Context

Context Component

Available as of Camel 2.7

> ⊙ Deprecated do NOT use

The **context** component allows you to create new Camel Components from a CamelContext with a number of routes which is then treated as a black box, allowing you to refer to the local endpoints within the component from other CamelContexts.

It is similar to the Routebox component in idea, though the Context component tries to be really simple for end users; just a simple convention over configuration approach to refer to local endpoints inside the CamelContext Component.

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

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

URI format

```
context:camelContextId:localEndpointName[?options]
```

Or you can omit the "context:" prefix.

```
camelContextId:localEndpointName[?options]
```

- **camelContextId** is the ID you used to register the CamelContext into the Registry.
- **localEndpointName** can be a valid Camel URI evaluated within the black box CamelContext. Or it can be a logical name which is mapped to any local endpoints. For example if you locally have endpoints like **direct:invoices** and **seda:purchaseOrders** inside a CamelContext of id **supplyChain**, then you can just use the URIs **supplyChain:invoices** or **supplyChain:purchaseOrders** to omit the physical endpoint kind and use pure logical URIs.

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

Example

In this example we'll create a black box context, then we'll use it from another CamelContext.

Defining the context component

First you need to create a CamelContext, add some routes in it, start it and then register the CamelContext into the Registry (JNDI, Spring, Guice or OSGi etc).

This can be done in the usual Camel way from this test case (see the createRegistry() method); this example shows Java and JNDI being used...

```
// lets create our black box as a camel context and a set of routes
DefaultCamelContext blackBox = new DefaultCamelContext(registry);
blackBox.setName("blackBox");
blackBox.addRoutes(new RouteBuilder() {
    @Override
    public void configure() throws Exception {
        // receive purchase orders, lets process it in some way then send an invoice
        // to our invoice endpoint
        from("direct:purchaseOrder").
          setHeader("received").constant("true").
          to("direct:invoice");
    }
});
blackBox.start();

registry.bind("accounts", blackBox);
```

Notice in the above route we are using pure local endpoints (**direct** and **seda**). Also note we expose this CamelContext using the **accounts** ID. We can do the same thing in Spring via

```
<camelContext id="accounts" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:purchaseOrder"/>
    ...
    <to uri="direct:invoice"/>
  </route>
</camelContext>
```

Using the context component

Then in another CamelContext we can then refer to this "accounts black box" by just sending to **accounts:purchaseOrder** and consuming from **accounts:invoice**.

If you prefer to be more verbose and explicit you could use **context:accounts:purchaseOrder** or even **context:accounts:direct://purchaseOrder** if you prefer. But using logical endpoint URIs is preferred as it hides the implementation detail and provides a simple logical naming scheme.

For example if we wish to then expose this accounts black box on some middleware (outside of the black box) we can do things like...

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <!-- consume from an ActiveMQ into the black box -->
    <from uri="activemq:Accounts.PurchaseOrders"/>
    <to uri="accounts:purchaseOrders"/>
  </route>
  <route>
    <!-- lets send invoices from the black box to a different ActiveMQ Queue -->
    <from uri="accounts:invoice"/>
    <to uri="activemq:UK.Accounts.Invoices"/>
  </route>
</camelContext>
```

Naming endpoints

A context component instance can have many public input and output endpoints that can be accessed from outside it's CamelContext. When there are many it is recommended that you use logical names for them to hide the middleware as shown above.

However when there is only one input, output or error/dead letter endpoint in a component we recommend using the common posix shell names **in**, **out** and **err**