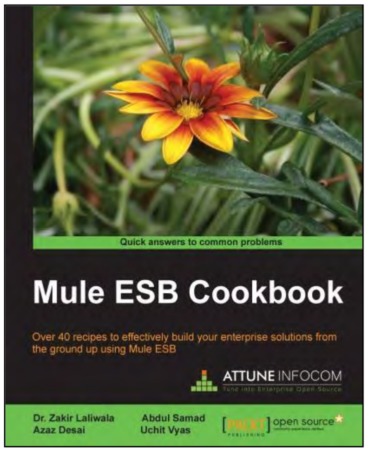
# Mule ESB Cookbook

Dr. Zakir Laliwala
Abdul Samad
Azaz Desai
Uchit Vyas


Quick answers to common problems
Mule ESB Cookbook
Over 40 recipes to effectively build your enterprise solutions from the ground up using Mule ESB
ATTUNE INFOCOM
Dr. Zakir Laliwala
Azaz Desai
Abdul Samad
Uchit Vyas
PACKT open source

## Chapter No. 2

## "Working with Components and Patterns"

## In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.2 "Working with Components and Patterns"

A synopsis of the book’s content

Information on where to buy this book

## About the Authors

**Dr. Zakir Laliwala** is an entrepreneur, open source specialist, and hands-on CTO of Attune Infocom. Attune Infocom provides enterprise-level open source solutions, and services for SOA, BPM, ESB, Portal, Cloud computing, and ECM. At Attune Infocom, he is responsible for solutions and services delivery and product development. He is exploring new enterprise-level open source solutions, and defining architecture, roadmap, and best practices. He has provided consulting and training on various open source technologies to corporates around the world on Mule ESB, Activiti BPM, JBoss' jBPM and Drools, Liferay Portal, Alfresco ECM, Jboss SOA, and Cloud computing.

Dr. Zakir pursued Ph.D. in Information and Communication Technology from Dhirubhai Ambani Institute of Information and Communication Technology. He was the Adjunct Faculty at Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT) and CEPT University, where he now teaches students pursuing Master's.

He has published many research papers in IEEE and ACM international conferences on web services, SOA, Grid computing, and Semantic Web. He also serves as a reviewer at various international conferences and journals. He has contributed chapters on open source technologies and writes books on open source technologies.

**Abdul Samad** has more than seven years' hands-on experience in leading and implementing Java, J2EE, Portal, and ECM open source solutions. He has successfully migrated IBM WebSphere portal to Liferay Portal for a client based in the U.K. He has delivered successful training with his experience and expertise on Liferay Portal and jBPM to Sambaash, AT&T, Cognizant, Urja Technologies, and Protea Technologies. He was part of an enterprise-level, open source portal application implementation for media and publication houses, portal customization projects, and led a team of developers to achieve the client's requirements on time.

He has expertise in implementing J2EE technologies (JSP, Servlet, MVC Frameworks, BPM, ESB, and Portlet frameworks) to develop enterprise web applications. He has worked with various frameworks such as Mule ESB, jBPM, Liferay, Alfresco, and Oracle WebLogic portal on his journey.

**Azaz Desai** has more than three years' experience in Mule ESB, JBPM, and Liferay technologies. He is an Oracle Certified Java Programmer (OCJP). He is responsible for implementing, deploying, integrating, and optimizing services and business processes using ESB and BPM tools. He is a lead writer of *Mule ESB Cookbook*, *Packt Publishing*, as well playing a vital role of trainer on ESB to global clients at Attune Infocom.

He is very enthusiastic and active in understanding client-specific requirements on web service integration. He has done various integration of web services, such as Mule ESB with Liferay, Alfresco, jBPM, and Drools. He was a part of a key project on Mule ESB integration as a messaging system. He has worked on various web service standards and frameworks, such as CXF, AXIS, SOAP, and REST.

**Uchit Vyas** a B.Tech. graduate in Computer Science with a research interest in ESB and Cloud, is a certified Cloud Architect (AWS), Cisco (CCNA), VMware (VSP), and Red Hat Linux (RHCE) professional. He has an energetic strength to work on multiple platforms at a time and the ability to integrate open source technologies. He works as a Sr. Consultant and looks after AWS – Cloud, Mule ESB, Alfresco, Liferay and deploying Portal, and ECM system. He was previously working with TCS as an Assistant System Engineer.

With over three years' hands-on experience on open source technologies, he manages to guide the team and deliver projects and training sessions meeting client expectations. He has provided more than 13 training sessions on Cloud computing, Alfresco, and Liferay in a couple of months. In the last few years, he has moved over 80 percent of Attune Infocom business processes to the Cloud by implementing agile SDLC methodology on Amazon, Rackspace, and private Clouds such as Eucalyptus and OpenStack. His skills are not limited to designing and managing Cloud environment/infrastructure, server architecture. He is also active in Shell scripting, autodeployment, supporting hundreds of Linux and Windows physical and virtual servers hosting databases, and applications with continuous delivery using Jenkins/CruiseControl with Puppet/Chef scripting.

# Mule ESB Cookbook

Mule ESB is a lightweight Java-based enterprise service bus (ESB) and integration platform that allows developers to connect applications together quickly and easily, enabling them to efficiently exchange data. You can therefore use Mule ESB to allow different applications to communicate with each other via a transit system to carry data between applications within your enterprise or across the Internet. It is also useful if you use more than one type of communication protocol while integrating three or more applications/services.

*Mule ESB Cookbook* takes readers through the practical approach of using Mule ESB 3.3. This book solves numerous issues faced by developers working on Mule ESB in real time and provides use cases on how to integrate Mule with other technologies. It also focuses on development and delivery using Mule ESB through integrating, migrating, and upgrading advanced technological tools.

This book gives the reader a strong overview of the Mule framework using practical and easy-to-follow examples. It has three sections: problems, approaches, and solutions. The key aim of this book is to show you how to allow different applications to communicate with each other by creating a transit system to carry data between applications within your enterprise or across the Internet. Mule ESB enables easy integration of existing systems, regardless of the different technologies that the applications use, including JMS, web services, JDBC, HTTP, and more.

*Mule ESB Cookbook* will teach you everything to communicate between applications that are built on different platforms, as well as how to migrate them into your application across multiple platforms or on the Cloud.

## What This Book Covers

This book contains recipes related to deployment, scripting, and the API discussing core concepts of standard components, performance tuning, and Cloud integration through practical task-oriented recipes. This book will provide you practical knowledge of the Mule ESB architecture and its configuration. Core concepts and components required to understand how Mule ESB works are also explained.

*Chapter 1*, *Getting Started with Mule ESB*, discusses Mule core concepts and terminology. It also provides an environment setup for Mule ESB and Mule Studio. By the end of this chapter, you will be familiar with Mule IDE integration with Eclipse, and how to create a Hello World project and flow in Mule Studio. At the end of this chapter, you will learn how to configure Mule elements and deploy applications on the Mule server.

*Chapter 2*, *Working with Components and Patterns*, describes what a component is and its types, such as Echo, Logger, REST, SOAP, HTTP, and Java. You will also know how to configure a component, how to use it in a workflow, and what patterns are in Mule ESB.

*Chapter 3*, *Using Message Property, Processors, and Sources*, helps you understand what message sources, processors, and properties are. By the end of this chapter, you will be able to use processors in a workflow, and use message processors to control the message flow, and message property scopes.

*Chapter 4*, *Endpoints*, explains what an Endpoint is. Endpoints send and receive data and are responsible for connecting to external resources and delivering messages. The two types of Endpoints available in Mule Studio are: Inbound Endpoint and Outbound Endpoint. Inbound Endpoint is used for receive messages and Outbound Endpoint is used for sending messages.

*Chapter 5*, *Transformers*, explains what a transformer is. By the end of this chapter, you will be able to configure the JSON-to-Object and Object-to-XML transformers and DataMapper.

*Chapter 6*, *Configuring Filters*, explains what a filter is and how to configure the Logic filter. By the end of this chapter, you will be able to create a custom filter and configure the Message filter.

*Chapter 7*, *Handling Exceptions and Testing*, explains what an exception is. By the end of this chapter, you will be able to configure the Catch Exception Strategies, Rollback Exception Strategies, and JUnit testing.

*Chapter 8*, *Introducing Web Services*, explains what a web service is. By the end of this chapter, you will be able to create a JAX-WS web service and integrate external web services.

*Chapter 9*, *Understanding Flows, Routers, and Services*, explains what a Router is, and how to configure the Router and the Splitter Flow Control.

*Chapter 10*, *Configuring Cloud Connectors*, explains what a cloud connector is and how to integrate Twitter and Dropbox connectors.

# 2
# Working with Components and Patterns

In this chapter, we will cover:

- Configuring the component
- Using the Echo component to display the message payload
- Using a Flow Reference component to synchronously execute another flow
- Publishing a RESTful web service using the REST component
- Publishing a SOAP-based web service using the SOAP component

## Introduction

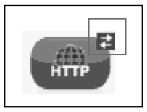
Mule has the ability of routing, filtering, transforming, and processing with components. Each of those abilities are assigned a good number of fine-grained processors. The configuration file of a Mule application that combines those elements can end up being large. The different types of configuration patterns provided by Mule are simple service pattern, bridge, validator, HTTP proxy, and WS proxy.

## Configuring the component

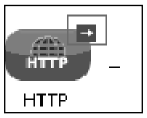
In this recipe, you will see how to configure a component in Mule Studio. We will have a look at how to use different components throughout this chapter.

### Getting ready

Mule uses HTTP Endpoints to send and receive requests over the HTTP protocol. Configured as either **Inbound** (also known as **message sources**) or **Outbound**, HTTP Endpoints use one of these two message exchange patterns: **request-response** or **one-way**.



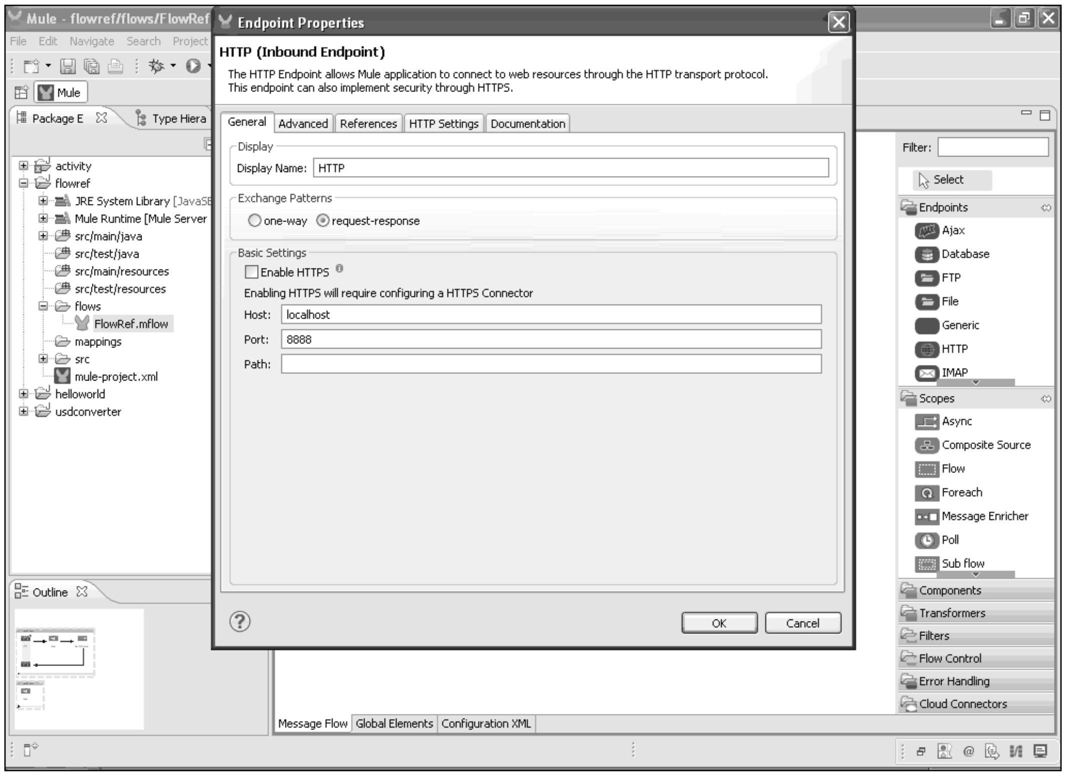
The arrows in the preceding screenshot indicate the request-response type of message exchange.



The arrow in this screenshot indicates the one-way type of message exchange.

## How to do it...

Double-click on the **HTTP** Endpoint to configure it. You will see a screen similar to the following screenshot on your window. You have to enter the **Host** and **Port** values. By default, the port number is **8081**; you can change the values of the **Host** and **Port** fields. However, these two fields are mandatory.


Endpoint Properties

HTTP (Inbound Endpoint)

The HTTP Endpoint allows Mule application to connect to web resources through the HTTP transport protocol.
This endpoint can also implement security through HTTPS.

Display Name: HTTP

Exchange Patterns: one-way [✓ request-response]

Basic Settings

Enable HTTPS

Enabling HTTPS will require configuring a HTTPS Connector

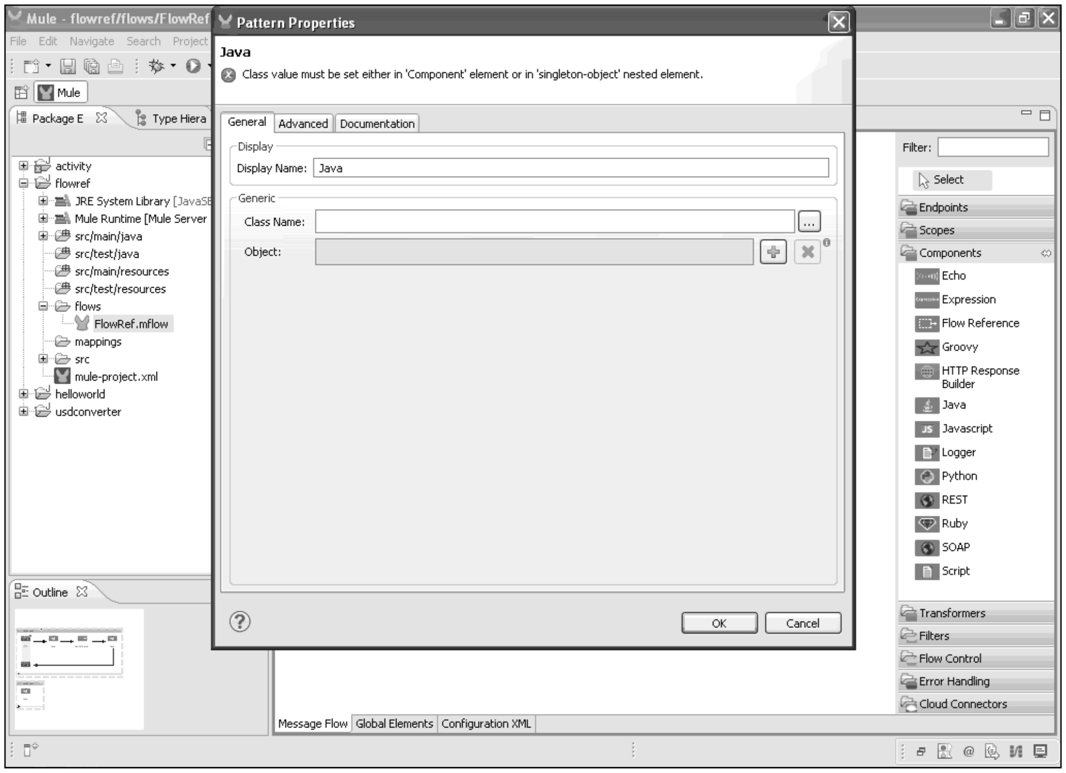
Host: localhost

Port: 8888

Path:

## The Java component

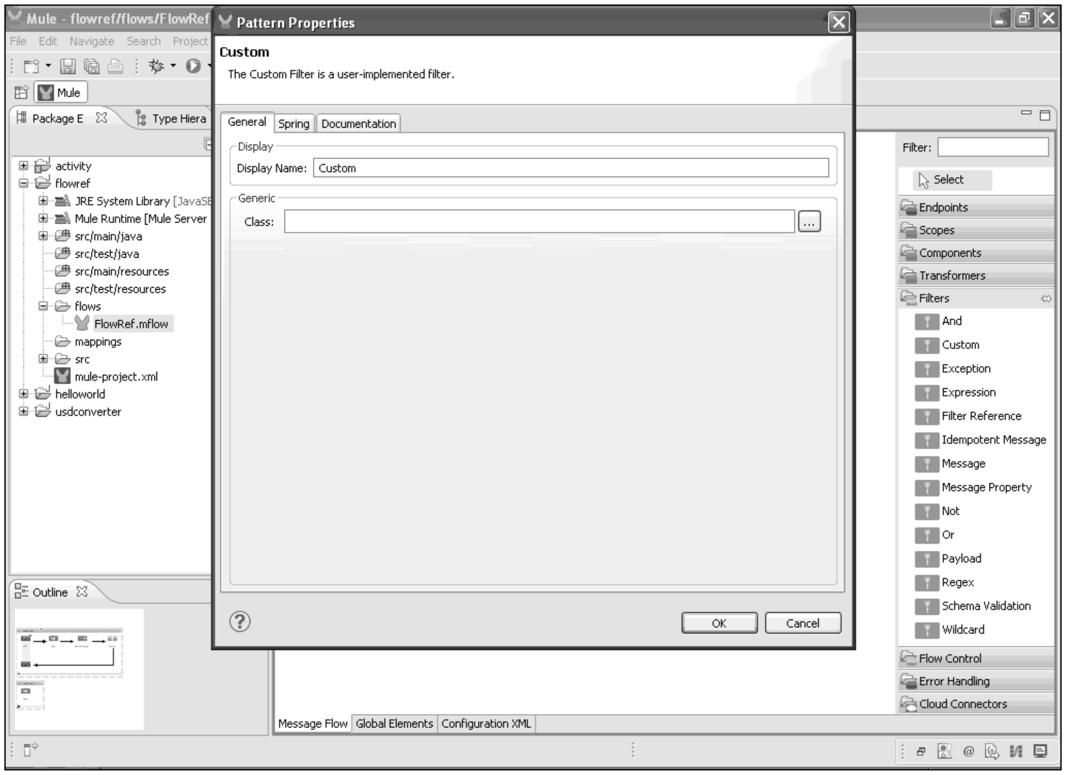
Double-click on the **Java** component to configure it. You can import the class you had created.



Class value must be set either in 'Component' element or in 'singleton-object' nested element.

## Custom filters

Filters specify conditions that must be met for a message to be routed to a service. If the condition is met, the message will go to another component. You can also create your own filter. To create a filter, implement the `Filter` interface, which has a single method. You can import a custom filter class using the extended `Filter` interface.


Pattern Properties

Custom

The Custom Filter is a user-implemented filter.

General | Spring | Documentation

Display

Display Name: Custom

Generic

Class:

OK Cancel

## How it works...

Components generally execute whenever a message is received; the logic embedded into components cannot be modified. Components such as **Logger**, **Echo**, **Java**, **Flow Ref**, and **Expression** all come under this category.

In the process of scripting a component, you have to develop your own business logic by writing a script, or you can import a script file written in scripting languages such as Ruby, Python, and Groovy. The Java component allows you to reference a Java class.

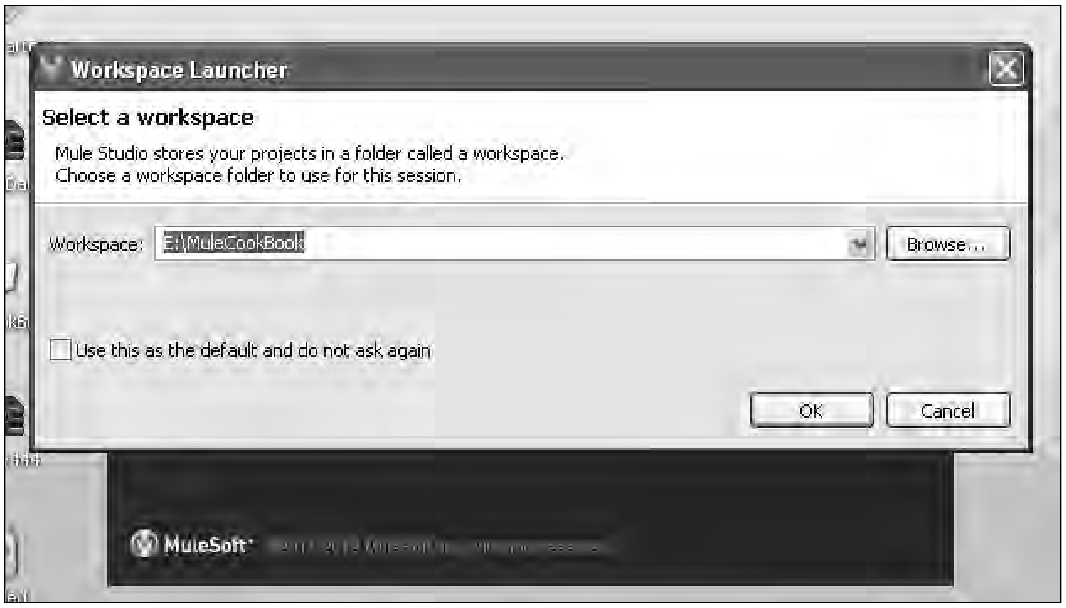
## Using the Echo component to display the message payload

The **Echo** component is used to display the message payload. The Echo component is used for displaying message payloads, which receives the end user HTTP request and returns the payload message to the HTTP response, which is then sent to the end user. In this recipe, you will see how to use and configure the Echo component in Mule Studio.

### Getting ready

In this example, we'll use the following components: HTTP, Logger, and Echo.

1. Open Mule Studio and enter the workspace name as shown in the following screenshot:



Workspace Launcher

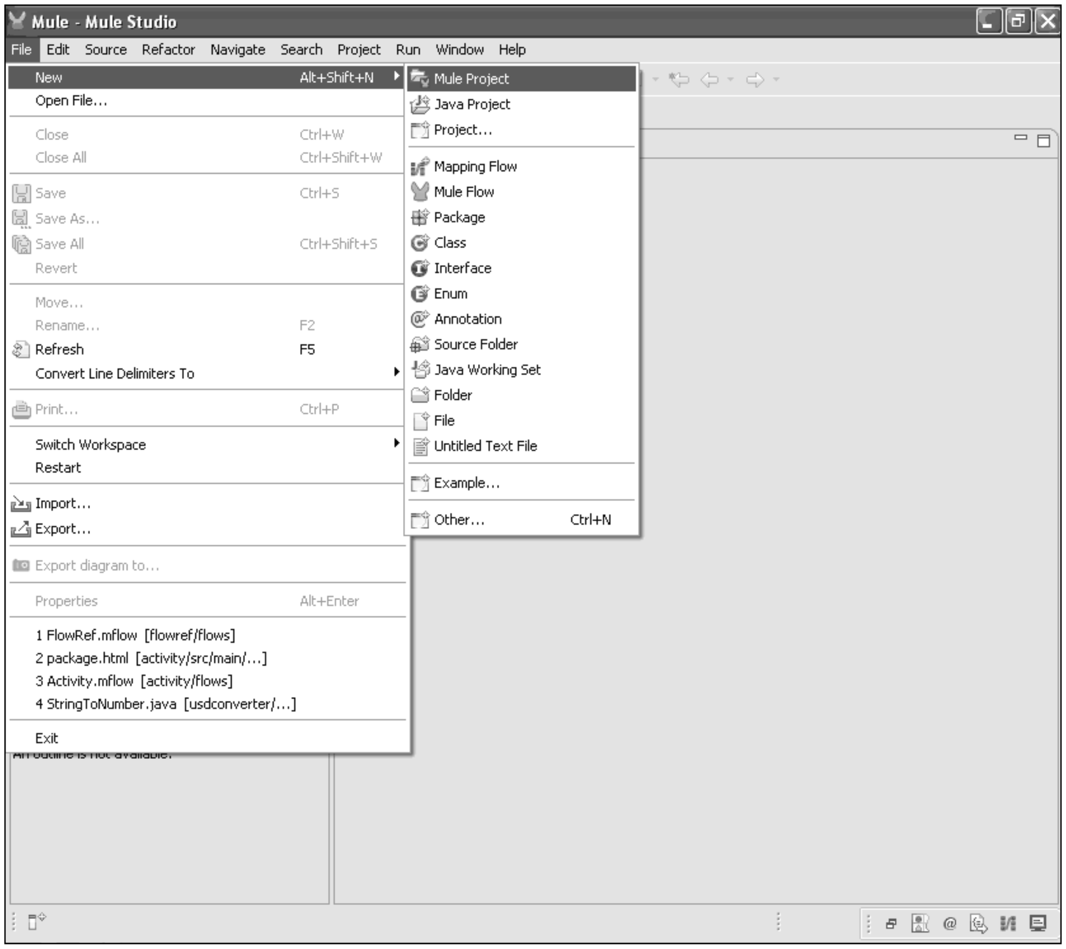
Select a workspace

Mule Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

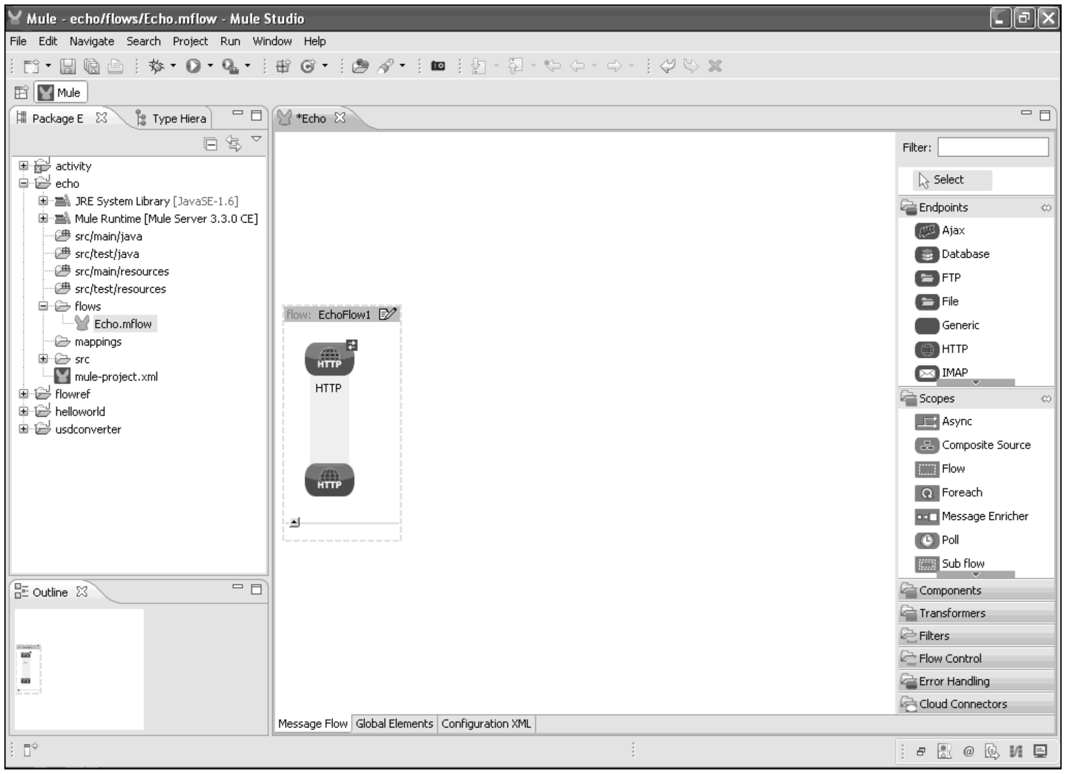
Workspace: E:\MuleCookBook

[ ] Use this as the default and do not ask again

2. To create a new project, go to **File** | **New** | **Mule Project**. Enter the project name, Echo, and click on **Next** and then on **Finish**. Your new project is created. You can now start the implementation.



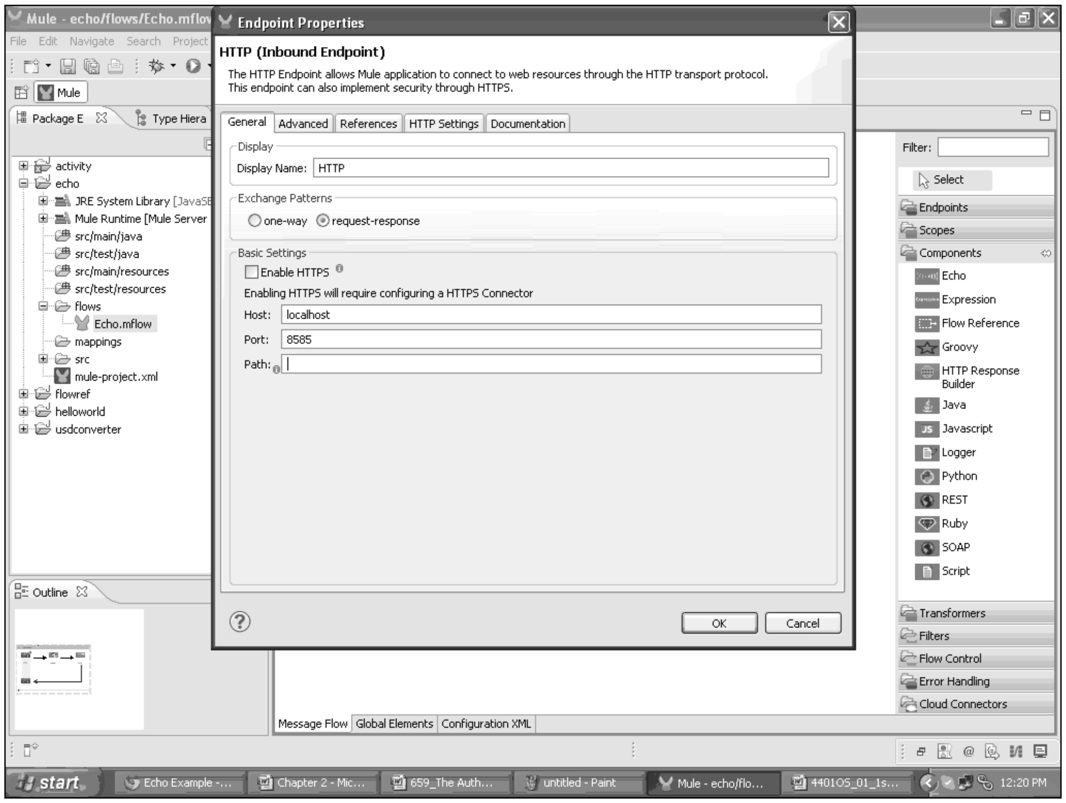
3. To create a flow, go to the `Echo.mflow` file, drag the **HTTP** Endpoint onto the canvas, and configure it by double-clicking on it.



## How to do it...

In this section, we will see how we can use the Logger and Echo components in a flow.

1. Double-click on the **HTTP** Endpoint to configure it. You can change the hostname and port number. Here, we have used port number 8585.



Endpoint Properties

HTTP (Inbound Endpoint)

The HTTP Endpoint allows Mule application to connect to web resources through the HTTP transport protocol.
This endpoint can also implement security through HTTPS.

Display Name: HTTP

Exchange Patterns: one-way [✓ request-response]

Basic Settings

Enable HTTPS

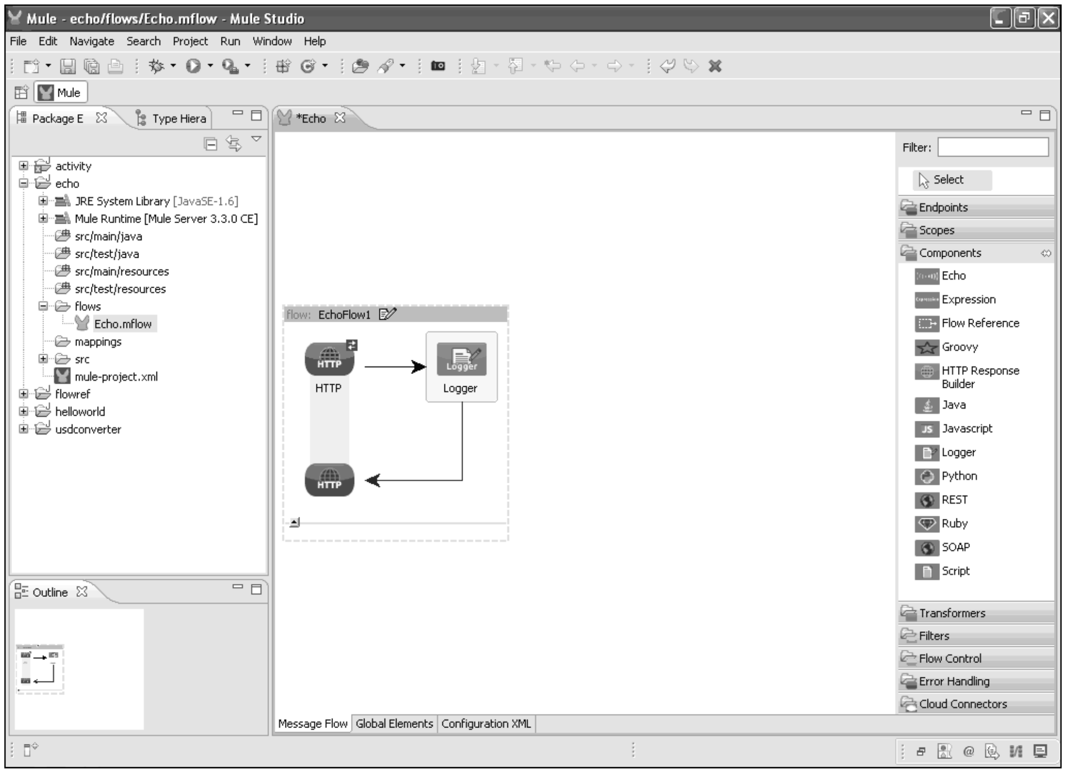
Enabling HTTPS will require configuring a HTTPS Connector

Host: localhost

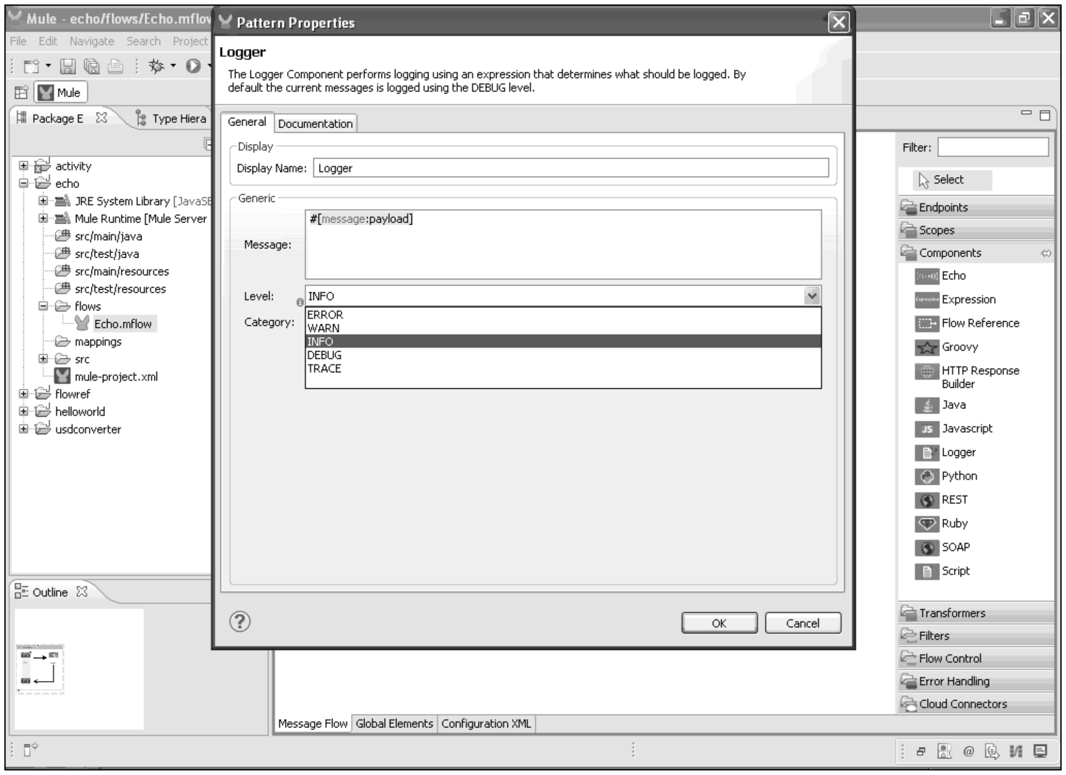
Port: 8585

Path:

2. To display messages on the console, drag the **Logger** component onto the canvas and configure it. The Logger component uses an expression to determine what information in the message should be displayed on the console. **Mule Expression Language** (**MEL**) is the primary language used for formulating such expressions throughout the Mule ESB.



3. Double-click on the **Logger** component to configure it. You can see different types of levels. The Logger component level is used for displaying error messages or exceptions. We have selected the **INFO** level here. In the message box, you should use the expression `#[message:payload]`. This expression is used for displaying messages on the console.



Logger

The Logger Component performs logging using an expression that determines what should be logged. By default the current messages is logged using the DEBUG level.

Display Name: Logger

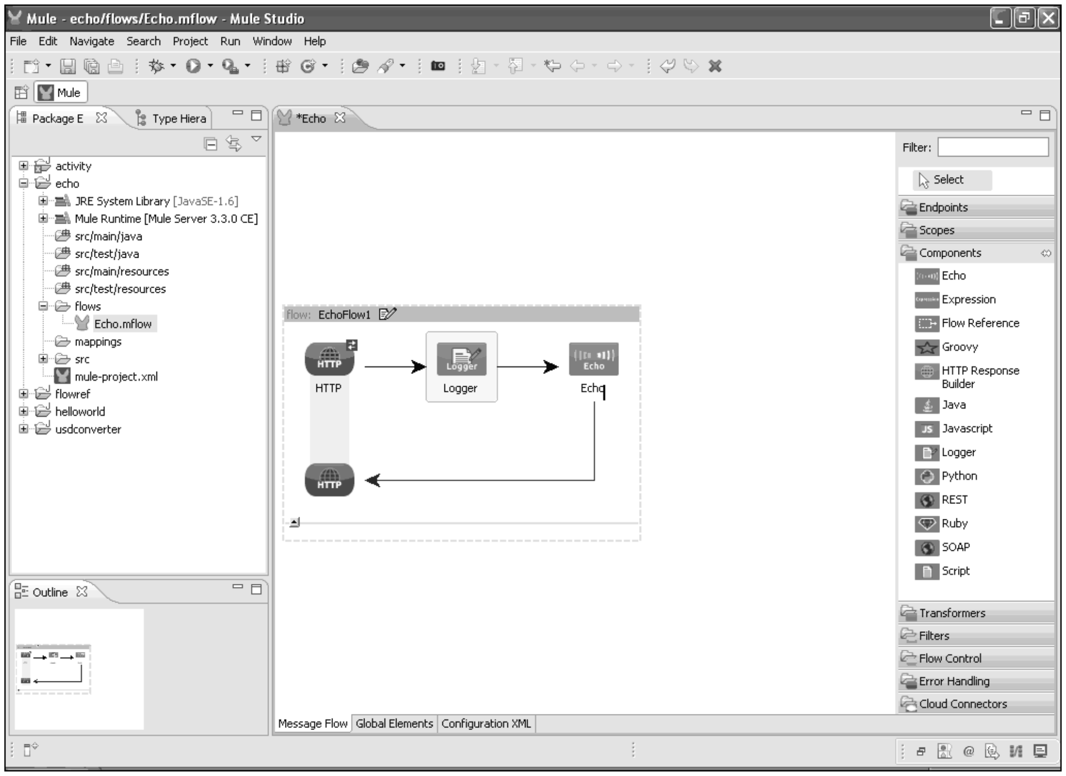
Message: #[message:payload]

Level: INFO

Category:

ERROR
WARN
INFO
DEBUG
TRACE

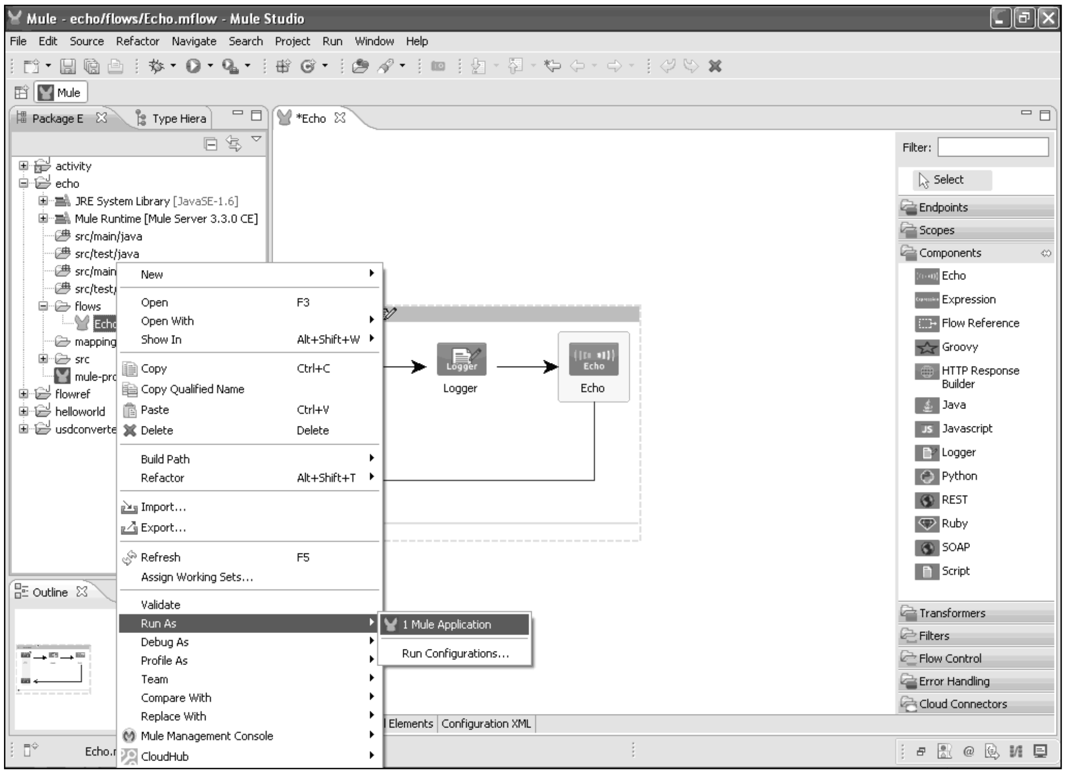
4. Drag the **Echo** component onto the canvas; there is no need to configure it. Messages sent to an **Echo** component simply return the message payload as the response to an end user.



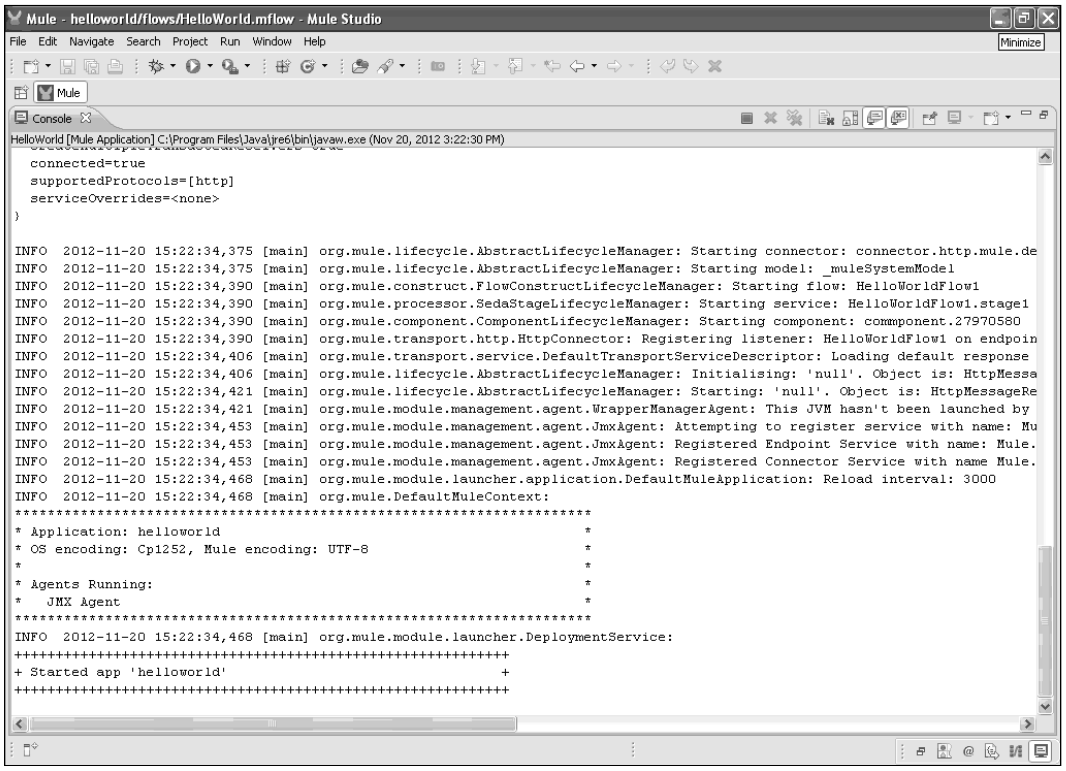
## How it works...

In this section, you will see how to deploy the application and how to run the application on the browser.

1. Now we are ready for the deployment. If you haven't saved your application code, do save it. After saving your project, right-click on the `Echo.mflow` file and go to **Run As | Mule Application**.

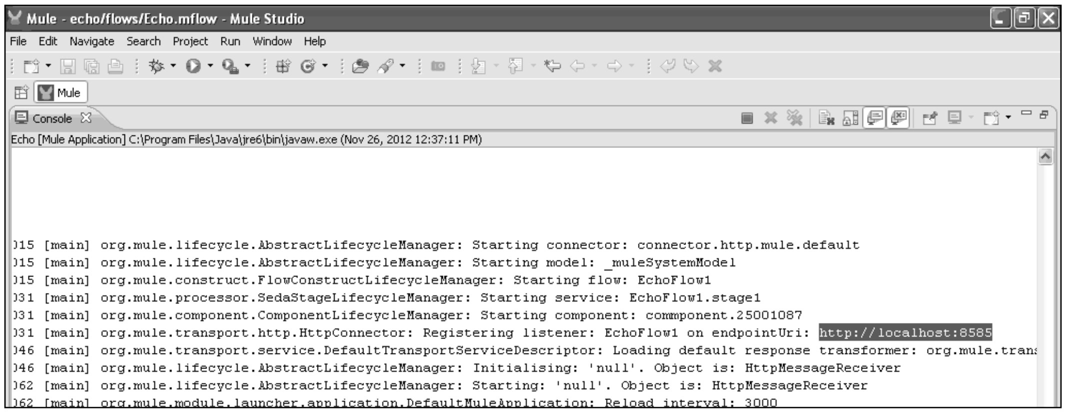


2. If your application code is successfully deployed, you will see the following message on the console: `Started app 'helloworld'`.


Mule - helloworld/flows/HelloWorld.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Minimize
Mule
Console
HelloWorld [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Nov 20, 2012 3:22:30 PM)

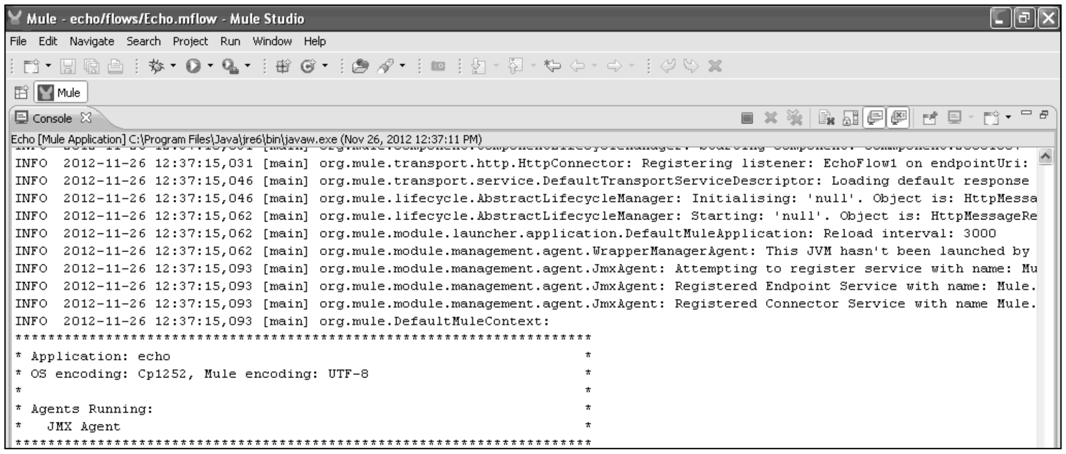
```
  connected=true
  supportedProtocols=[http]
  serviceOverrides=<none>
}

INFO  2012-11-20 15:22:34,375 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting connector: connector.http.mule.de
INFO  2012-11-20 15:22:34,375 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting model: _muleSystemModel
INFO  2012-11-20 15:22:34,390 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: HelloWorldFlow1
INFO  2012-11-20 15:22:34,390 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: HelloWorldFlow1.stage1
INFO  2012-11-20 15:22:34,390 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.27970580
INFO  2012-11-20 15:22:34,390 [main] org.mule.transport.http.HttpConnector: Registering listener: HelloWorldFlow1 on endpoin
INFO  2012-11-20 15:22:34,406 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default response
INFO  2012-11-20 15:22:34,406 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMessa
INFO  2012-11-20 15:22:34,421 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessageRe
INFO  2012-11-20 15:22:34,421 [main] org.mule.module.management.agent.WrapperManagerAgent: This JVM hasn't been launched by
INFO  2012-11-20 15:22:34,453 [main] org.mule.module.management.agent.JmxAgent: Attempting to register service with name: Mu
INFO  2012-11-20 15:22:34,453 [main] org.mule.module.management.agent.JmxAgent: Registered Endpoint Service with name: Mule.
INFO  2012-11-20 15:22:34,453 [main] org.mule.module.management.agent.JmxAgent: Registered Connector Service with name Mule.
INFO  2012-11-20 15:22:34,468 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
INFO  2012-11-20 15:22:34,468 [main] org.mule.DefaultMuleContext:
**********************************************************************
* Application: helloworld                                            *
* OS encoding: Cp1252, Mule encoding: UTF-8                          *
*                                                                    *
* Agents Running:                                                    *
*   JMX Agent                                                        *
**********************************************************************
INFO  2012-11-20 15:22:34,468 [main] org.mule.module.launcher.DeploymentService:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Started app 'helloworld'                                  +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3. Copy the URL `http://localhost:8585` and paste it in your browser.


Mule - echo/flows/Echo.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console
Echo [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Nov 26, 2012 12:37:11 PM)

```
)15 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting connector: connector.http.mule.default
)15 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting model: _muleSystemModel
)15 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: EchoFlow1
)31 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: EchoFlow1.stage1
)31 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.25001087
)31 [main] org.mule.transport.http.HttpConnector: Registering listener: EchoFlow1 on endpointUri: http://localhost:8585
)46 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default response transformer: org.mule.tran
)46 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMessageReceiver
)62 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessageReceiver
)62 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
```

4. To see the output on the console, paste the URL in your browser and type in `/EchoExample`. When a user types `http://localhost:8585/EchoExample` in the browser, Mule returns a message in the browser that reads `/EchoExample`, as shown in the following screenshot:


Mule - echo/flows/Echo.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console

```
Echo [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Nov 26, 2012 12:37:11 PM)
INFO  2012-11-26 12:37:15,031 [main] org.mule.transport.http.HttpConnector: Registering listener: EchoFlow1 on endpointUri:
INFO  2012-11-26 12:37:15,046 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default response
INFO  2012-11-26 12:37:15,046 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMessa
INFO  2012-11-26 12:37:15,062 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessageRe
INFO  2012-11-26 12:37:15,062 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
INFO  2012-11-26 12:37:15,062 [main] org.mule.module.management.agent.WrapperManagerAgent: This JVM hasn't been launched by
INFO  2012-11-26 12:37:15,093 [main] org.mule.module.management.agent.JmxAgent: Attempting to register service with name: Mu
INFO  2012-11-26 12:37:15,093 [main] org.mule.module.management.agent.JmxAgent: Registered Endpoint Service with name: Mule.
INFO  2012-11-26 12:37:15,093 [main] org.mule.module.management.agent.JmxAgent: Registered Connector Service with name Mule.
INFO  2012-11-26 12:37:15,093 [main] org.mule.DefaultMuleContext:
**********************************************************************
* Application: echo                                                  *
* OS encoding: Cp1252, Mule encoding: UTF-8                          *
*                                                                    *
* Agents Running:                                                    *
*   JMX Agent                                                        *
**********************************************************************
```

## Using the command prompt

To run a Mule application, enter the following command on the command prompt:

```
mule [-config <your-config.xml>]
```

Here, <`your-config.xml`> is the Mule configuration file you want to use. If you don't specify the configuration file, Mule looks for `mule-config.xml`, which is a generic name that does not exist in the default configuration file. If you have only one configuration file, you can name it `mule-config.xml` so that you can run Mule with it just by typing in mule. To stop Mule, press *Ctrl* + *C*.

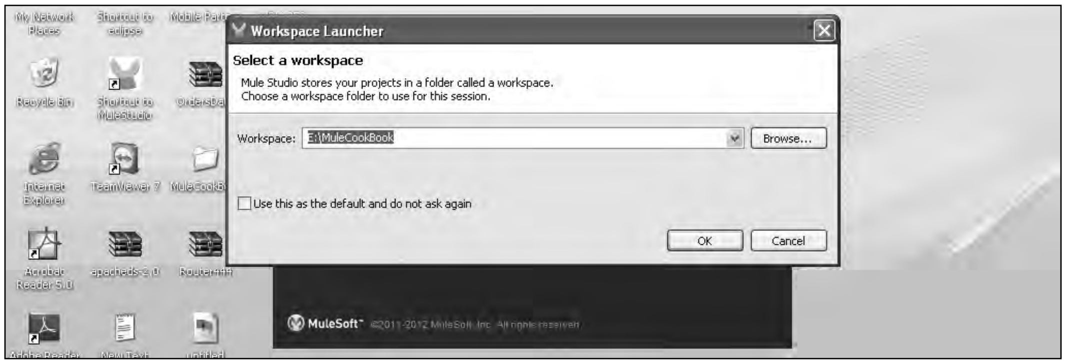
## Using a Flow Reference component to synchronously execute another flow

**Flow Reference** is used to synchronously execute another flow that is external to the current flow. If a message reaches the Flow Reference component, Mule invokes the external flow referenced by it. Once the referenced flow completes, the control passes back to the initiating flow only after the external process is completed.

## Getting ready

To demonstrate this example, we'll use the following four components: HTTP, Logger, Java, and Flow Ref.

1. Open Mule Studio and enter the workspace name as shown in the following screenshot:



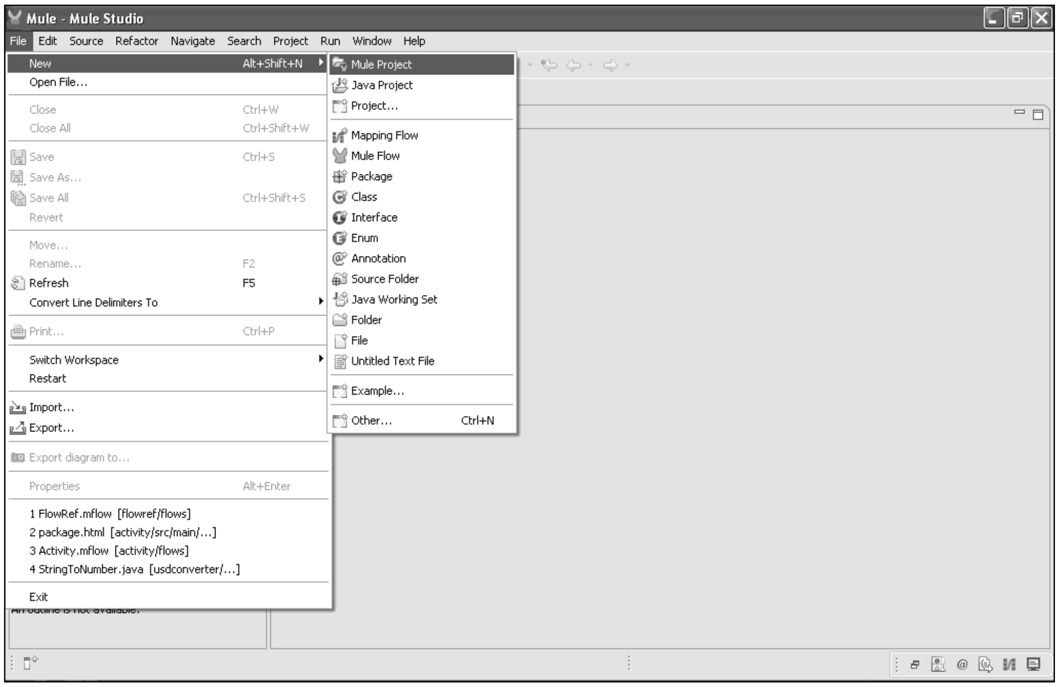
Select a workspace

Mule Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: E:\MuleCookBook

Use this as the default and do not ask again

2. To create a new project, go to **File** | **New** | **Mule Project**. Enter the project name, `FlowRef`, and click on **Next** and then on **Finish**. Your new project has been created now; so we are ready to start the implementation.



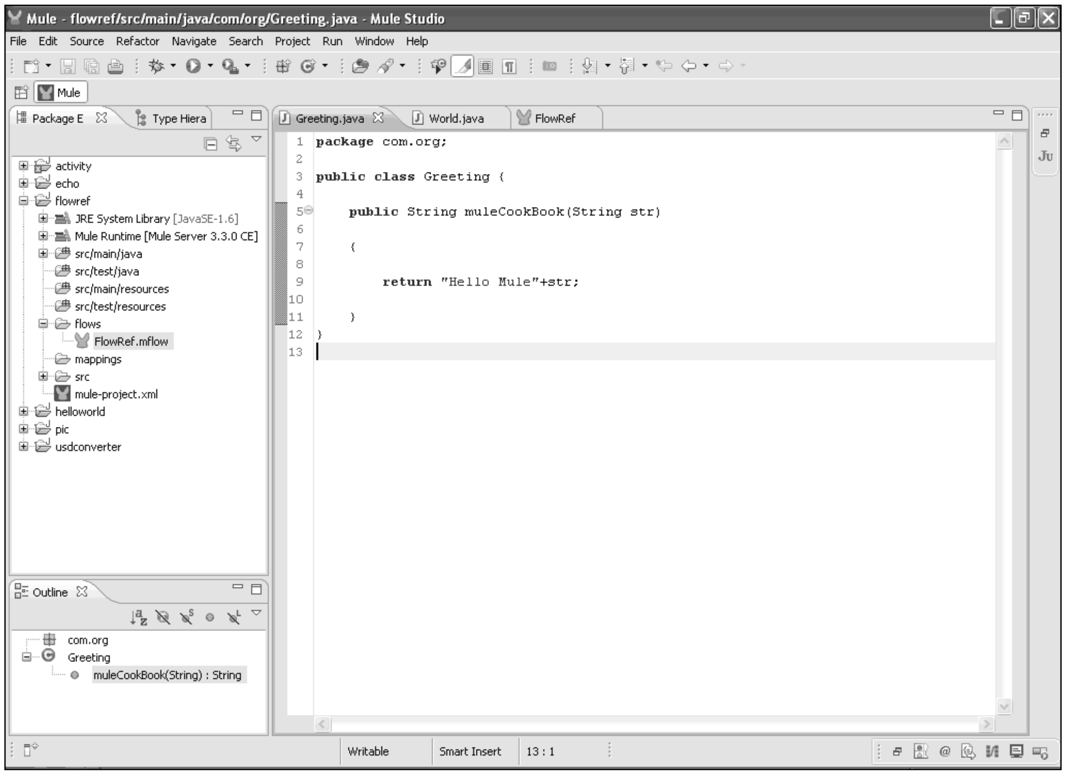
## How to do it...

In this section, you will see how to configure the Java component and the Flow Ref component. Here you are creating a class, and the output will be displayed on the browser through this class.

1. To create a class, go to `src/main/java`, right-click on it, and go to **New** | **Class**. Create a class named `Greeting` under the package `com.org`; here, we create the `muleCookBook` method and its return type is set to `String`:

   ```
   public String muleCookBook(String str)

     {
       return "HelloMule"+str;
     }
   ```

   You can see the creation of this method in the following screenshot:

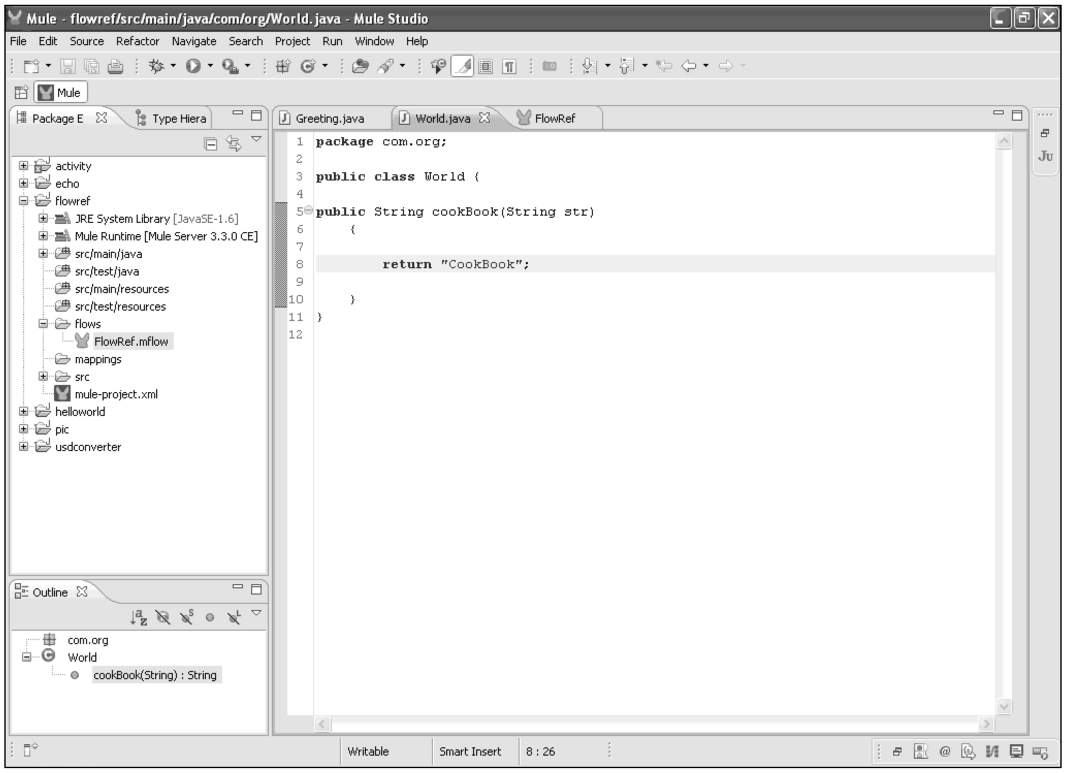


```
package com.org;

public class Greeting {

    public String muleCookBook(String str)

    {

        return "Hello Mule"+str;

    }
}
```

2. To create a class, right-click on the package. Create one more class, `World`, under the same package. Here, we create the method `cookbook` and its return type is set to `String`:

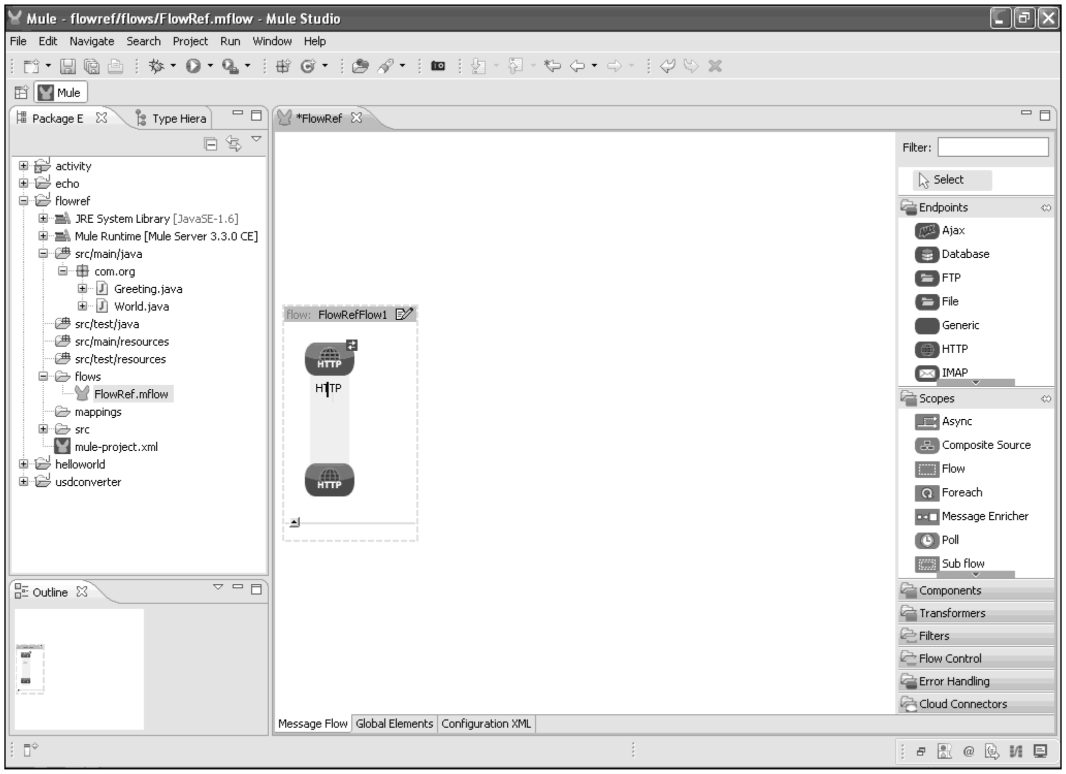
```
public String cookBook(String str)
  {
    return "CookBook";
  }
```

You can see the creation of this method in the following screenshot:

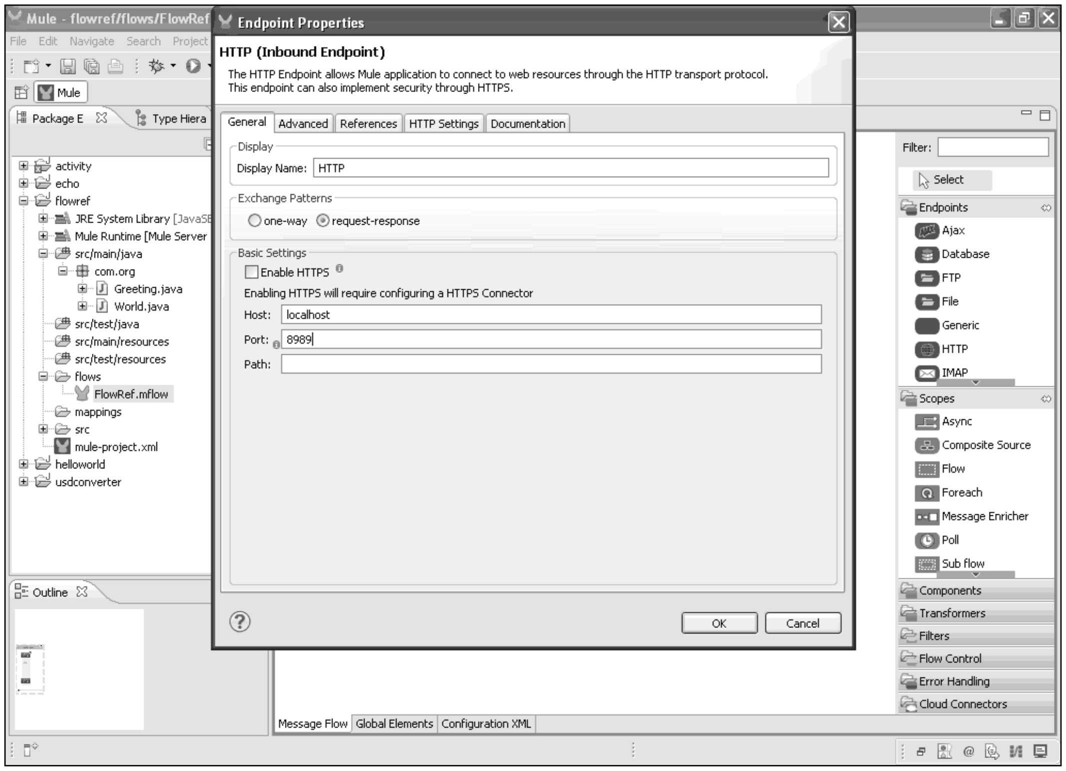


```
package com.org;

public class World {

public String cookBook(String str)
    {

        return "CookBook";

    }
}
```

3. To create a flow, go to the `FlowRef.Mflow` file. Drag the **HTTP** Endpoint onto the canvas and configure it.



4. Double-click on the **HTTP** Endpoint to configure it. You can change the hostname and port number. We have used the port number 8989 here. Click on the **OK** button. By default, the **request-response** method is selected, as shown in the following screenshot:



Endpoint Properties

HTTP (Inbound Endpoint)

The HTTP Endpoint allows Mule application to connect to web resources through the HTTP transport protocol.
This endpoint can also implement security through HTTPS.

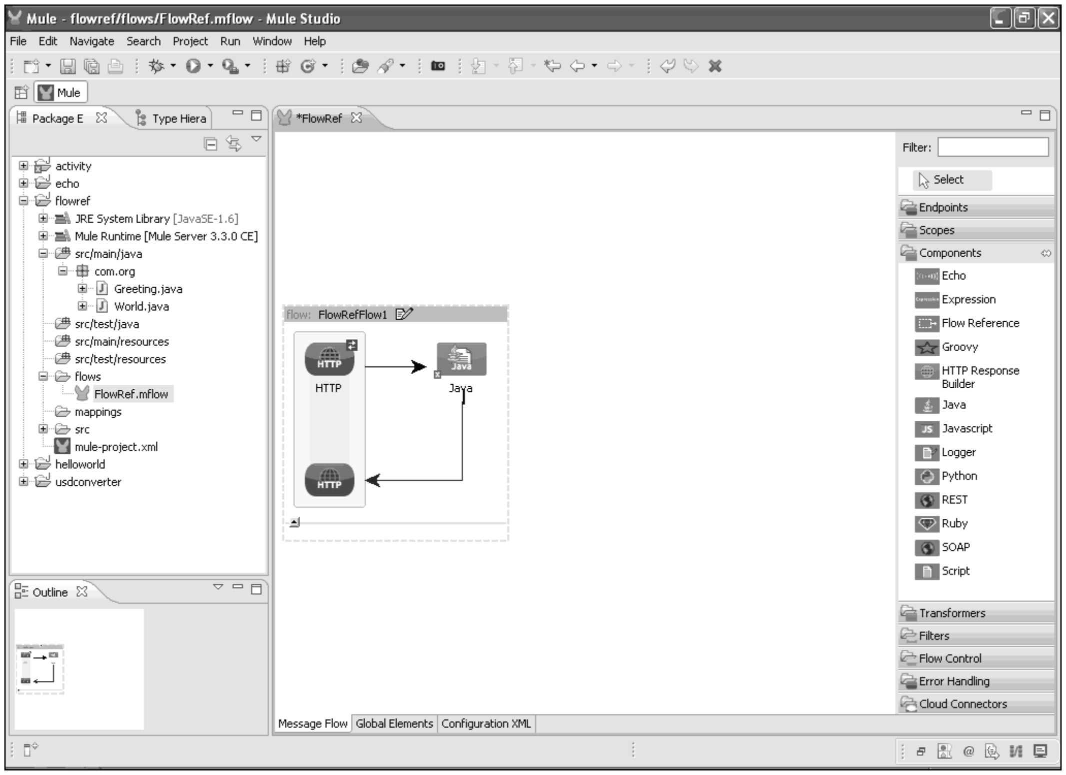
General | Advanced | References | HTTP Settings | Documentation

Display
Display Name: HTTP

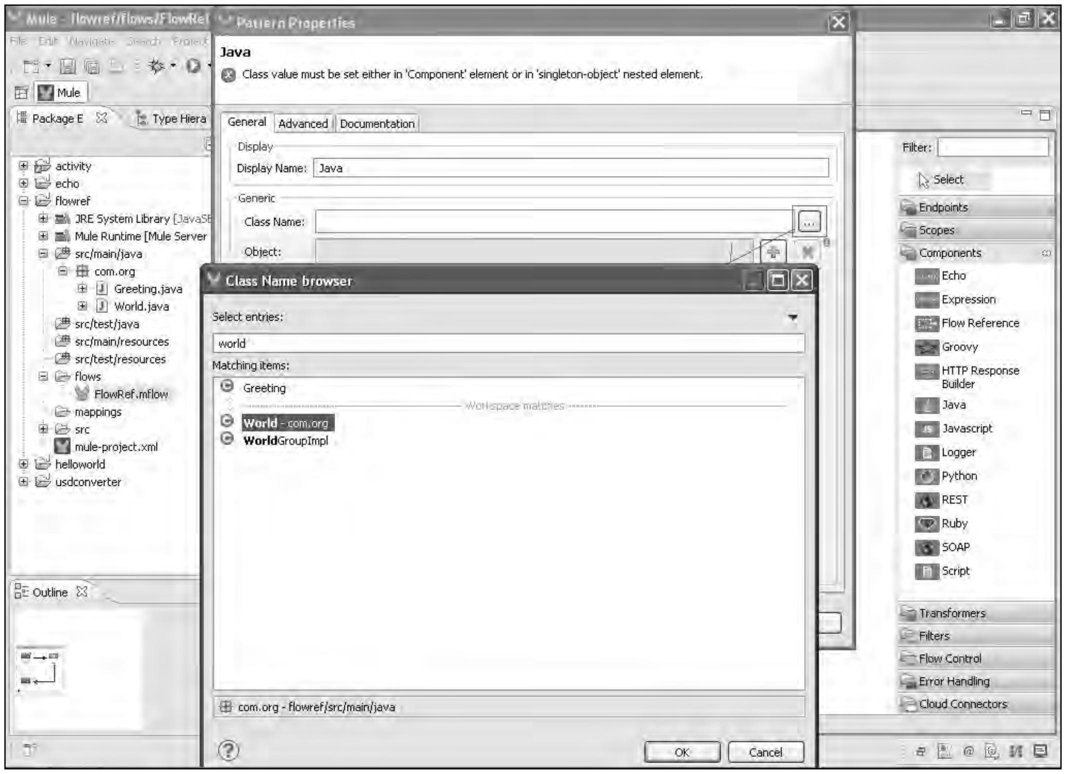
Exchange Patterns
one-way [✓ request-response]

Basic Settings
Enable HTTPS
Enabling HTTPS will require configuring a HTTPS Connector
Host: localhost
Port: 8989
Path:

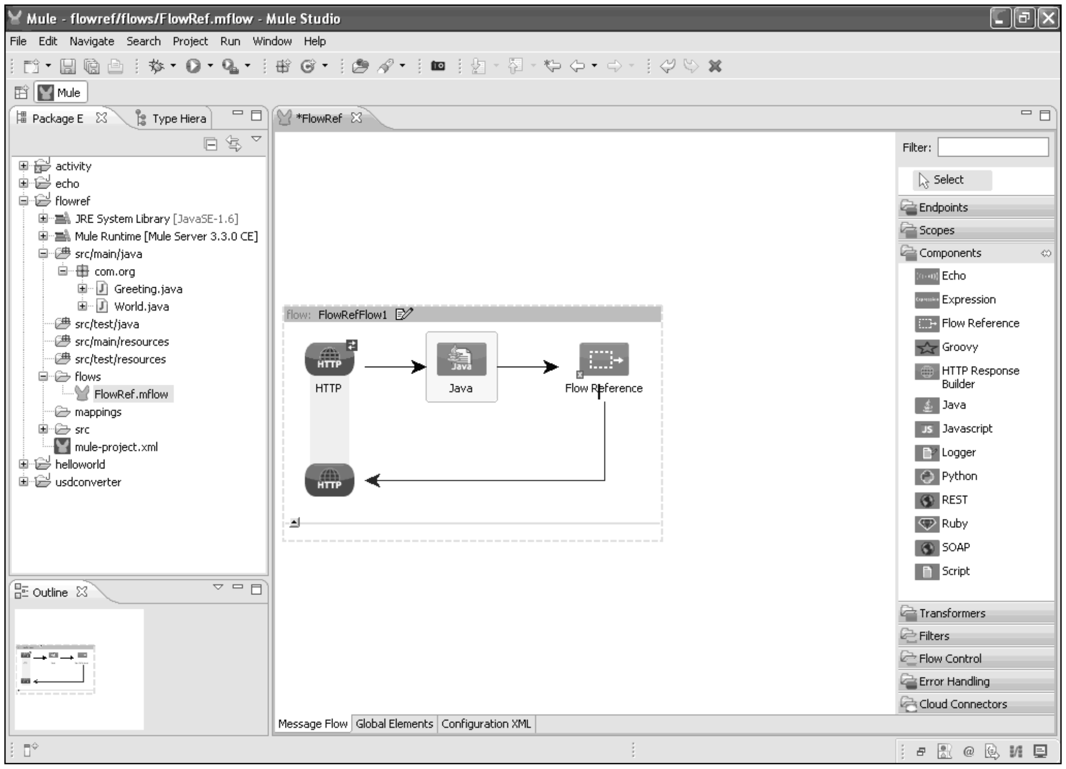
5. To import a class, drag the **Java** component and configure it.



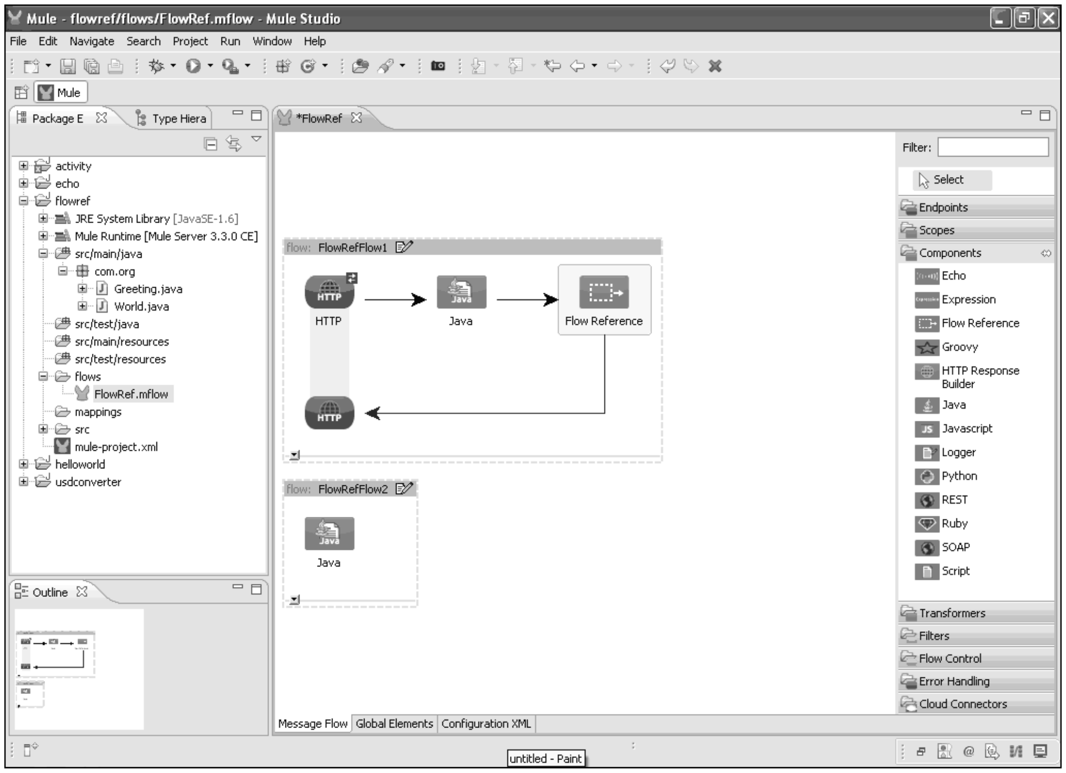
6. Double-click on the **Java** component to configure it. Just click on the **Browse** button and a new window, **Class name browser**, will open. Here you can import the `World.java` class, which was created before, and click on the **OK** button.



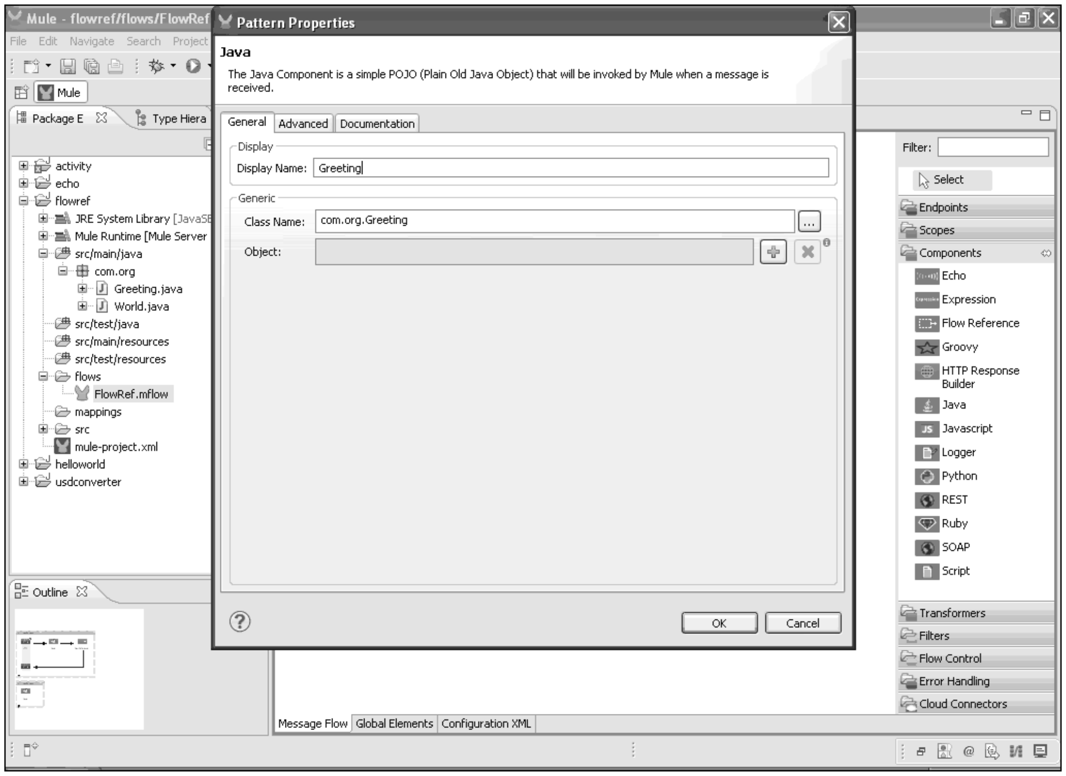
7. To reference another flow name, drag the **Flow Reference** component onto the canvas.



8. Before configuring the **Flow Reference** component, you have to drag the **Java** component onto another flow (you can see this in the following screenshot). Configure that **Java** component. If you create another flow, just drag the component onto the canvas outside the first flow; this will create another flow.



9. Double-click on the **Java** component to configure it; change the display name so you can identify the class name. Import the `Greeting.java` class that was created before and click on the **OK** button.



Java

The Java Component is a simple POJO (Plain Old Java Object) that will be invoked by Mule when a message is received.

Display

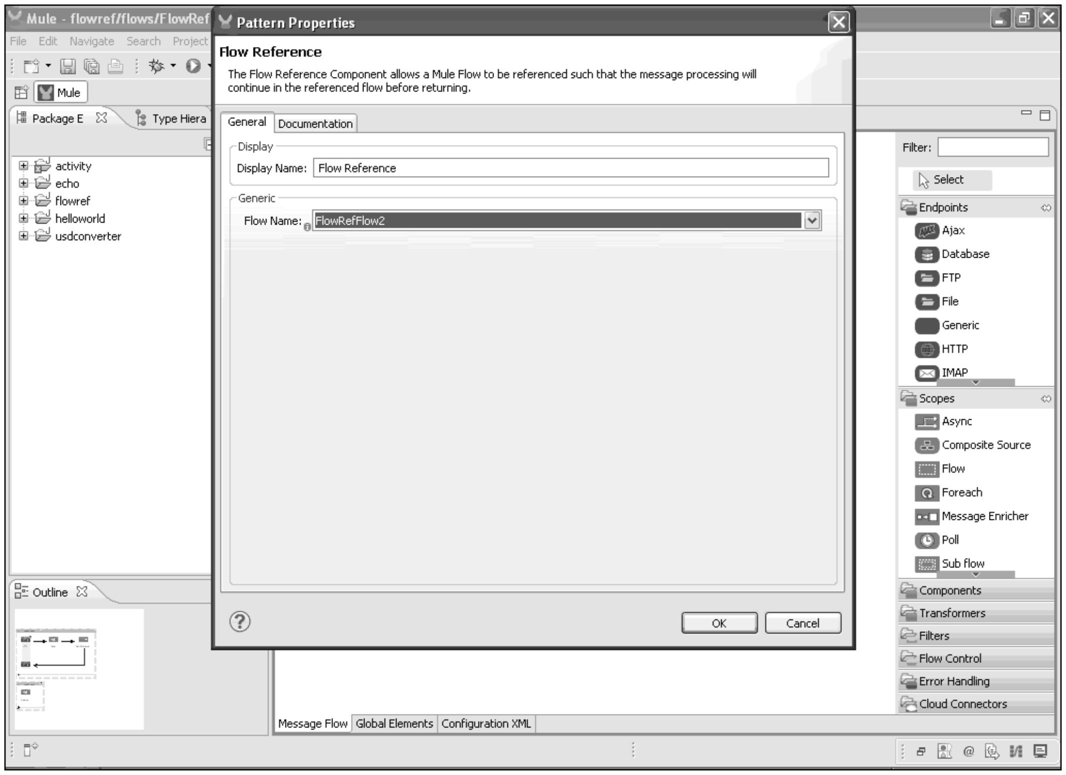
Display Name: Greeting

Generic

Class Name: com.org.Greeting

Object:

10. Now double-click on the **Flow Ref** component to configure it. Assign another Flow Reference name. Now we are ready to deploy our application.



Pattern Properties

Flow Reference

The Flow Reference Component allows a Mule Flow to be referenced such that the message processing will continue in the referenced flow before returning.

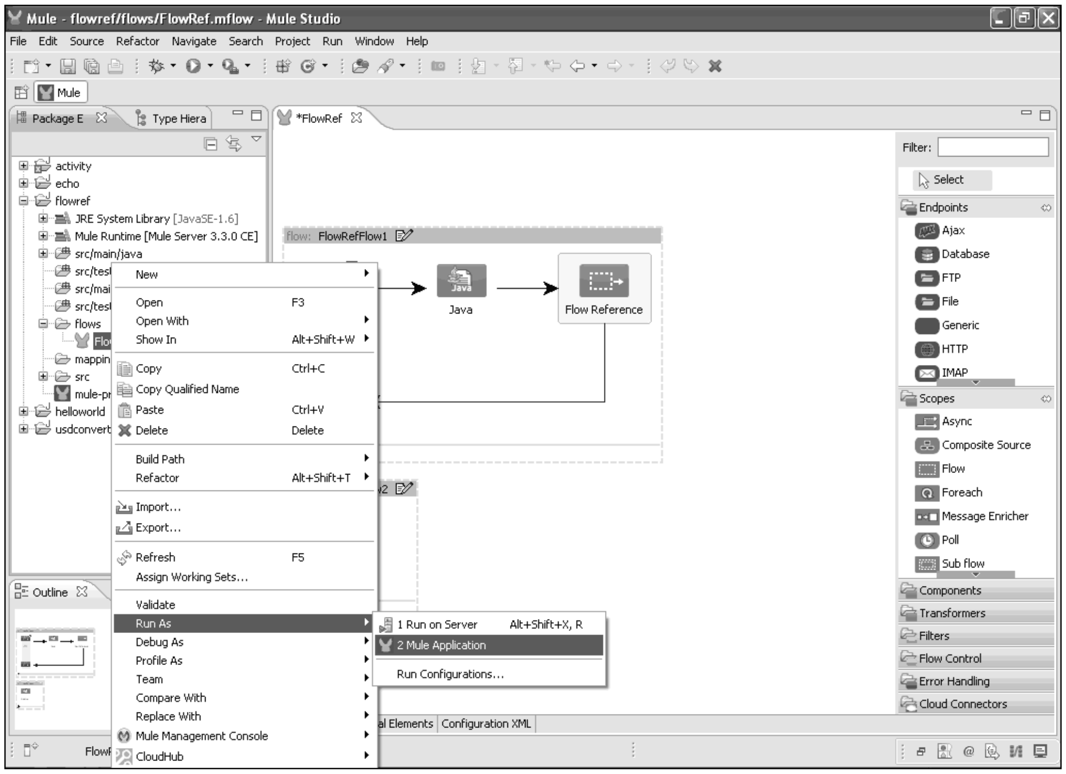
Display Name: Flow Reference

Flow Name: FlowRefFlow2

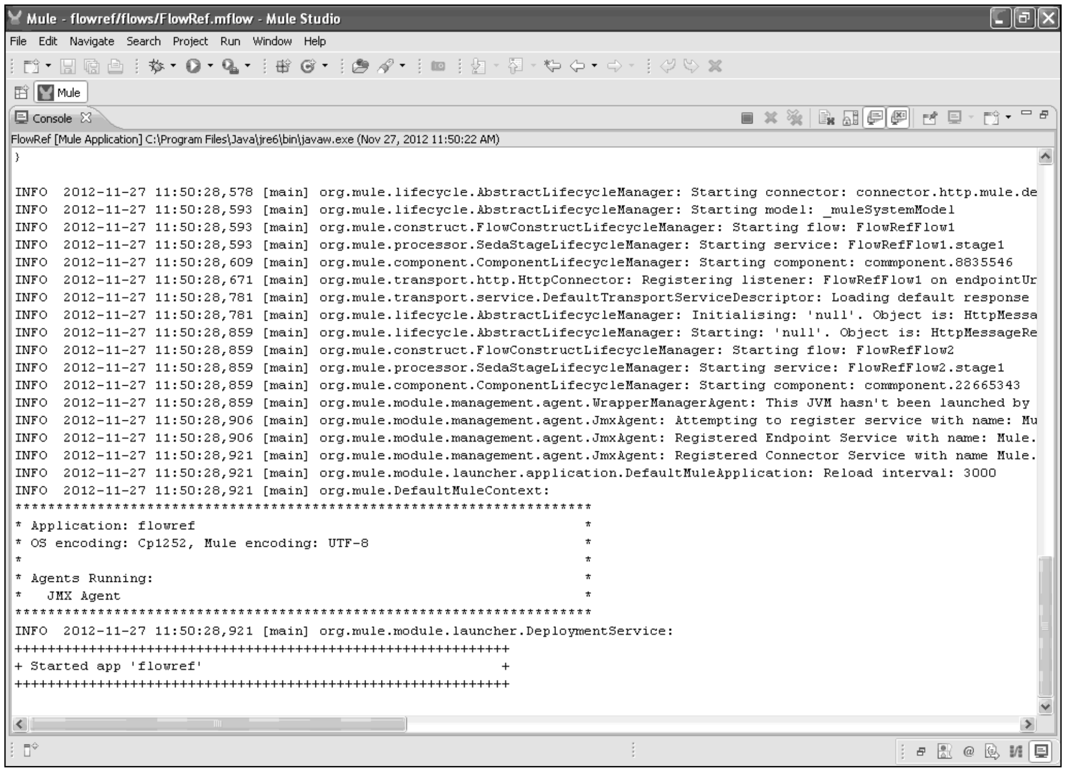
## How it works...

In this section, you will see how to deploy the application and how to run the application in the browser.

1. Now we are ready for the deployment. If you haven't saved your application code, do save it. After saving your project, right-click on the `Echo.mflow` file and go to **Run As** | **Mule Application**.

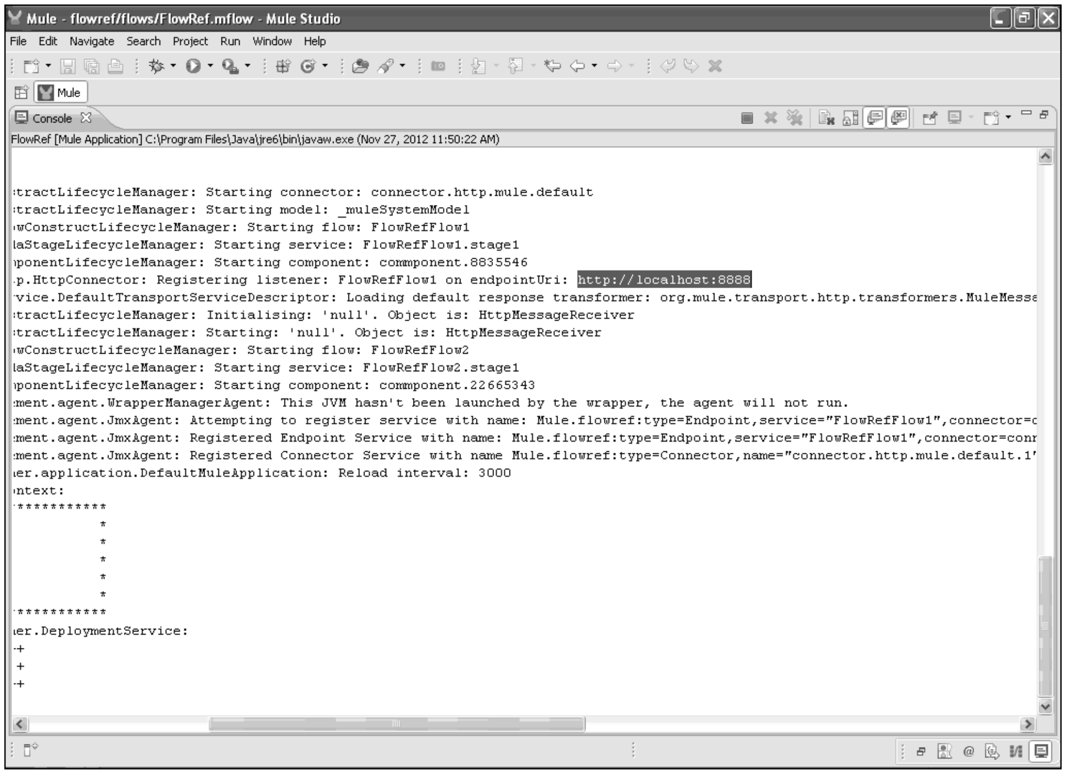


2. If your application code is successfully deployed, you will see the message `Started app 'helloworld'` on the console.


Mule - flowref/flows/FlowRef.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console

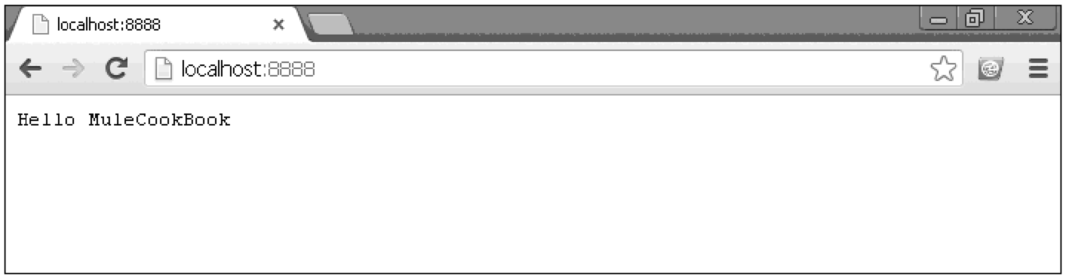
```
FlowRef [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Nov 27, 2012 11:50:22 AM)
}

INFO  2012-11-27 11:50:28,578 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting connector: connector.http.mule.de
INFO  2012-11-27 11:50:28,593 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting model: _muleSystemModel
INFO  2012-11-27 11:50:28,593 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: FlowRefFlow1
INFO  2012-11-27 11:50:28,593 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: FlowRefFlow1.stage1
INFO  2012-11-27 11:50:28,609 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.8835546
INFO  2012-11-27 11:50:28,671 [main] org.mule.transport.http.HttpConnector: Registering listener: FlowRefFlow1 on endpointUr
INFO  2012-11-27 11:50:28,781 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default response
INFO  2012-11-27 11:50:28,781 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMessa
INFO  2012-11-27 11:50:28,859 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessageRe
INFO  2012-11-27 11:50:28,859 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: FlowRefFlow2
INFO  2012-11-27 11:50:28,859 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: FlowRefFlow2.stage1
INFO  2012-11-27 11:50:28,859 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.22665343
INFO  2012-11-27 11:50:28,859 [main] org.mule.module.management.agent.WrapperManagerAgent: This JVM hasn't been launched by
INFO  2012-11-27 11:50:28,906 [main] org.mule.module.management.agent.JmxAgent: Attempting to register service with name: Mu
INFO  2012-11-27 11:50:28,906 [main] org.mule.module.management.agent.JmxAgent: Registered Endpoint Service with name: Mule.
INFO  2012-11-27 11:50:28,921 [main] org.mule.module.management.agent.JmxAgent: Registered Connector Service with name Mule.
INFO  2012-11-27 11:50:28,921 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
INFO  2012-11-27 11:50:28,921 [main] org.mule.DefaultMuleContext:
**********************************************************************
* Application: flowref                                               *
* OS encoding: Cp1252, Mule encoding: UTF-8                          *
*                                                                    *
* Agents Running:                                                    *
*   JMX Agent                                                        *
**********************************************************************
INFO  2012-11-27 11:50:28,921 [main] org.mule.module.launcher.DeploymentService:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Started app 'flowref'                                     +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3. Copy the URL `http://localhost:8989` and paste it in your browser.


Mule - flowref/flows/FlowRef.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console

```
FlowRef [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Nov 27, 2012 11:50:22 AM)

:tractLifecycleManager: Starting connector: connector.http.mule.default
:tractLifecycleManager: Starting model: _muleSystemModel
wConstructLifecycleManager: Starting flow: FlowRefFlow1
laStageLifecycleManager: Starting service: FlowRefFlow1.stage1
ponentLifecycleManager: Starting component: commponent.8835546
.p.HttpConnector: Registering listener: FlowRefFlow1 on endpointUri: http://localhost:8888
vice.DefaultTransportServiceDescriptor: Loading default response transformer: org.mule.transport.http.transformers.MuleMessa
:tractLifecycleManager: Initialising: 'null'. Object is: HttpMessageReceiver
:tractLifecycleManager: Starting: 'null'. Object is: HttpMessageReceiver
wConstructLifecycleManager: Starting flow: FlowRefFlow2
laStageLifecycleManager: Starting service: FlowRefFlow2.stage1
ponentLifecycleManager: Starting component: commponent.22665343
:ment.agent.WrapperManagerAgent: This JVM hasn't been launched by the wrapper, the agent will not run.
:ment.agent.JmxAgent: Attempting to register service with name: Mule.flowref:type=Endpoint,service="FlowRefFlow1",connector=c
:ment.agent.JmxAgent: Registered Endpoint Service with name: Mule.flowref:type=Endpoint,service="FlowRefFlow1",connector=conr
:ment.agent.JmxAgent: Registered Connector Service with name Mule.flowref:type=Connector,name="connector.http.mule.default.1"
ler.application.DefaultMuleApplication: Reload interval: 3000
ntext:
***********
          *
          *
          *
          *
          *
***********
ler.DeploymentService:
-+
 +
-+
```

4. To see the output, paste the URL onto your browser. You will see the output as shown in the following screenshot. Here, `Hello Mule` is called from the `Greeting` class through the **Flow Reference** component, and `CookBook` is called from the `World` class:



```
Hello MuleCookBook
```

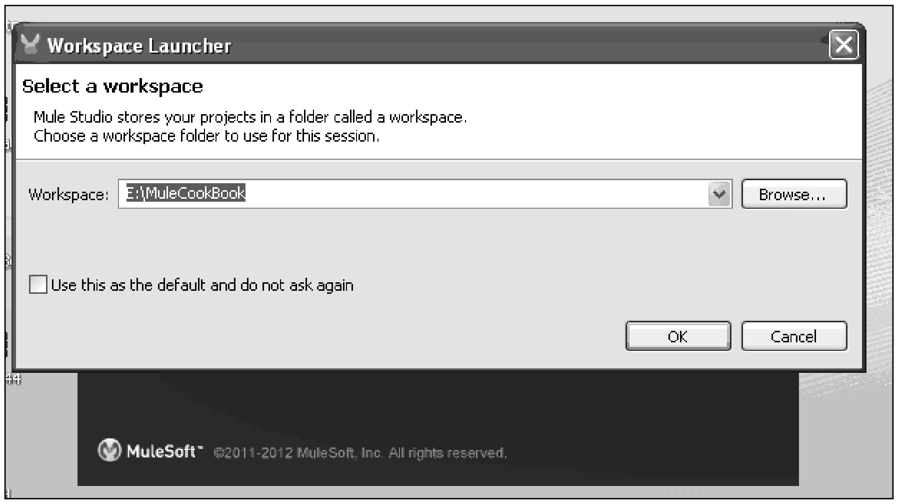
## Publishing a RESTful web service using the REST component

**REST** stands for **Representational State Transfer**. REST exposes a much simpler interface than SOAP. REST components are bound with HTTP. So, if you are designing an application to be used exclusively on the Web, REST is a very good option. RESTful applications simply rely on the built-in HTTP security. A REST design is good for database-driven applications and also when a client wants quick integration.

### Getting ready

In this example, we'll use three components: HTTP, Logger, and REST.

1. Open Mule Studio and enter the workspace name as shown in following screenshot:



Workspace Launcher

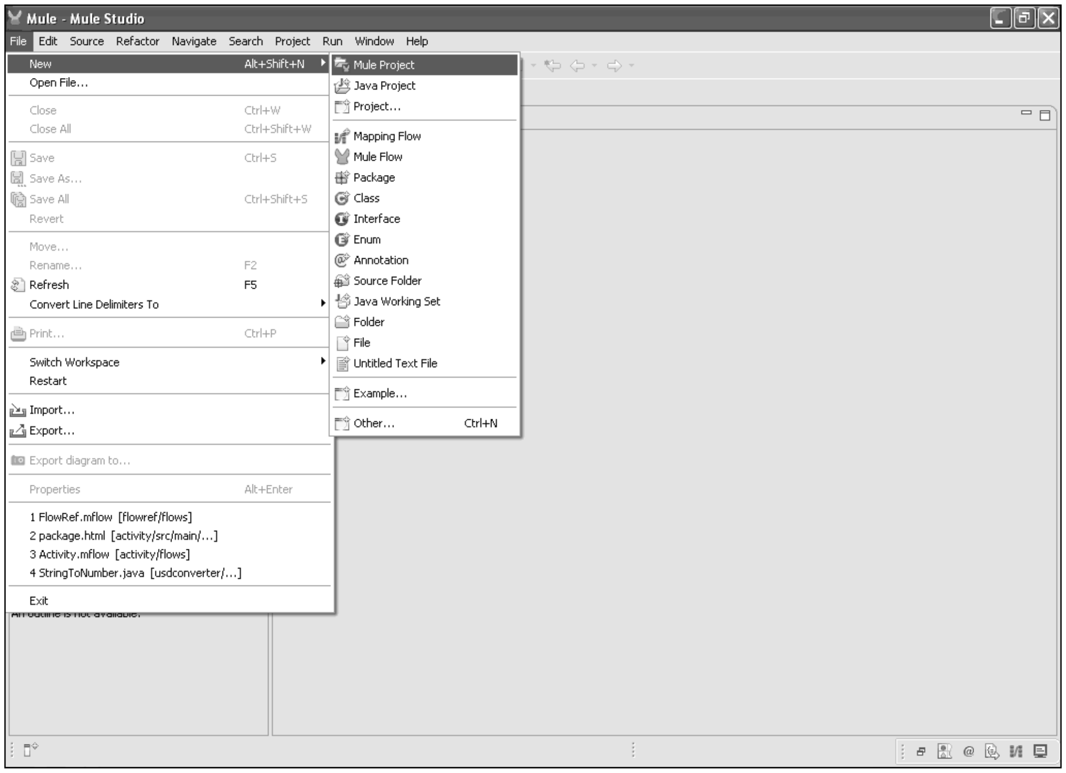
Select a workspace

Mule Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: E:\MuleCookBook

Use this as the default and do not ask again

2. To create a new project, go to **File** | **New** | **Mule Project**. Enter the project name, REST, and click on **Next** and then on **Finish**. Your new project has been created; now you can start the implementation.



## How to do it...

Here we will create a RESTful web service using the annotation. We will create a method named `getwelcomeMsg()`.

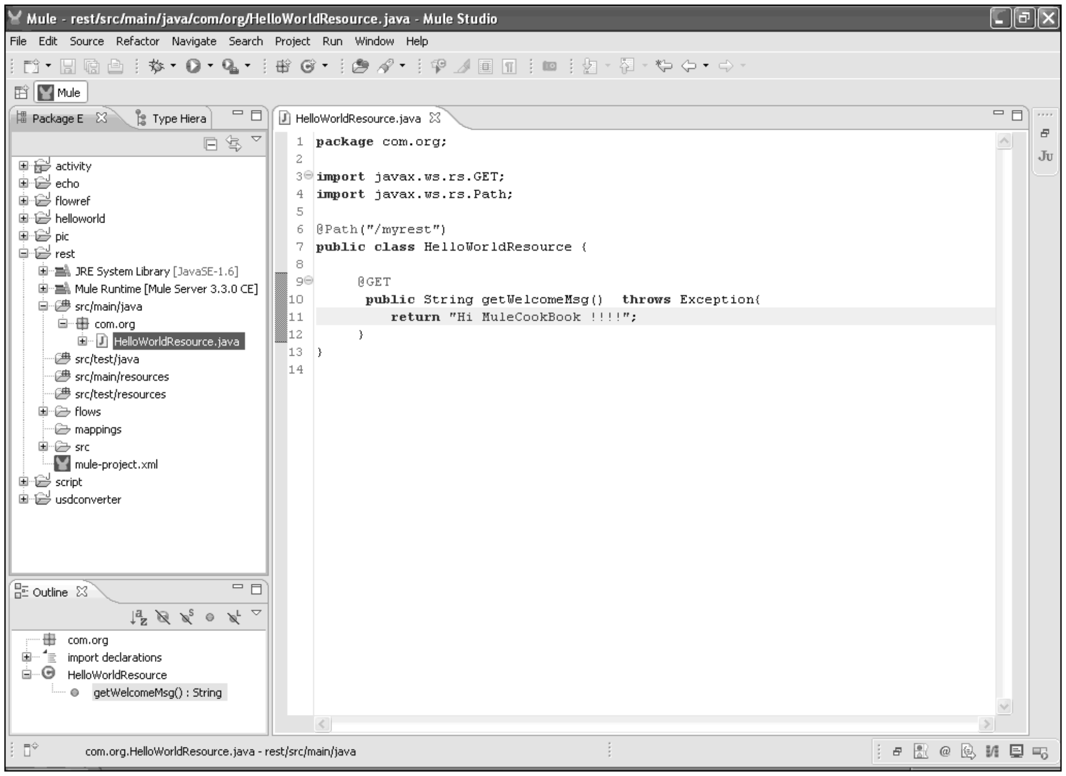
1. To create a class, go to `src/main/java` and right-click on it. Create a class named `HelloWorldResource` to print a message. Enter the package name and click on **Next** and then on **Finish**. Here we have used the JAX-WS annotation. For details on the JAX-WS annotation, you can refer to this URL: `http://publib.boulder.ibm.com/infocenter/radhelp/v7r0m0/index.jsp?topic=/com.ibm.ws.jaxws.emitter.doc/topics/rwsandoc002.html`.

```
package com.org;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
@Path("/myrest")
```

```
public class HelloWorldResource {
  @GET
    public String getWelcomeMsg () throws Exception {
       return "Hi MuleCookBook!!!!";
  }
}
```

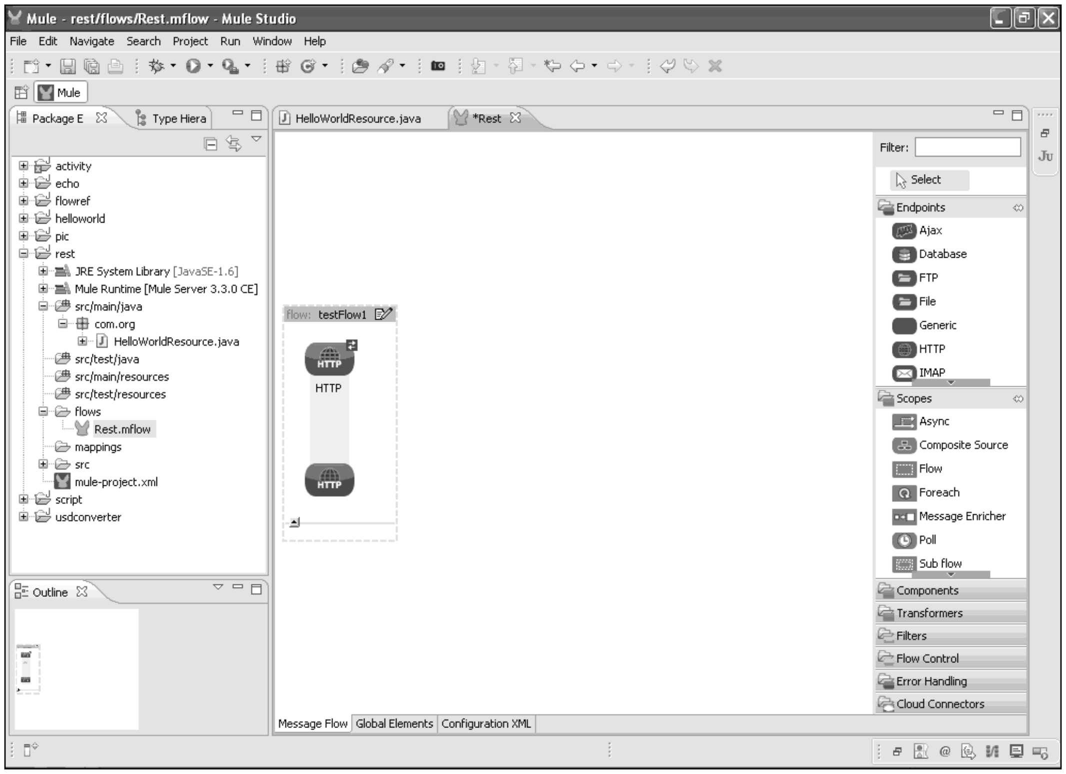
> The `@Get` annotation indicates that the annotated method responds to an HTTP GET request.
>
> The `@Path` annotation is used to map a given URL.

You can see the creation of this method in the following screenshot:

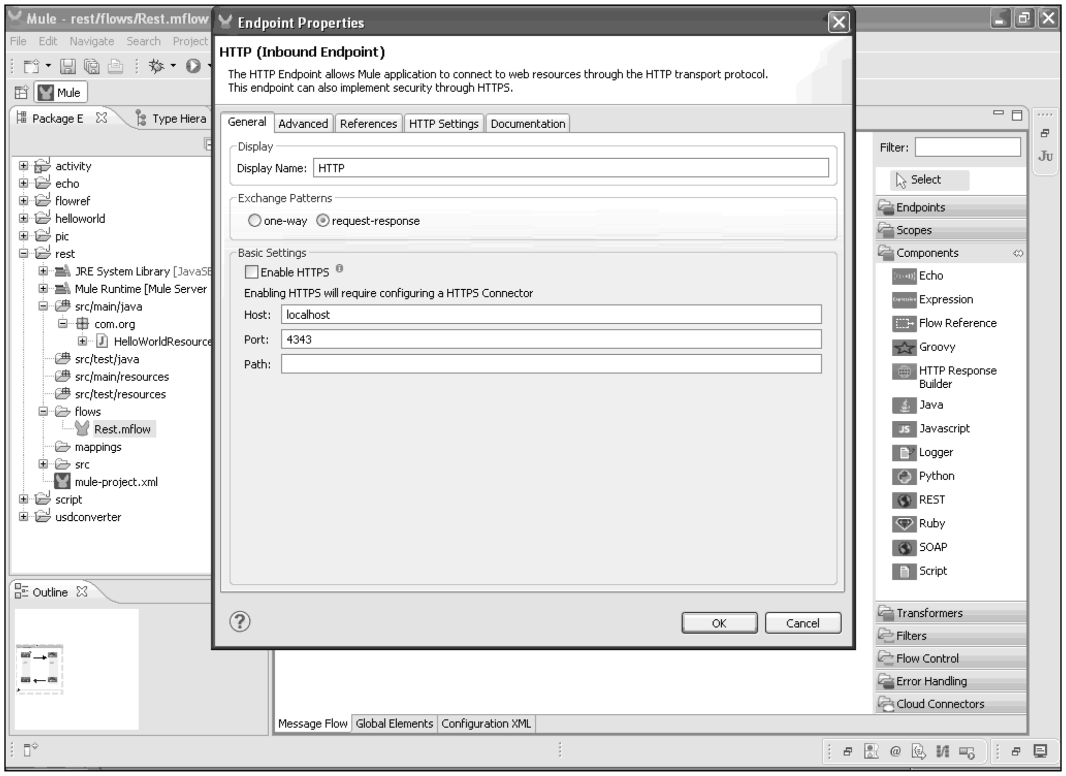


```
package com.org;

import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("/myrest")
public class HelloWorldResource {

     @GET
      public String getWelcomeMsg()  throws Exception{
         return "Hi MuleCookBook !!!!";
      }
}

```

2. To create a flow, go to the `Rest.mflow` file. First of all, you have to drag the **HTTP** Endpoint from the palette and drop it on the canvas area.



3. Double-click on the **HTTP** Endpoint to configure it. You will see the hostname and port number. You can change the **Host** and **Port** field values. These two fields are mandatory. By default, port number **8081** will be taken by the Mule server. We have used port number 4343 here.



Endpoint Properties

HTTP (Inbound Endpoint)

The HTTP Endpoint allows Mule application to connect to web resources through the HTTP transport protocol.
This endpoint can also implement security through HTTPS.

Display

Display Name: HTTP

Exchange Patterns

one-way [✓ request-response]

Basic Settings

Enable HTTPS

Enabling HTTPS will require configuring a HTTPS Connector

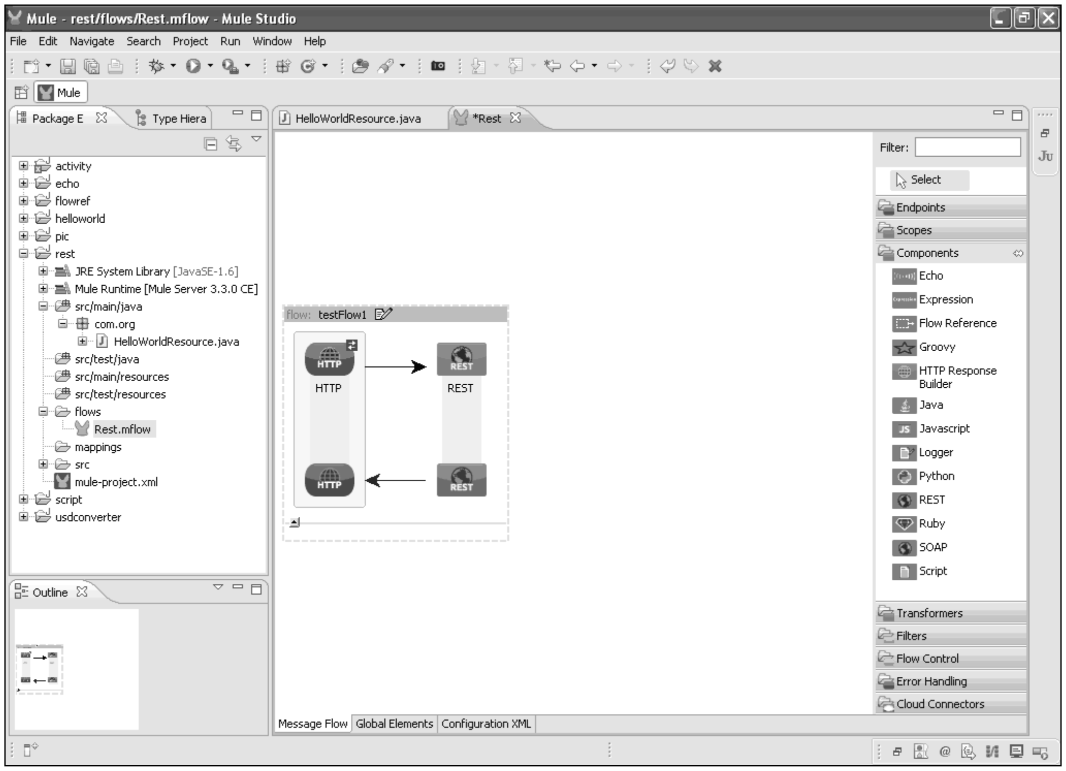
Host: localhost

Port: 4343

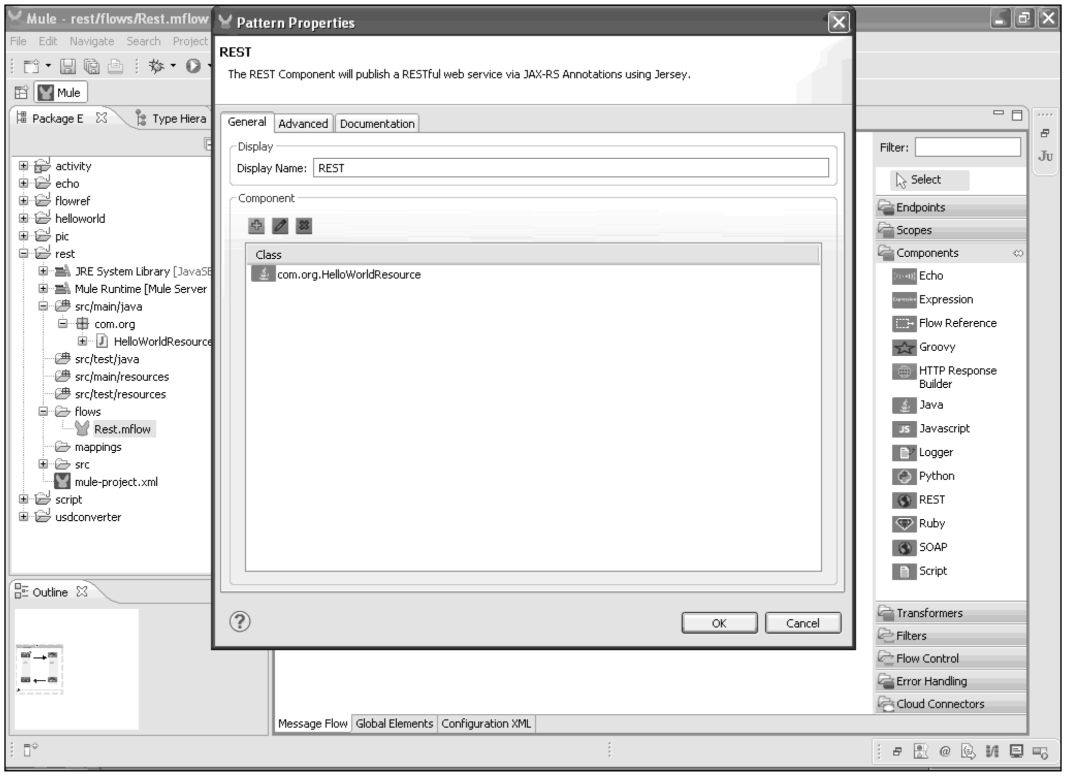
Path:

4. To create a RESTful web service, drag the **REST** component from the palette and drop it on the canvas area. This **REST** component is used to make a REST service available via **Jersey**. Jersey is an open source, production-quality, JAX-RS (JSR 311) reference implementation for building RESTful web services.

   REST is the formalized architecture of HTTP and is based on concepts of resources, links, and a uniform interface. It uses the HTTP protocol. We can create a web service using the **REST** component.



5. Double-click on the **REST** component to configure it. Here you can add a **Java** component that was created before.



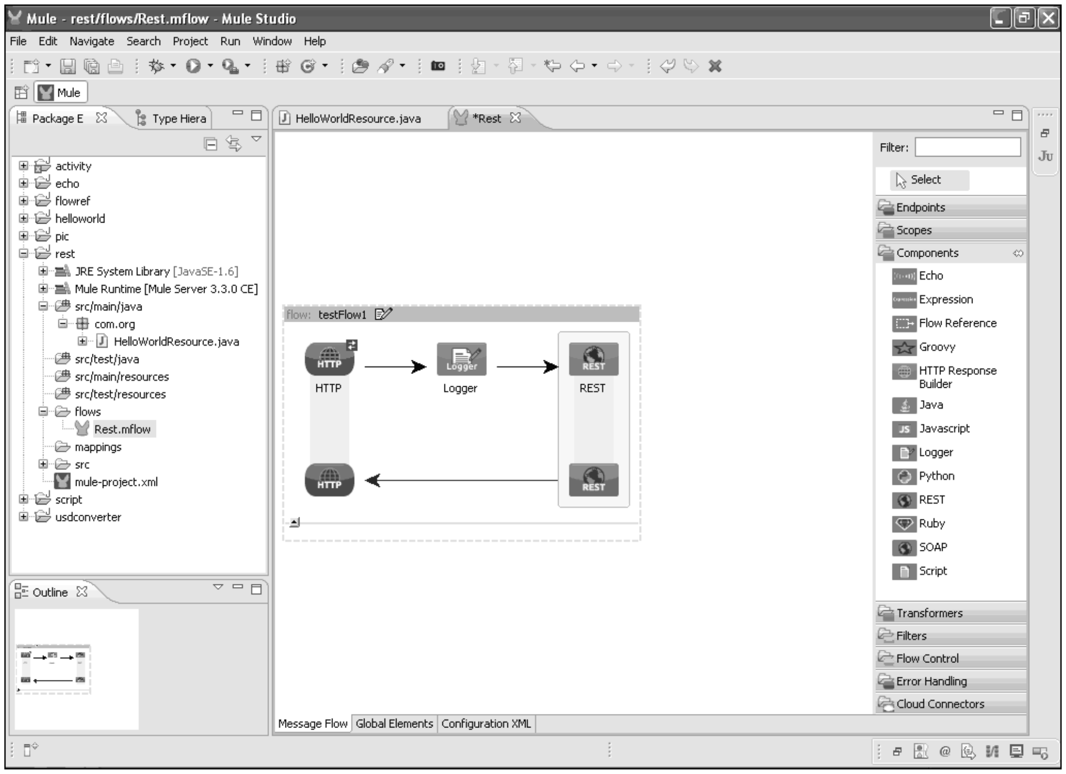
REST

The REST Component will publish a RESTful web service via JAX-RS Annotations using Jersey.

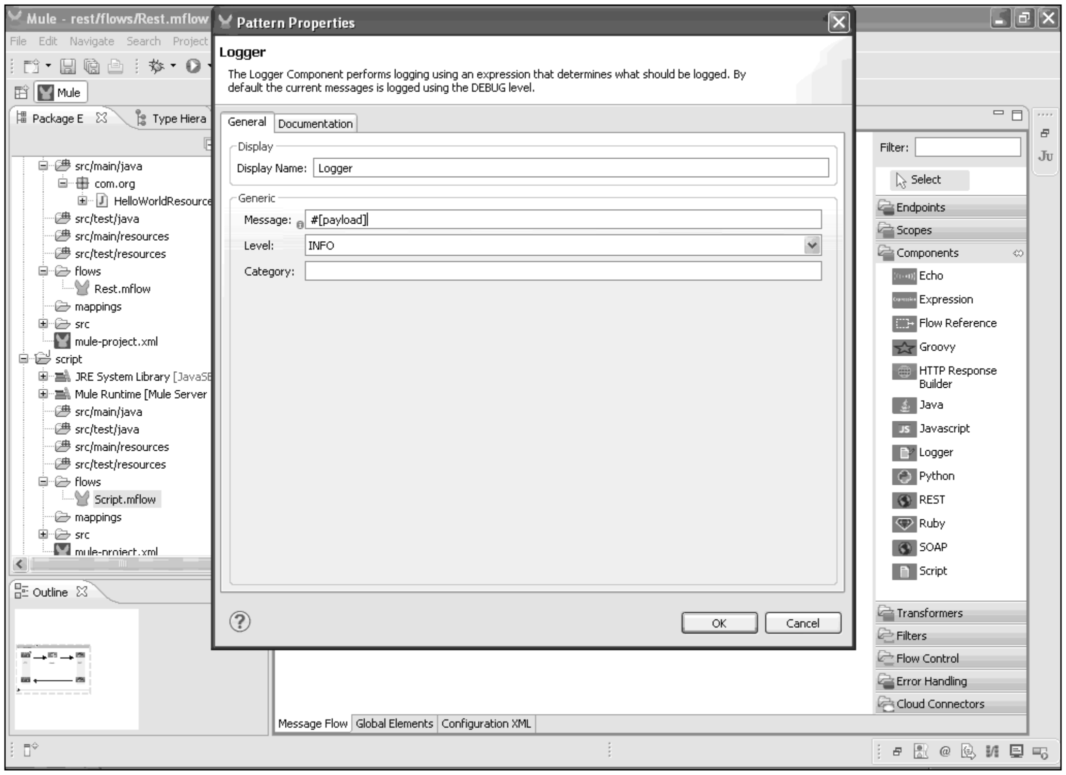
Display Name: REST

com.org.HelloWorldResource

6. To display messages on the console, drag-and-drop the **Logger** component from the palette onto the canvas area and configure it.



7. To configure the **Logger** component, double-click on it. You will see the **Message:** textbox; just enter the payload expression, `#[payload]`.



Pattern Properties

Logger

The Logger Component performs logging using an expression that determines what should be logged. By default the current messages is logged using the DEBUG level.

Display Name: Logger

Message: #[payload]

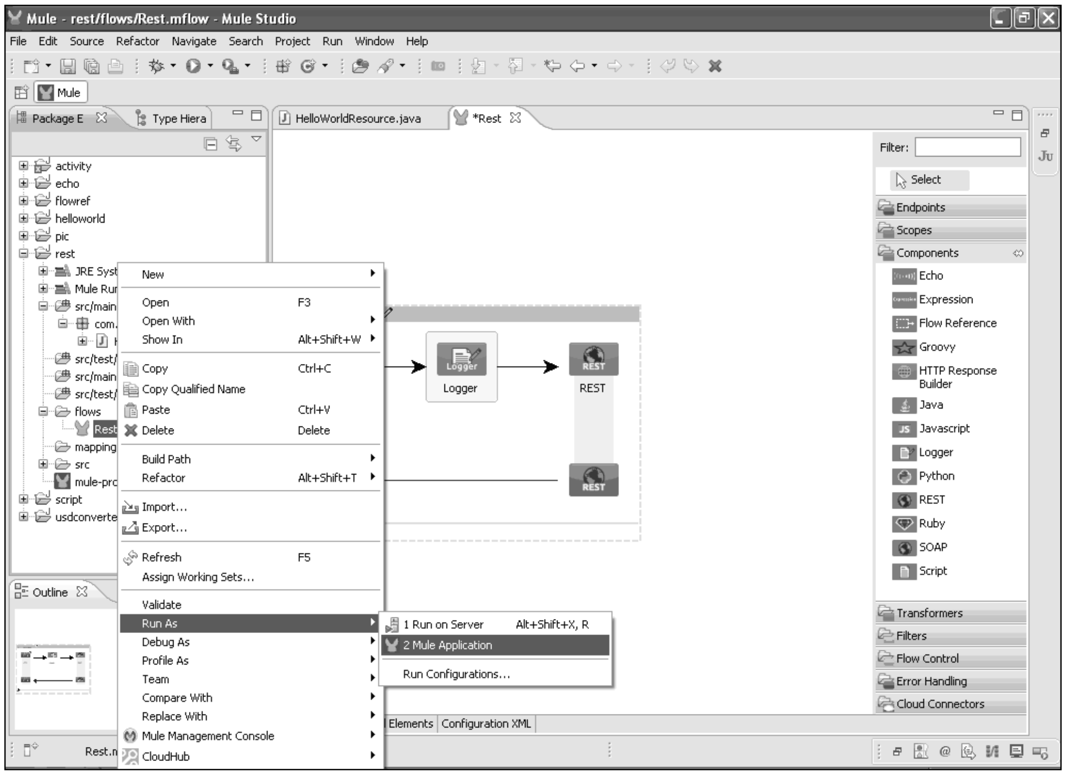
Level: INFO

Category:

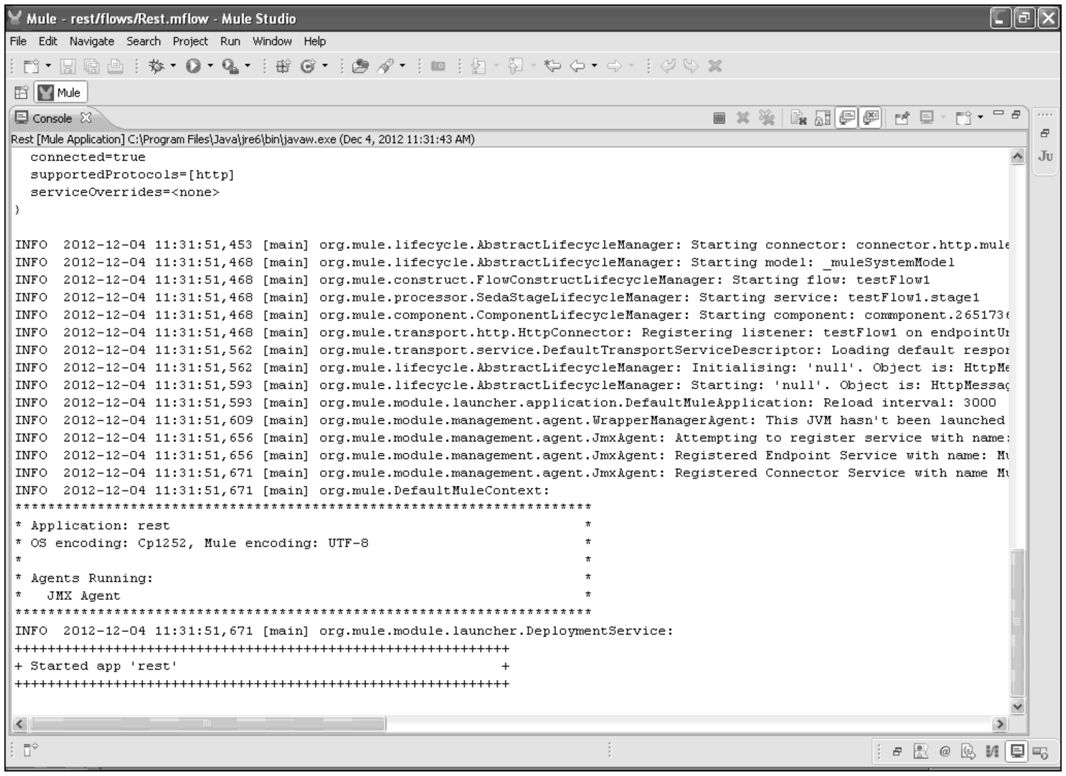
## How it works...

To deploy your application go through the following steps:

1. If you haven't saved your application code, do save it. After saving your project, right-click on the `Echo.mflow` file and go to **Run As** | **Mule Application**.

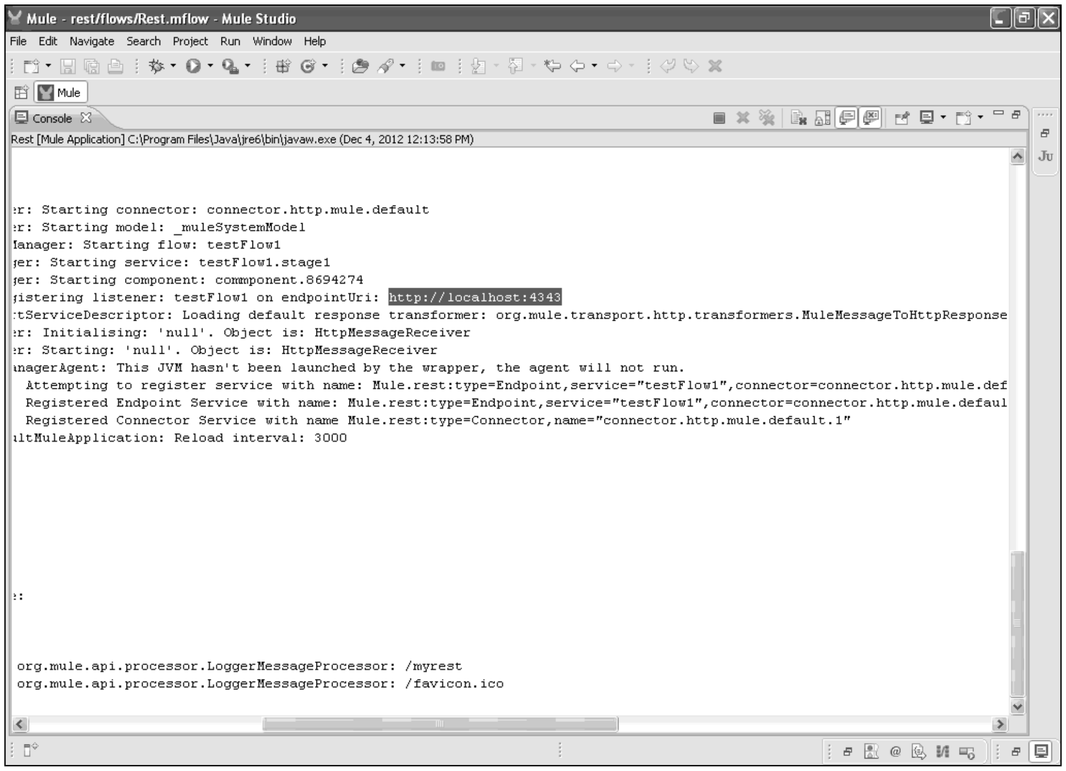


2. If your application code is successfully deployed, you will see the message `Started app 'helloworld'` on the console.


Mule - rest/flows/Rest.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console
Rest [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Dec 4, 2012 11:31:43 AM)

```
  connected=true
  supportedProtocols=[http]
  serviceOverrides=<none>
}

INFO  2012-12-04 11:31:51,453 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting connector: connector.http.mule
INFO  2012-12-04 11:31:51,468 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting model: _muleSystemModel
INFO  2012-12-04 11:31:51,468 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: testFlow1
INFO  2012-12-04 11:31:51,468 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: testFlow1.stage1
INFO  2012-12-04 11:31:51,468 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.265173
INFO  2012-12-04 11:31:51,468 [main] org.mule.transport.http.HttpConnector: Registering listener: testFlow1 on endpointU
INFO  2012-12-04 11:31:51,562 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default respo
INFO  2012-12-04 11:31:51,562 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMe
INFO  2012-12-04 11:31:51,593 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessa
INFO  2012-12-04 11:31:51,593 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
INFO  2012-12-04 11:31:51,609 [main] org.mule.module.management.agent.WrapperManagerAgent: This JVM hasn't been launched
INFO  2012-12-04 11:31:51,656 [main] org.mule.module.management.agent.JmxAgent: Attempting to register service with name:
INFO  2012-12-04 11:31:51,656 [main] org.mule.module.management.agent.JmxAgent: Registered Endpoint Service with name: M
INFO  2012-12-04 11:31:51,671 [main] org.mule.module.management.agent.JmxAgent: Registered Connector Service with name M
INFO  2012-12-04 11:31:51,671 [main] org.mule.DefaultMuleContext:
************************************************************************
* Application: rest                                                    *
* OS encoding: Cp1252, Mule encoding: UTF-8                            *
*                                                                      *
* Agents Running:                                                      *
*   JMX Agent                                                          *
************************************************************************
INFO  2012-12-04 11:31:51,671 [main] org.mule.module.launcher.DeploymentService:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Started app 'rest'                                         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

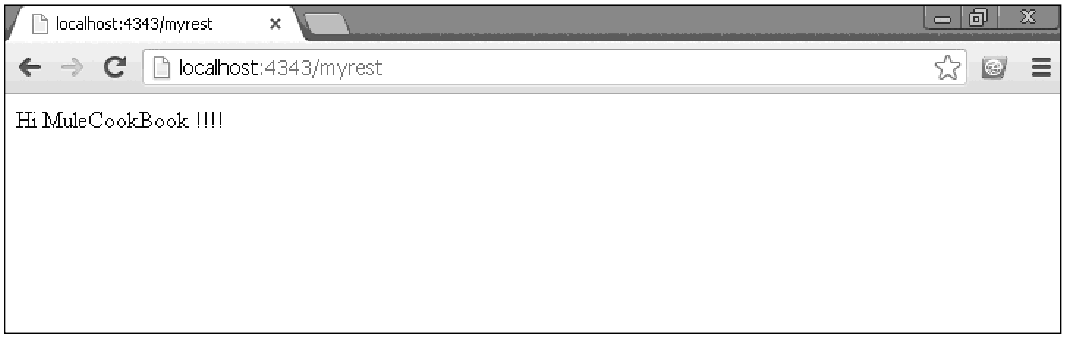
3. Copy the URL `http://localhost:4343/myrest` and paste it in your browser.


Mule - rest/flows/Rest.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console

Rest [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Dec 4, 2012 12:13:58 PM)

```
er: Starting connector: connector.http.mule.default
er: Starting model: _muleSystemModel
lanager: Starting flow: testFlow1
ger: Starting service: testFlow1.stage1
ger: Starting component: commponent.8694274
gistering listener: testFlow1 on endpointUri: http://localhost:4343
:tServiceDescriptor: Loading default response transformer: org.mule.transport.http.transformers.MuleMessageToHttpResponse
er: Initialising: 'null'. Object is: HttpMessageReceiver
er: Starting: 'null'. Object is: HttpMessageReceiver
inagerAgent: This JVM hasn't been launched by the wrapper, the agent will not run.
  Attempting to register service with name: Mule.rest:type=Endpoint,service="testFlow1",connector=connector.http.mule.def
  Registered Endpoint Service with name: Mule.rest:type=Endpoint,service="testFlow1",connector=connector.http.mule.defaul
  Registered Connector Service with name Mule.rest:type=Connector,name="connector.http.mule.default.1"
iltMuleApplication: Reload interval: 3000




e:



 org.mule.api.processor.LoggerMessageProcessor: /myrest
 org.mule.api.processor.LoggerMessageProcessor: /favicon.ico
```

4. To see the output, paste the URL on your browser and type `/myrest`; this is required because we have used the `@Path` annotation in the custom Java class.



Hi MuleCookBook !!!!

## Publishing a SOAP-based web service using the SOAP component

The Mule **SOAP** component is used for publishing, consuming, and proxying of SOAP web services within a Mule flow. Using the SOAP component, you can also enable Web Service Security. Apache CXF is an open source services framework. CXF helps you build services using frontend programming APIs such as JAX-WS and JAX-RS.

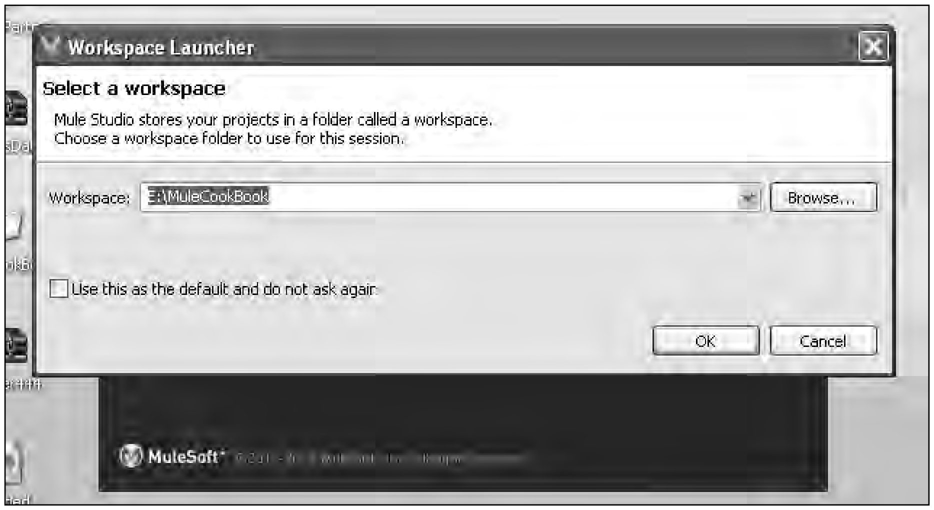
You can create a CXF web service by configuring a SOAP component in your Mule flow to perform any of the following CXF web service operations:

- Publish a simple service
- Publish a JAX-WS service
- Proxy a published service
- Consume a service using a simple client
- Consume a service using the JAX-WS client
- Proxy to a service

### Getting ready

In this example, we will see how to create a SOAP-based web service using the SOAP component. To create a SOAP web service, we'll use three components: HTTP, Java, and SOAP.

1. Open Mule Studio and enter the workspace name as shown in the following screenshot:

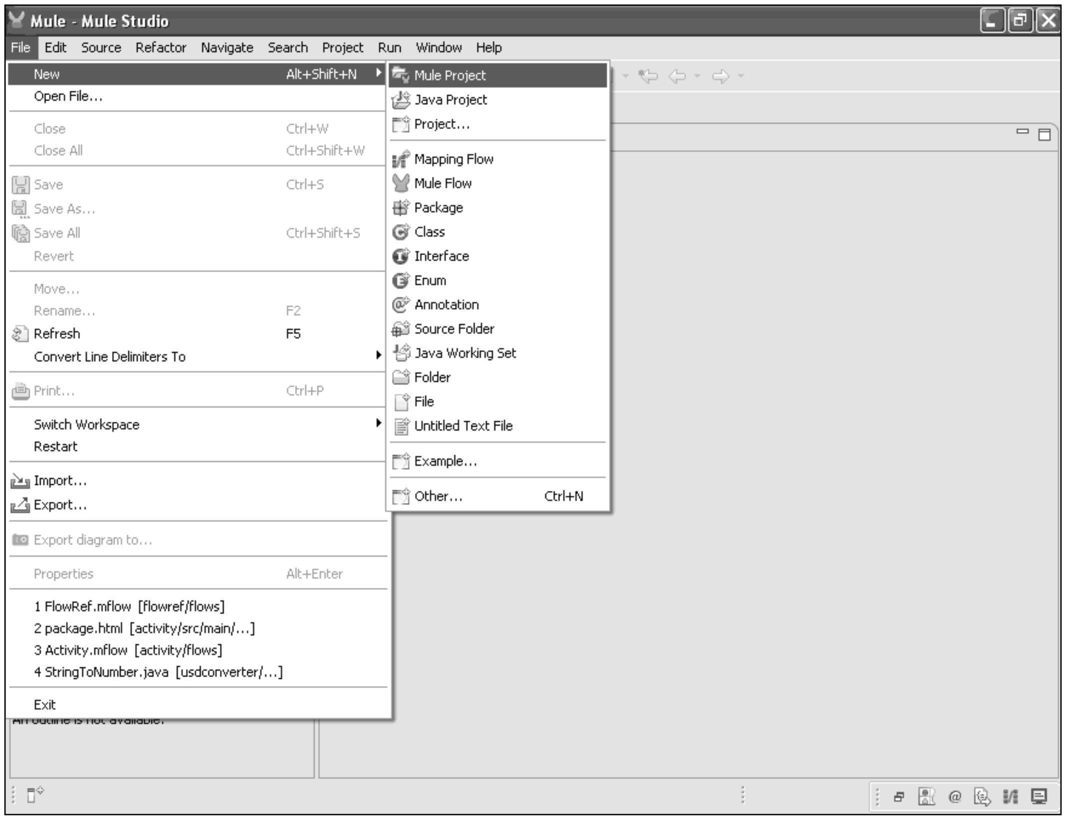

Workspace Launcher

Select a workspace

Mule Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: E:\MuleCookBook

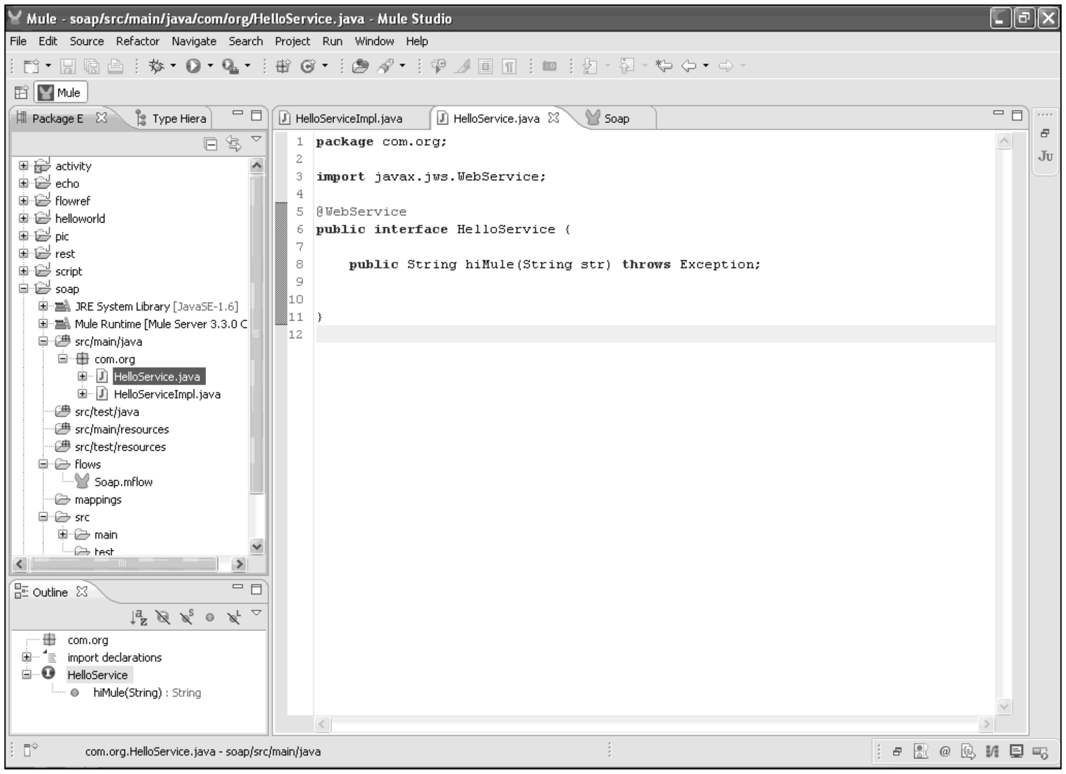
2. To create a new project, go to **File** | **New** | **Mule Project**. Enter the project name, SOAP, and click on **Next** and then on **Finish**. Your new project is created and you are now ready to start the implementation.



3. To create a class, go to `src/main/java`, right-click on it, and go to **New** | **Interface**. Create an interface named `HelloService` under the package `com.org`. Here, we create the `hiMule` method and set its return type to `String`.

```
package com.org;
import javax.jws.WebService;
@WebService
public interface HelloService {
  public String hiMule(String str) throws Exception;
}
```

You can see the creation of this class in the following screenshot:



```
package com.org;

import javax.jws.WebService;

@WebService
public interface HelloService {

    public String hiMule(String str) throws Exception;


}
```

## How to do it...

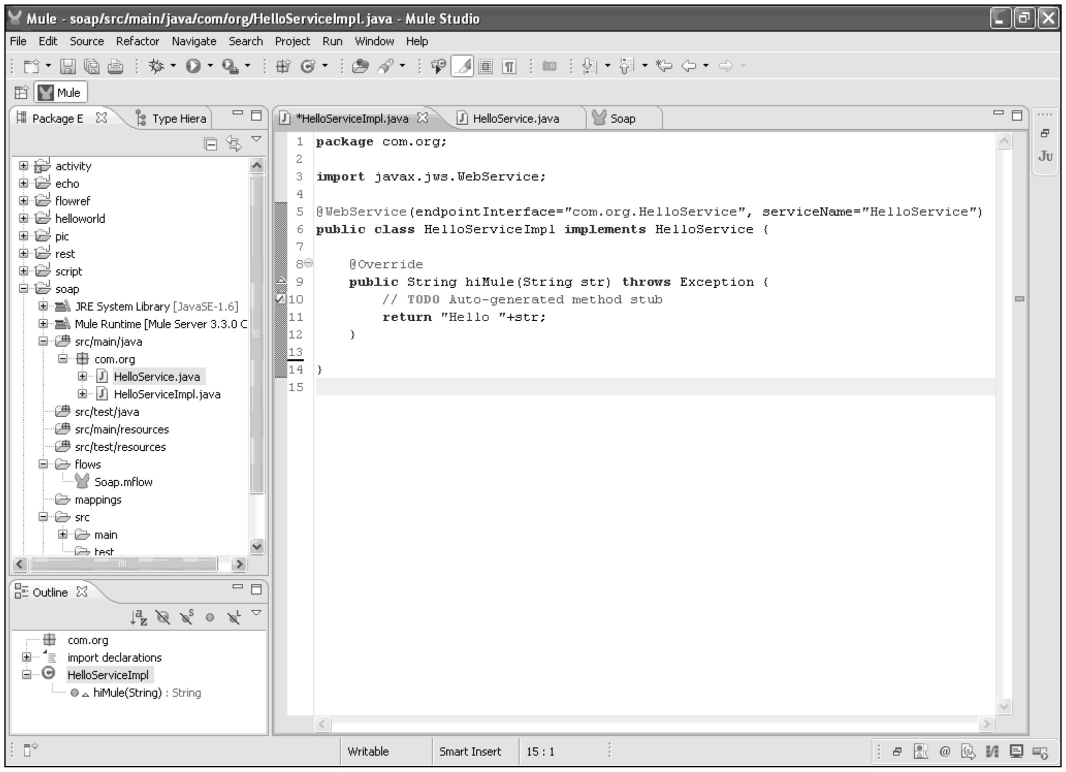
In this section, you will refer to the JAX-WS annotation and create the method `hiMule()`. We will use this method to generate the output using a browser.

1. Create a class called `HelloServiceImpl` under the same package directory (`com.org`) and implement it with the interface. Here, we have used the `@WebService` annotation to create a SOAP-based web service and override the `hiMule` method. You can refer to this URL for more information on the JAX-WS annotation: `http://publib.boulder.ibm.com/infocenter/radhelp/v7r0m0/index.jsp?topic=/com.ibm.ws.jaxws.emitter.doc/topics/rwsandoc002.html`.

```
package com.org;
import javax.jws.WebService;
@WebService(EndpointInterface="com.org.HelloService",
serviceName="HelloService")
public class HelloServiceImpl implements HelloService {
  @Override
```

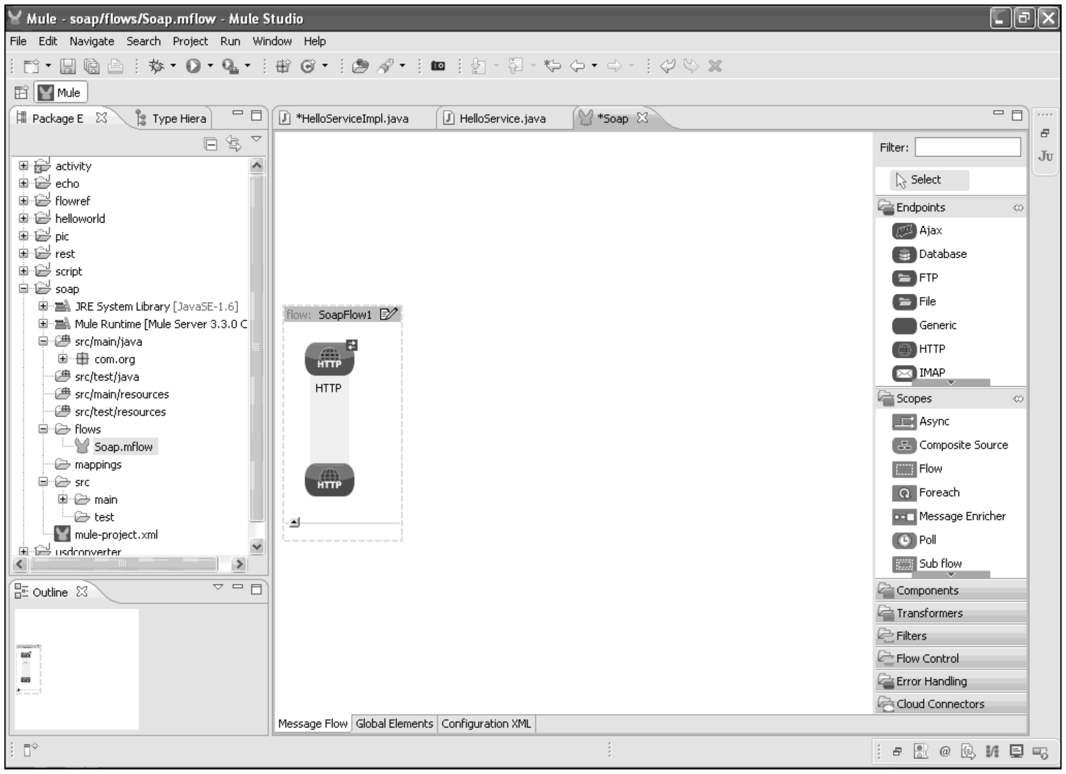
```
    public String hiMule(String str) throws Exception {
      // TODO Auto-generated method stub
      return "Hello "+str;
    }
  }
```

Through the `@WebService` annotation, we call the `HelloService` interface and also provide a service name.

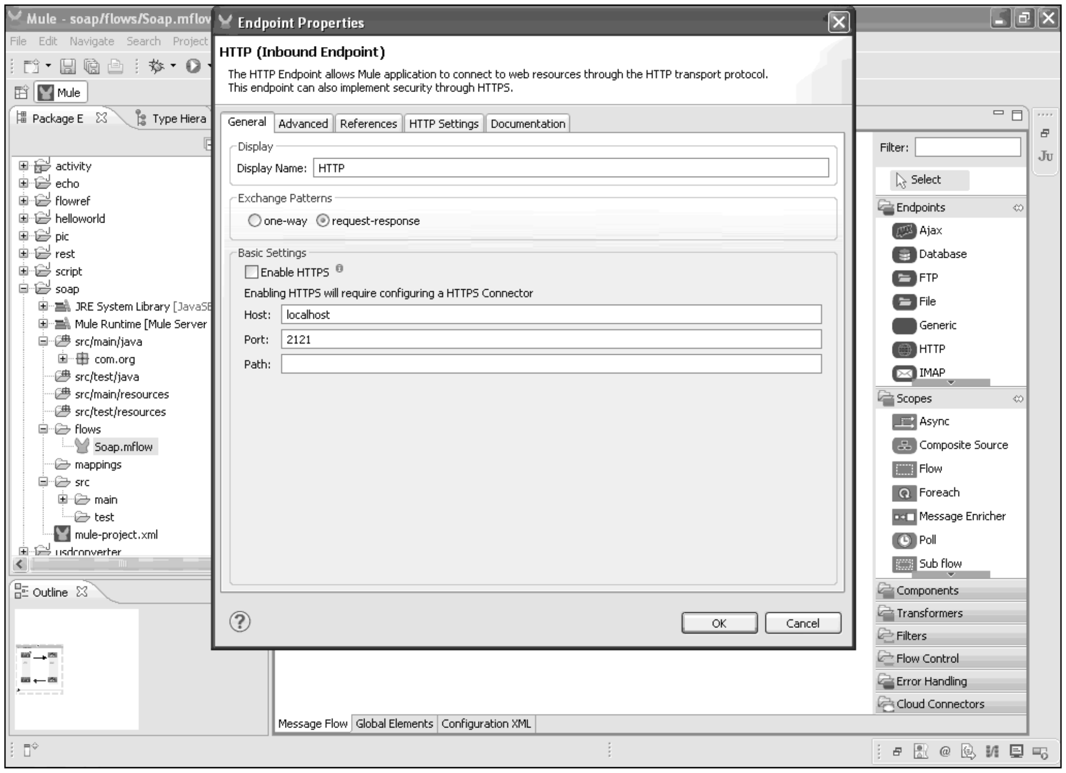


```
package com.org;

import javax.jws.WebService;

@WebService(endpointInterface="com.org.HelloService", serviceName="HelloService")
public class HelloServiceImpl implements HelloService {

    @Override
    public String hiMule(String str) throws Exception {
        // TODO Auto-generated method stub
        return "Hello "+str;
    }

}
```

2. To create a flow, go to the `Soap.mflow` file. Drag the **HTTP** Endpoint from the palette and drop it onto the canvas. You have to configure the **HTTP** Endpoint.



3. Double-click on the **HTTP** Endpoint to configure it. You will see the **Host** and **Port** fields. These two fields are mandatory. In this example, we used `localhost` as the hostname. By default, port number **8081** will be taken by the Mule server. We have used port number 2121 here.



Endpoint Properties

HTTP (Inbound Endpoint)

The HTTP Endpoint allows Mule application to connect to web resources through the HTTP transport protocol.
This endpoint can also implement security through HTTPS.

Display Name: HTTP

Exchange Patterns: one-way [✓ request-response]

Enable HTTPS

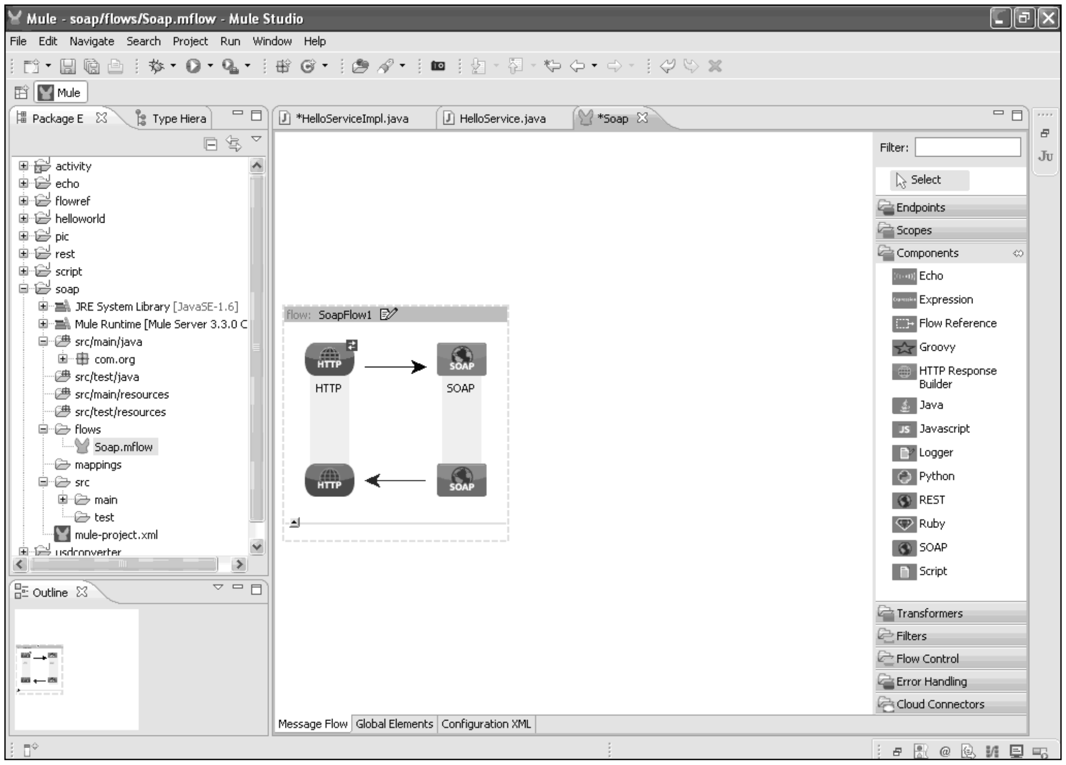
Enabling HTTPS will require configuring a HTTPS Connector

Host: localhost

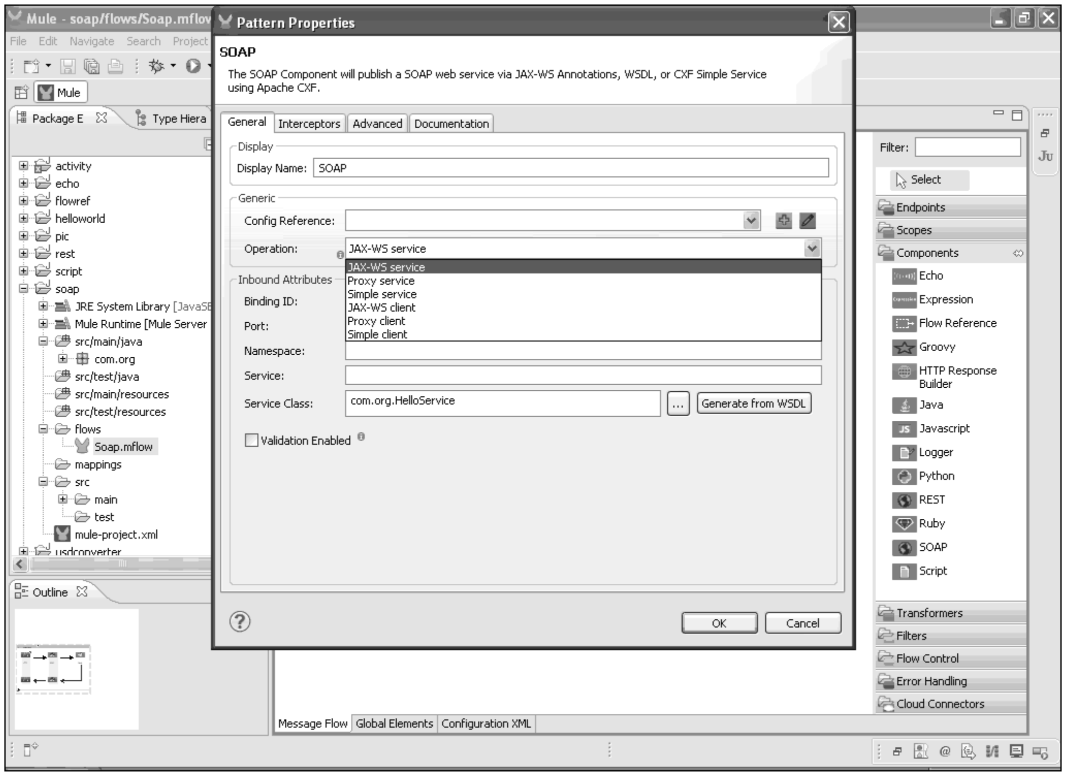
Port: 2121

Path:

4. To create a SOAP-based web service, drag the **SOAP** component from the palette, drop it onto the canvas, and configure it. Here, we create web services using the **SOAP** component.



5. Double-click on the **SOAP** component to configure it. We select the **JAX-WS** services for the operation and then we will import the `HelloService` interface that was created before.


Pattern Properties

SOAP

The SOAP Component will publish a SOAP web service via JAX-WS Annotations, WSDL, or CXF Simple Service using Apache CXF.

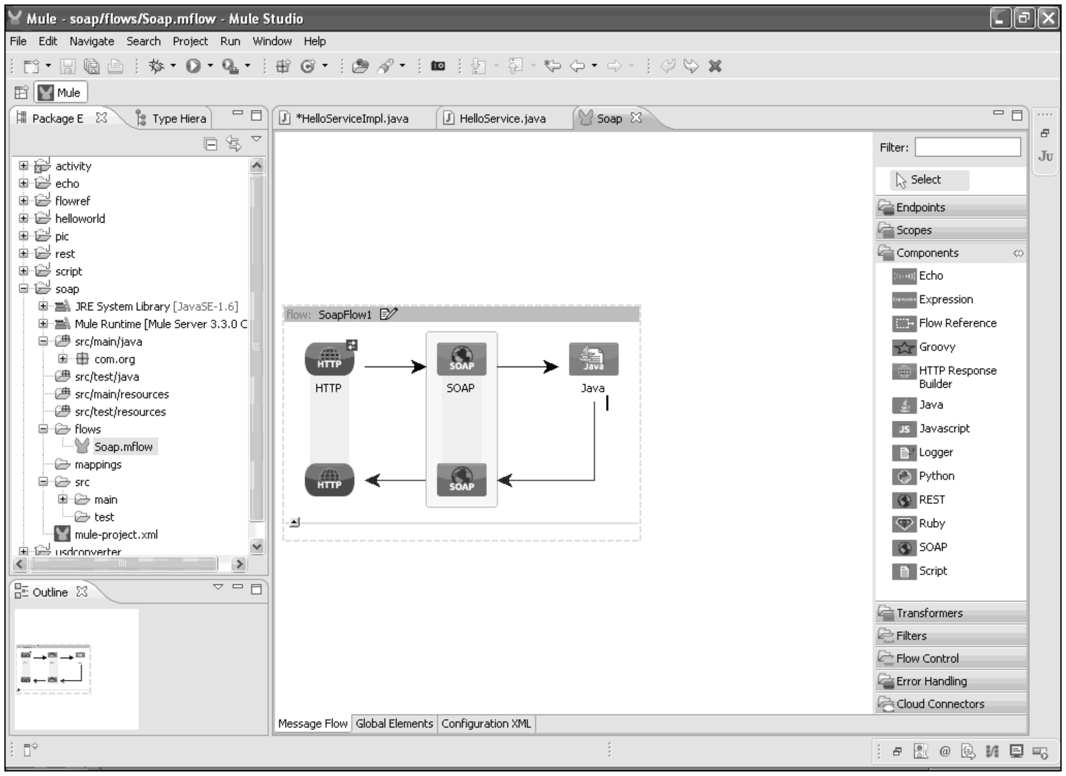
Display Name: SOAP

Config Reference:

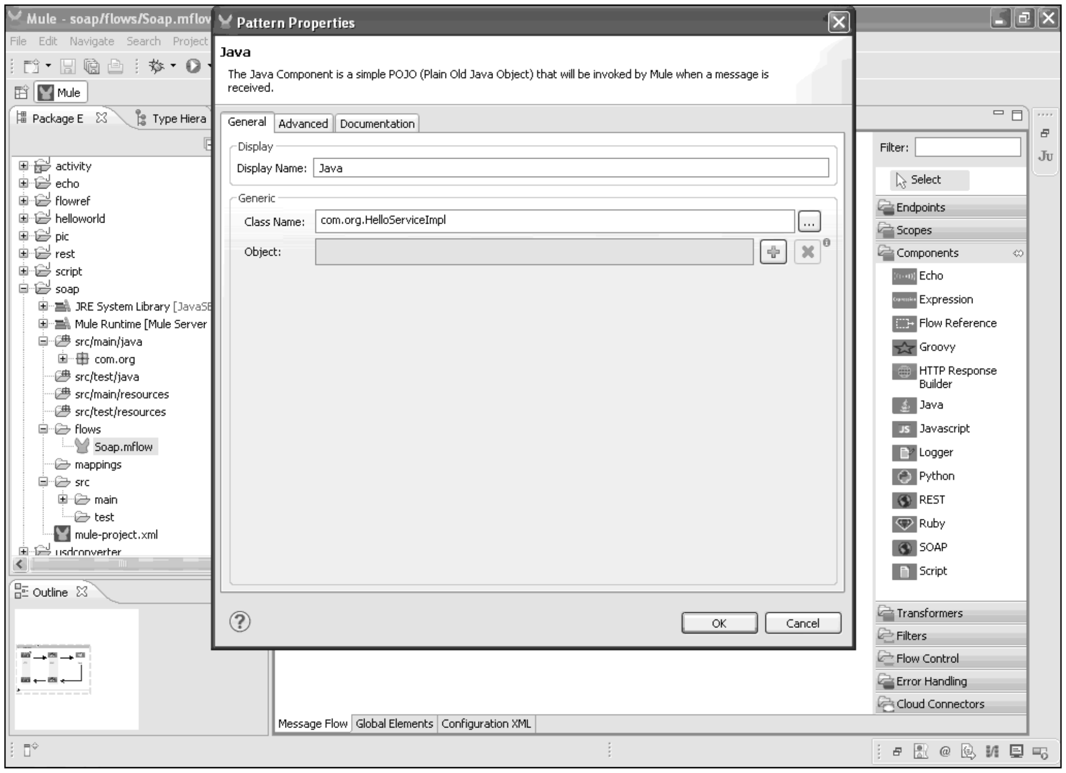
Operation: JAX-WS service

Service Class: com.org.HelloService

6. To import a class, drag the **Java** component from the palette, drop it on the canvas area, and configure it. If you want to change its name, you can do so.



7. Double-click on the **Java** component to configure it. Here, we import the `HelloServiceImpl` class that was created before.



Pattern Properties

Java

The Java Component is a simple POJO (Plain Old Java Object) that will be invoked by Mule when a message is received.

Display Name: Java

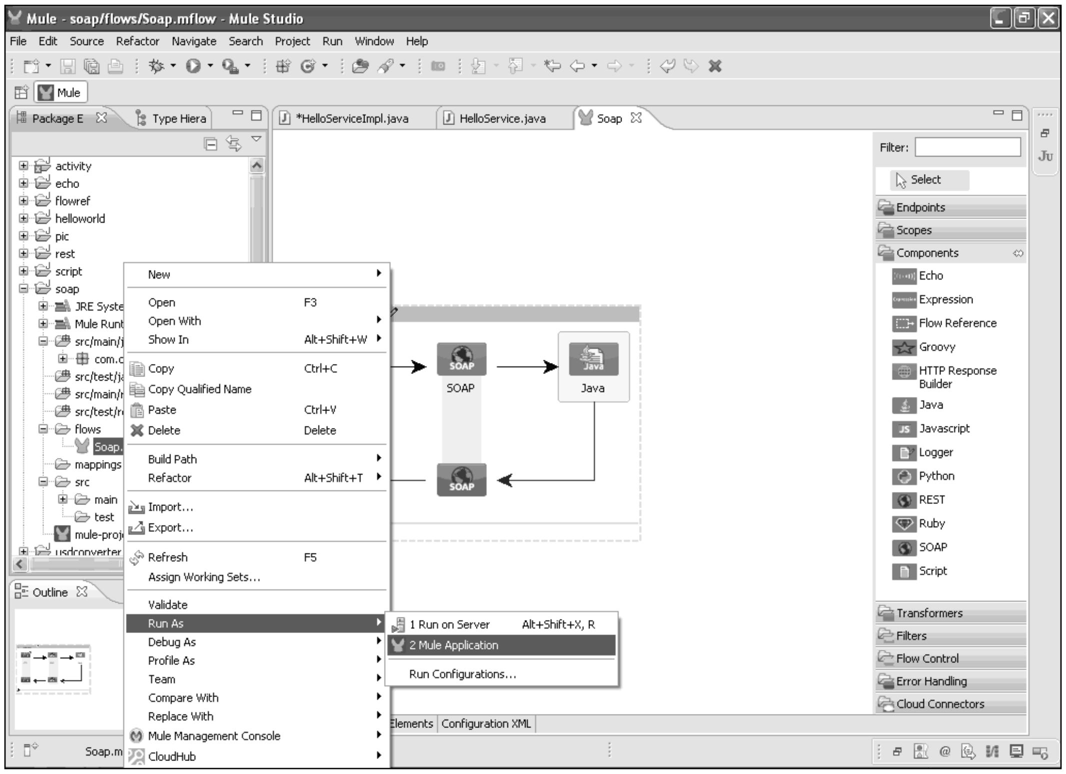
Class Name: com.org.HelloServiceImpl

Object:

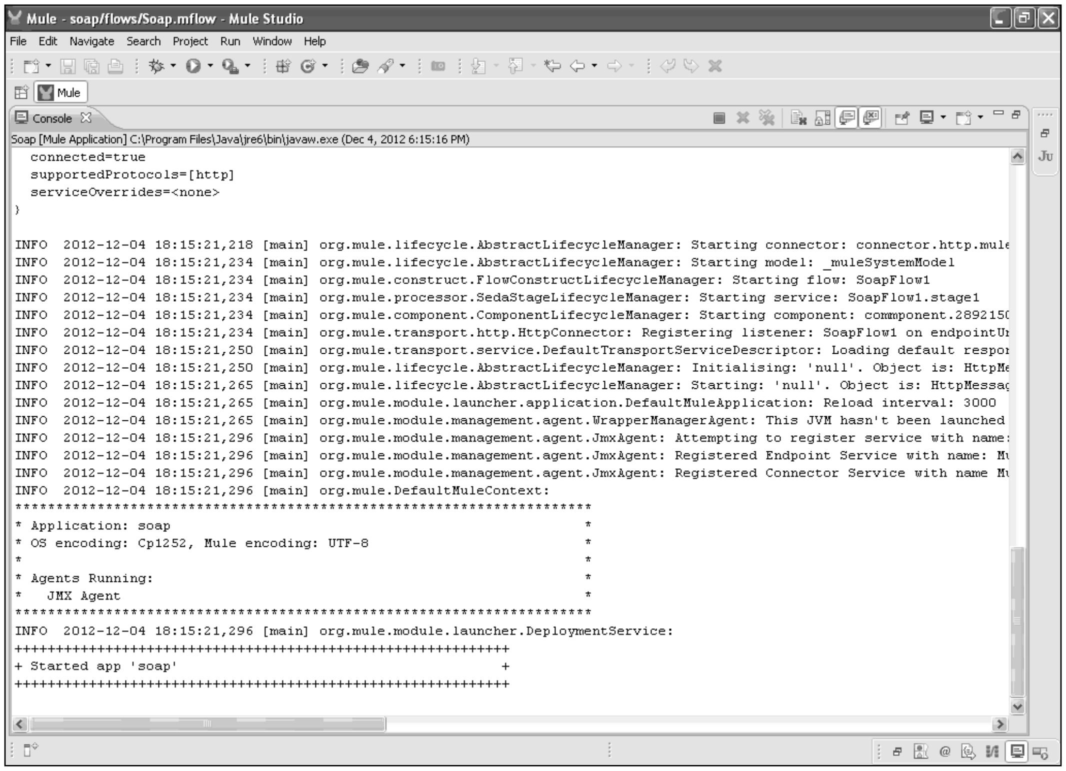
## How it works...

To deploy your application, right-click on your `.mflow` file and deploy your Mule application by performing the following steps:

1. If you haven't saved your application code, do save it. After saving your project, right-click on the `Echo.mflow` file and go to **Run As** | **Mule Application**.

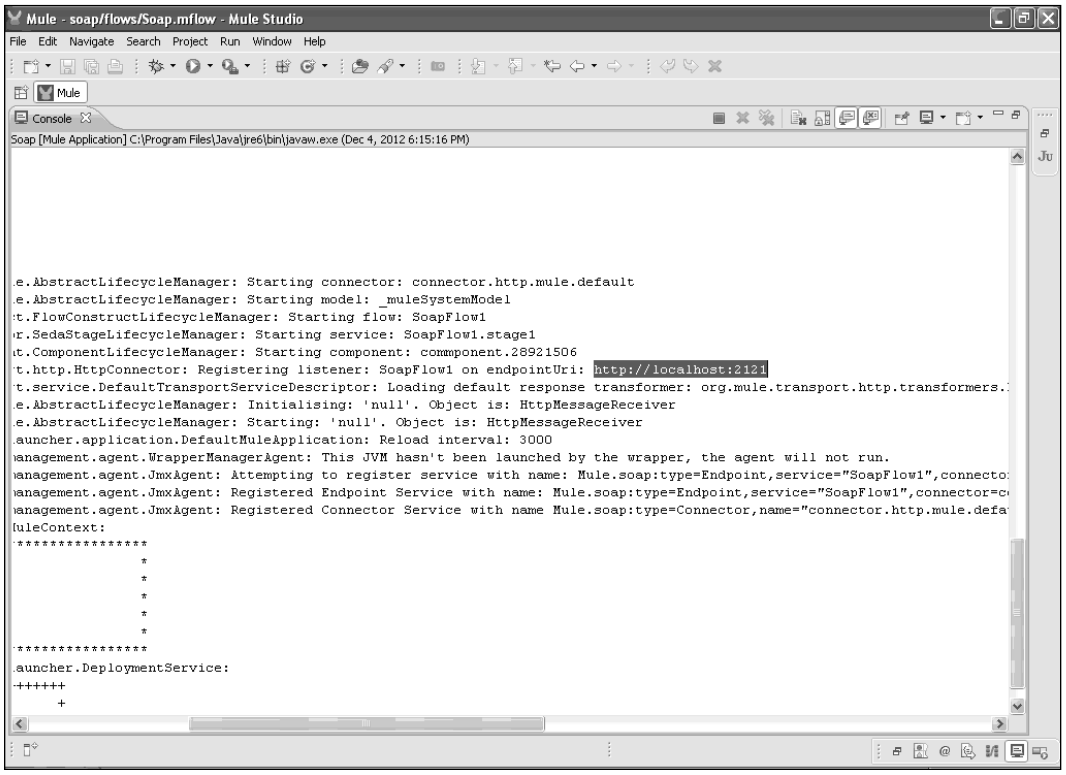


2. If your application code is successfully deployed, you will see the message `Started app 'helloworld'` on the console.


Mule - soap/flows/Soap.mflow - Mule Studio
File Edit Navigate Search Project Run Window Help
Mule
Console

```
Soap [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Dec 4, 2012 6:15:16 PM)
  connected=true
  supportedProtocols=[http]
  serviceOverrides=<none>
}

INFO  2012-12-04 18:15:21,218 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting connector: connector.http.mule
INFO  2012-12-04 18:15:21,234 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting model: _muleSystemModel
INFO  2012-12-04 18:15:21,234 [main] org.mule.construct.FlowConstructLifecycleManager: Starting flow: SoapFlow1
INFO  2012-12-04 18:15:21,234 [main] org.mule.processor.SedaStageLifecycleManager: Starting service: SoapFlow1.stage1
INFO  2012-12-04 18:15:21,234 [main] org.mule.component.ComponentLifecycleManager: Starting component: commponent.289215C
INFO  2012-12-04 18:15:21,234 [main] org.mule.transport.http.HttpConnector: Registering listener: SoapFlow1 on endpointUr
INFO  2012-12-04 18:15:21,250 [main] org.mule.transport.service.DefaultTransportServiceDescriptor: Loading default respor
INFO  2012-12-04 18:15:21,250 [main] org.mule.lifecycle.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMe
INFO  2012-12-04 18:15:21,265 [main] org.mule.lifecycle.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessag
INFO  2012-12-04 18:15:21,265 [main] org.mule.module.launcher.application.DefaultMuleApplication: Reload interval: 3000
INFO  2012-12-04 18:15:21,265 [main] org.mule.module.management.agent.WrapperManagerAgent: This JVM hasn't been launched
INFO  2012-12-04 18:15:21,296 [main] org.mule.module.management.agent.JmxAgent: Attempting to register service with name:
INFO  2012-12-04 18:15:21,296 [main] org.mule.module.management.agent.JmxAgent: Registered Endpoint Service with name: Mu
INFO  2012-12-04 18:15:21,296 [main] org.mule.module.management.agent.JmxAgent: Registered Connector Service with name Mu
INFO  2012-12-04 18:15:21,296 [main] org.mule.DefaultMuleContext:
**********************************************************************
* Application: soap                                                  *
* OS encoding: Cp1252, Mule encoding: UTF-8                          *
*                                                                    *
* Agents Running:                                                    *
*   JMX Agent                                                        *
**********************************************************************
INFO  2012-12-04 18:15:21,296 [main] org.mule.module.launcher.DeploymentService:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ Started app 'soap'                                       +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

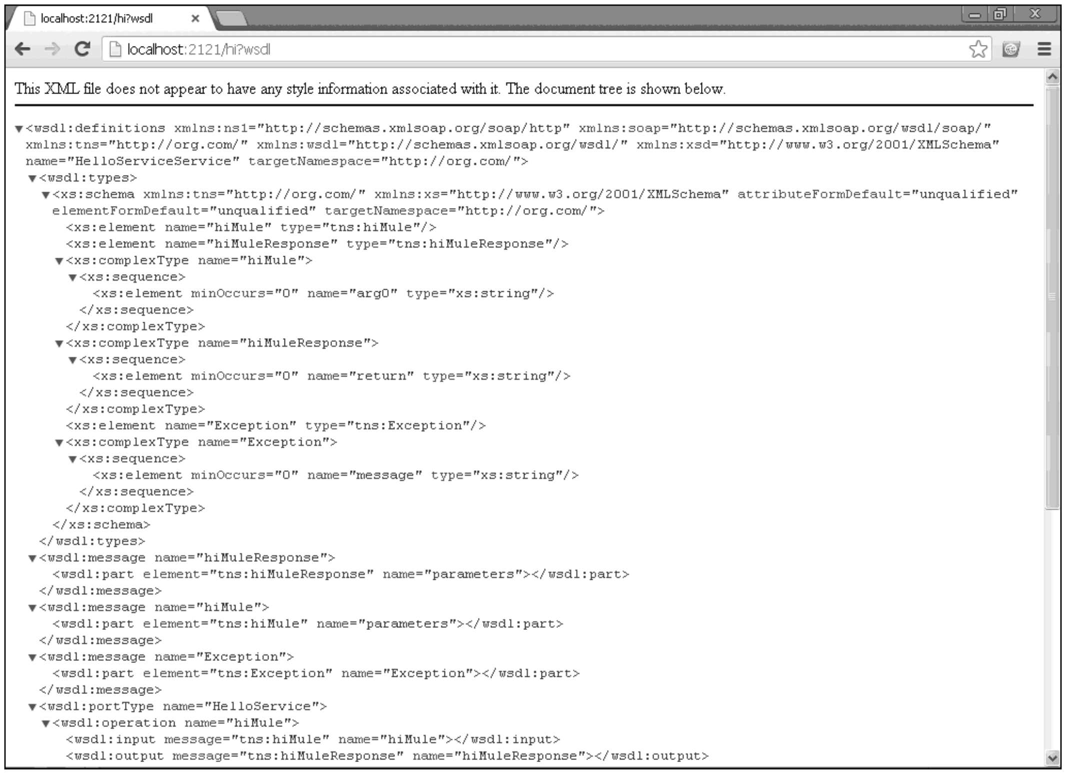
3. Copy the URL `http://localhost:2121/` and paste it on your browser.



Soap [Mule Application] C:\Program Files\Java\jre6\bin\javaw.exe (Dec 4, 2012 6:15:16 PM)

```
.e.AbstractLifecycleManager: Starting connector: connector.http.mule.default
.e.AbstractLifecycleManager: Starting model: _muleSystemModel
:t.FlowConstructLifecycleManager: Starting flow: SoapFlow1
r.SedaStageLifecycleManager: Starting service: SoapFlow1.stage1
t.ComponentLifecycleManager: Starting component: commponent.28921506
t.http.HttpConnector: Registering listener: SoapFlow1 on endpointUri: http://localhost:2121
t.service.DefaultTransportServiceDescriptor: Loading default response transformer: org.mule.transport.http.transformers.
.e.AbstractLifecycleManager: Initialising: 'null'. Object is: HttpMessageReceiver
.e.AbstractLifecycleManager: Starting: 'null'. Object is: HttpMessageReceiver
.auncher.application.DefaultMuleApplication: Reload interval: 3000
anagement.agent.WrapperManagerAgent: This JVM hasn't been launched by the wrapper, the agent will not run.
anagement.agent.JmxAgent: Attempting to register service with name: Mule.soap:type=Endpoint,service="SoapFlow1",connecto
anagement.agent.JmxAgent: Registered Endpoint Service with name: Mule.soap:type=Endpoint,service="SoapFlow1",connector=c
anagement.agent.JmxAgent: Registered Connector Service with name Mule.soap:type=Connector,name="connector.http.mule.defa
uleContext:
****************
               *
               *
               *
               *
               *
****************
.auncher.DeploymentService:
++++++
     +
```

4. To see the output, paste the URL on your browser and type in `/hi?wsdl`; here, `wsdl` stands for **Web Services Description Language**.



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wsdl:definitions xmlns:ns1="http://schemas.xmlsoap.org/soap/http" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://org.com/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 name="HelloServiceService" targetNamespace="http://org.com/">
  ▼<wsdl:types>
    ▼<xs:schema xmlns:tns="http://org.com/" xmlns:xs="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
     elementFormDefault="unqualified" targetNamespace="http://org.com/">
       <xs:element name="hiMule" type="tns:hiMule"/>
       <xs:element name="hiMuleResponse" type="tns:hiMuleResponse"/>
      ▼<xs:complexType name="hiMule">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="arg0" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
      ▼<xs:complexType name="hiMuleResponse">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="return" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
       <xs:element name="Exception" type="tns:Exception"/>
      ▼<xs:complexType name="Exception">
        ▼<xs:sequence>
           <xs:element minOccurs="0" name="message" type="xs:string"/>
         </xs:sequence>
       </xs:complexType>
     </xs:schema>
   </wsdl:types>
  ▼<wsdl:message name="hiMuleResponse">
     <wsdl:part element="tns:hiMuleResponse" name="parameters"></wsdl:part>
   </wsdl:message>
  ▼<wsdl:message name="hiMule">
     <wsdl:part element="tns:hiMule" name="parameters"></wsdl:part>
   </wsdl:message>
  ▼<wsdl:message name="Exception">
     <wsdl:part element="tns:Exception" name="Exception"></wsdl:part>
   </wsdl:message>
  ▼<wsdl:portType name="HelloService">
    ▼<wsdl:operation name="hiMule">
       <wsdl:input message="tns:hiMule" name="hiMule"></wsdl:input>
       <wsdl:output message="tns:hiMuleResponse" name="hiMuleResponse"></wsdl:output>
```