Routebox

Routebox Component

Available as of Camel 2.6

Routebox subject for change

The Routebox component will be revisited in upcoming releases to see if it can be further simplified, be more intuitive and user friendly. The related Context component may be regardes as the simpler component. This component may be @deprecated in favor of Context.

The **routebox** component enables the creation of specialized endpoints that offer encapsulation and a strategy based indirection service to a collection of camel routes hosted in an automatically created or user injected camel context.

Routebox endpoints are camel endpoints that may be invoked directly on camel routes. The routebox endpoint performs the following key functions

- encapsulation - acts as a blackbox, hosting a collection of camel routes stored in an inner camel context. The inner context is fully under the control of the routebox component and is **JVM bound**.
- strategy based indirection - direct payloads sent to the routebox endpoint along a camel route to specific inner routes based on a user defined internal routing strategy or a dispatch map.
- exchange propagation - forward exchanges modified by the routebox endpoint to the next segment of the camel route.

The routebox component supports both consumer and producer endpoints.

Producer endpoints are of two flavors

- Producers that send or dispatch incoming requests to a external routebox consumer endpoint
- Producers that directly invoke routes in an internal embedded camel context thereby not sending requests to an external consumer.

Maven users will need to add the following dependency to their `pom.xml` for this component:

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-routebox</artifactId>
    <version>x.x.x</version>
    <!-- use the same version as your Camel core version -->
</dependency>
```

The need for a Camel Routebox endpoint

The routebox component is designed to ease integration in complex environments needing

- a large collection of routes and
- involving a wide set of endpoint technologies needing integration in different ways

In such environments, it is often necessary to craft an integration solution by creating a sense of layering among camel routes effectively organizing them into

- Coarse grained or higher level routes - aggregated collection of inner or lower level routes exposed as Routebox endpoints that represent an integration focus area. For example

Focus Area	Coarse grained Route Examples
Department Focus	HR routes, Sales routes etc
Supply chain & B2B Focus	Shipping routes, Fulfillment routes, 3rd party services etc
Technology Focus	Database routes, JMS routes, Scheduled batch routes etc

- Fine grained routes - routes that execute a singular and specific business and/or integration pattern.

Requests sent to Routebox endpoints on coarse grained routes can then delegate requests to inner fine grained routes to achieve a specific integration objective, collect the final inner result, and continue to progress to the next step along the coarse-grained route.

URI format

```
routebox:routeboxname[?options]
```

You can append query options to the URI in the following format, `?option=value&option=value&...`

Options

Name	Default Value	Description
dispatchStrategy	null	A string representing a key in the Camel Registry matching an object value implementing the interface *org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy*
dispatchMap	null	A string representing a key in the Camel Registry matching an object value of the type HashMap<String, String>. The HashMap key should contain strings that can be matched against the value set for the exchange header **ROUTE_DISPATCH_KEY**. The HashMap value should contain inner route consumer URI's to which requests should be directed.
innerContext	auto created	A string representing a key in the Camel Registry matching an object value of the type *org.apache.camel.CamelContext*. If a CamelContext is not provided by the user a CamelContext is automatically created for deployment of inner routes.
innerRegistry	null	A string representing a key in the Camel Registry matching an object value that implements the interface *org.apache.camel.spi.Registry*. If Registry values are utilized by inner routes to create endpoints, an innerRegistry parameter must be provided
routeBuilders	empty List	A string representing a key in the Camel Registry matching an object value of the type *List<org.apache.camel.builder.RouteBuilder>*. If the user does not supply an innerContext pre-primed with inner routes, the routeBuilders option must be provided as a non-empty list of RouteBuilders containing inner routes
innerProtocol	Direct	The Protocol used internally by the Routebox component. Can be Direct or SEDA. **The Routebox component currently offers protocols that are JVM bound.**
sendToConsumer	true	Dictates whether a Producer endpoint sends a request to an external routebox consumer. If the setting is false, the Producer creates an embedded inner context and processes requests internally.
forkContext	true	The Protocol used internally by the Routebox component. Can be Direct or SEDA. **The Routebox component currently offers protocols that are JVM bound.**
threads	20	Number of threads to be used by the routebox to receive requests. **Setting applicable only for innerProtocol SEDA**.
queueSize	unlimited	Create a fixed size queue to receive requests. **Setting applicable only for innerProtocol SEDA**.

Sending/Receiving Messages to/from the routebox

Before sending requests it is necessary to properly configure the routebox by loading the required URI parameters into the Registry as shown below. In the case of Spring, if the necessary beans are declared correctly, the registry is automatically populated by Camel.

Step 1: Loading inner route details into the Registry

```
@Override
protected JndiRegistry createRegistry() throws Exception {
    JndiRegistry registry = new JndiRegistry(createJndiContext());

    // Wire the routeDefinitions & dispatchStrategy to the outer camelContext where the routebox is declared
    List<RouteBuilder> routes = new ArrayList<RouteBuilder>();
    routes.add(new SimpleRouteBuilder());
    registry.bind("registry", createInnerRegistry());
    registry.bind("routes", routes);

    // Wire a dispatch map to registry
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("addToCatalog", "seda:addToCatalog");
    map.put("findBook", "seda:findBook");
    registry.bind("map", map);

    // Alternatively wiring a dispatch strategy to the registry
    registry.bind("strategy", new SimpleRouteDispatchStrategy());

    return registry;
}

private JndiRegistry createInnerRegistry() throws Exception {
    JndiRegistry innerRegistry = new JndiRegistry(createJndiContext());
    BookCatalog catalogBean = new BookCatalog();
    innerRegistry.bind("library", catalogBean);

    return innerRegistry;
}
...
CamelContext context = new DefaultCamelContext(createRegistry());
```

Step 2: Optionaly using a Dispatch Strategy instead of a Dispatch Map

Using a dispatch Strategy involves implementing the interface *org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy* as shown in the example below.

```
public class SimpleRouteDispatchStrategy implements RouteboxDispatchStrategy {

    /* (non-Javadoc)
     * @see org.apache.camel.component.routebox.strategy.RouteboxDispatchStrategy#selectDestinationUri(java.
util.List, org.apache.camel.Exchange)
     */
    public URI selectDestinationUri(List<URI> activeDestinations,
            Exchange exchange) {
        URI dispatchDestination = null;

        String operation = exchange.getIn().getHeader("ROUTE_DISPATCH_KEY", String.class);
        for (URI destination : activeDestinations) {
            if (destination.toASCIIString().equalsIgnoreCase("seda:" + operation)) {
                dispatchDestination = destination;
                break;
            }
        }

        return dispatchDestination;
    }
}
```

Step 2: Launching a routebox consumer

When creating a route consumer, note that the # entries in the routeboxUri are matched to the created inner registry, routebuilder list and dispatchStrategy /dispatchMap in the CamelContext Registry. Note that all routebuilders and associated routes are launched in the routebox created inner context

```
private String routeboxUri = "routebox:multipleRoutes?
innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map";

public void testRouteboxRequests() throws Exception {
    CamelContext context = createCamelContext();
    template = new DefaultProducerTemplate(context);
    template.start();

    context.addRoutes(new RouteBuilder() {
        public void configure() {
            from(routeboxUri)
                .to("log:Routes operation performed?showAll=true");
        }
    });
    context.start();

    // Now use the ProducerTemplate to send the request to the routebox
    template.requestBodyAndHeader(routeboxUri, book, "ROUTE_DISPATCH_KEY", "addToCatalog");
}
```

Step 3: Using a routebox producer

When sending requests to the routebox, it is not necessary for producers do not need to know the inner route endpoint URI and they can simply invoke the Routebox URI endpoint with a dispatch strategy or dispatchMap as shown below

It is necessary to set a special exchange Header called **ROUTE_DISPATCH_KEY** (optional for Dispatch Strategy) with a key that matches a key in the dispatch map so that the request can be sent to the correct inner route

```
from("direct:sendToStrategyBasedRoutebox")
    .to("routebox:multipleRoutes?innerRegistry=#registry&routeBuilders=#routes&dispatchStrategy=#strategy")
    .to("log:Routes operation performed?showAll=true");

from ("direct:sendToMapBasedRoutebox")
    .setHeader("ROUTE_DISPATCH_KEY", constant("addToCatalog"))
    .to("routebox:multipleRoutes?innerRegistry=#registry&routeBuilders=#routes&dispatchMap=#map")
    .to("log:Routes operation performed?showAll=true");
```