advanced Flex 3 (Apress, 2008) and Professional BlazeDS (Wiley, 2009). He lives with his wife and two sons in New York. More information about him can be accessed at his website (www.shanky.org).

RECOMMENDED BOOK

This informative resource provides you with detailed examples and walkthroughs that explain the best practices for creating RIAs using BlazeDS. You'll begin with the essentials of BlazeDS and then more on to more advanced topics. Along the way, you'll learn the real-world concerns that surround enterprise-based Java and Flex applications.

BUY NOW
books.dzone.com/books/blazeds

Professional Cheat Sheets You Can Trust



"Exactly what busy developers need: simple, short, and to the point."
James Ward, Adobe Systems

Upcoming Titles
Blaze DS
Domain Driven Design
Virtualization
Java Performance Tuning
Expression Web
Spring Web Flow
BPEL

Most Popular
Spring Configuration
jQuery Selectors
Windows Powershell
Dependency Injection with EJB 3
Netbeans IDE JavaEditor
Getting Started with Eclipse
Very First Steps in Flex

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-90-5
ISBN-10: 1-934238-90-2
50795
9 781934 238905
$7.95

Version 1.0

Copyright © 2009 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.