ESB Samples Setup

At this page you can find step-by-step instructions on how to install the pre-requisites, and set up the samples - generic instructions on how to start the sample server, deploy a service, and run the client.

Prerequisites | Understanding the Samples | Using the Sample Clients | Starting the Sample Services | Starting Sample ESB Configurations | Setting up the JMS Listener | Setting up Mail Transport Sender | Setting up Mail Transport Receiver | Configuring WSO2 ESB for the FIX Transport | Setting up the FIX Transport | Configure WSO2 ESB for AMQP Transport | Configure WSO2 ESB for TCP Transport | Configure WSO2 ESB for UDP Transport | Configuring the ESB for Script Mediator Support | Setting up Derby database server | Setting up Synapse DataSources

Prerequisites

You need a Java Development Kit / JRE version 1.6.x or later and Apache Ant 1.7.0 or later, at a minimum, to try out the samples. Apache Ant can be downloaded from here: http://ant.apache.org. The JMS examples can be executed against an ActiveMQ installation by default (or another JMS provider with configuration) and any HTTPS examples would require a JDK version 1.6 or later.

Once you setup ant go to the `bin` directory of the distribution and run the `build.xml` file which resides in that directory using Apache Ant. You can do this by typing "ant" without quotes on a console in this directory.

Note

The samples and the documentation assumes that you are running the ESB in DEBUG mode. You can switch from the default INFO log messages to DEBUG log messages by changing the line `log4j.category.org.apache.synapse=INFO` as `log4j.category.org.apache.synapse=DEBUG` in the `lib/log4j.properties` file.

Understanding the Samples

Client	WSO2 ESB	Service
ant stockquote	./wso2esb-samples.sh -sn <n>	SimpleStockQuoteService
		SecureStockQuoteService etc.

The above table depicts the interactions between the clients, the ESB and services at a high level. The clients are able to send SOAP/REST or POX messages over transports such as HTTP/HTTPS or JMS with WS-Addressing, WS-Security, or WS-Reliable messaging. They could send binary optimized content using MTOM or SwA or binary or plain text JMS messages. After mediation through the ESB, the requests are passed over to the sample services. The sample clients and services are explained below.

Using the Sample Clients

The sample clients can be executed from the `samples/axis2Client` directory through the provided Ant script. Simply executing `ant` displays the available clients and some of the sample options used to configure them. The sample clients available are listed below:

1. Stock Quote Client

This is a simple SOAP client that could send stock quote requests, and receive and display the last sale price for a stock symbol.

```
ant stockquote [\-Dsymbol=IBM|MSFT|SUN|..]
[\-Dmode=quote |customquote|fullquote | placeorder | marketactivity]
[&#45;Daddurl=http://localhost:9000/services/SimpleStockQuoteService&#124;\||]
[&#45;Dtrpurl=http://localhost:8280/&#124;\||]
[&#45;Dprxurl=http://localhost:8280/&#124;\||]
[&#45;Dpolicy=../../repository/samples/resources/policy/policy_1.xml&#124;\||]
```

The client is able to operate in the following modes, and send the payloads listed below as SOAP messages:

- **quote -** Sends a quote request for a single stock as follows. The response contains the last sales price for the stock which would be displayed.

```
<m:getQuote xmlns:m="http://services.samples/xsd">
<m:request>
<m:symbol>IBM</m:symbol>
</m:request>
</m:getQuote>
```

- **customquote -** Sends a quote request in a custom format. The ESB would transform this custom request into the standard stock quote request format and send it to the service. Upon receipt of the response, it would be transformed again to a custom response format and returned to the client, which will then display the last sales price.

```
<m0:checkPriceRequest xmlns:m0="http://services.samples/xsd">
<m0:Code>symbol</m0:Code>
</m0:checkPriceRequest>
```

- **fullquote -** Gets quote reports for the stock over a number of days (for example, last 100 days of the year).

```
<m:getFullQuote xmlns:m="http://services.samples/xsd">
<m:request>
<m:symbol>IBM</m:symbol>
</m:request>
</m:getFullQuote>
```

- **placeorder** - Places an order for stocks using a one way request.

```
<m:placeOrder xmlns:m="http://services.samples/xsd">
<m:order>
<m:price>3.141593E0</m:price>
<m:quantity>4</m:quantity>
<m:symbol>IBM</m:symbol>
</m:order>
</m:placeOrder>
```

- **marketactivity** - Gets a market activity report for the day (for example, quotes for multiple symbols).

```
<m:getMarketActivity xmlns:m="http://services.samples/xsd">
<m:request>
<m:symbol>IBM</m:symbol>
...
<m:symbol>MSFT</m:symbol>
</m:request>
</m:getMarketActivity>
```

Smart Client Mode:

The `addurl` property sets the WS-Addressing EPR, and the `trpurl` sets a transport URL for a message. Thus by specifying both properties, the client can operate in the `smart client` mode, where the addressing EPR could specify the ultimate receiver, while the transport URL set to the ESB ensures that any necessary mediation takes place before the message is delivered to the ultimate receiver. For example:

```
ant stockquote -Daddurl=<addressingEPR> -Dtrpurl=<esb>
```

Gateway / Dumb Client Mode:

By specifying only a transport URL, the client operates in the `dumb client` mode, where it sends the message to the ESB and depends on the ESB rules for proper mediation and routing of the message to the ultimate destination. For example:

```
ant stockquote -Dtrpurl=<esb>
```

Proxy Client Mode:

In this mode, the client uses the `prxurl` as an HTTP proxy to send the request. Thus by setting the `prxurl` to the ESB, the client could ensure that the message would reach the ESB for mediation. The client could optionally set a WS-Addressing EPR if required. For example:

```
ant stockquote -Dprxurl=<esb> [-Daddurl=<addressingEPR>]
```

Specifying a Policy

By specifying a WS-Policy using the `policy` property, QoS aspects such as WS-Security could be enforced on the request. The policy specifies details such as timestamps, signatures, and encryption.

2. Generic JMS Client

The JMS client is able to send plain text, plain binary content, or POX content by directly publishing a JMS message to the specified destination. The JMS destination name should be specified with the `jms_dest` property. The `jms_type` property could specify text, binary or pox to specify the type of message payload.

The plain text payload for a text message can be specified through the `payload` property. For binary messages, the `payload` property would contain the path to the binary file. For POX messages, the `payload` property will hold a stock symbol name to be used within the POX request for stock order placement request. For example:

```
ant jmsclient -Djms_type=text -Djms_dest=dynamicQueues/JMSTextProxy
-Djms_payload="24.34 100 IBM"
ant jmsclient -Djms_type=pox -Djms_dest=dynamicQueues/JMSPoxProxy -Djms_payload=MSFT
ant jmsclient -Djms_type=binary -Djms_dest=dynamicQueues/JMSFileUploadProxy

-Djms_payload=./../../repository/samples/resources/mtom/asf-logo.gif
```

Note

The JMS client assumes the existence of a default ActiveMQ (5.2.0) installation on the local machine.

3. MTOM / SwA Client

The MTOM / SwA client can send a binary image file as a MTOM or SwA optimized message, and receive the same file again through the response and save it as a temporary file. The `opt_mode` can specify `mtom` or `swa` respectively for the above mentioned optimizations. Optionally, the path to a custom file can be specified through the `opt_file` property, and the destination address can be changed through the `opt_url` property if required. For example:

```
ant optimizeclient -Dopt_mode=[mtom | swa]
```

Starting the Sample Services

The sample services ship with a pre-configured Axis2 server, and demonstrates in-only and in-out SOAP/REST or POX messaging over HTTP/HTTPS and JMS transports using WS-Addressing, WS-Security, and WS-Reliable Messaging. It also handles binary content using MTOM and SwA.

The sample services can be found in the `samples/axis2Server/src directory` and can be built and deployed using Ant from each service directory.

```
user@host:/tmp/wso2esb-2.0/samples/axis2Server/src/SimpleStockQuoteService$ ant
Buildfile: build.xml
 ...
build-service:
   ....
      [jar] Building jar:
/tmp/wso2esb-2.0/samples/axis2Server/repository/services/SimpleStockQuoteService.aar

BUILD SUCCESSFUL
Total time: 3 seconds
```

To start the Axis2 server, go to the `samples/axis2Server` directory and execute the `axis2server.sh` or `axis2server.bat` script. This starts the Axis2 server with the HTTP transport listener on port 9000 and HTTPS on 9002 respectively. To enable JMS transport, you need to set up and start a JMS provider. An ActiveMQ 5.2.0 JMS server on the local machine is supported by default, and can be easily enabled by uncommenting the JMS transport from the `repository/conf/axis2.xml`.

The Sample services are as follows:

1. `SimpleStockQuoteService`

This service has four operations:

- `getQuote` (in-out) - Generates a sample stock quote for a given symbol.
- `getFullQuote` (in-out) - Generates a history of stock quotes for the symbol for a number of days.
- `getMarketActivity` (in-out) - Returns stock quotes for a list of given symbols.
- `placeOrder` (in-only) - Accepts a one way message for an order.

2. `SecureStockQuoteService`

This service is a clone of the `SimpleStockQuoteService`, but has WS-Security enabled and an attached security policy for signing and encrypting messages.

3. `MTOMSwASampleService`

This service has three operations:

- `uploadFileUsingMTOM` (in-out) - Accepts a binary image from the SOAP request as MTOM, and returns this image back again as the response.
- `uploadFileUsingSwA` (in-out) - Accepts a binary image from the SOAP request as SwA, and returns this image back again as the response.
- `oneWayUploadUsingMTOM` (in-only) - Saves the request message to the disk.

This service also demonstrates the use of MTOM and SwA.

Starting Sample ESB Configurations

To start the WSO2 ESB with the sample default configuration, execute the `wso2server.bat` or `wso2server.sh` script found in the `/bin` directory. This starts up an instance of the ESB using the Synapse and Axis2 configuration files located in the `conf` directory. The `repository/samples` directory contains the sample configurations available as `synapse_sample_<n>.xml` files. To start a specific sample configuration of the ESB, use the `wso2esb-samples.sh` or `wso2esb-samples.bat` as follows:

```
wso2esb-samples.bat -sn <n>
 ./wso2esb-samples.sh -sn <n>
```

Setting up the JMS Listener

The samples used in this guide assumes the existence of a local ActiveMQ 5.1.0 or higher (http://activemq.apache.org) installation, which is properly installed and started.

To enable the JMS transport, you need to uncomment the JMS transport listener configuration. If you are using a JMS provider other than ActiveMQ, this configuration should be updated to reflect your environment. Once uncommented, the default configuration should be as follows. To enable JMS for the ESB, the `repository/conf/axis2.xml` must be updated. To enable JMS support for the sample Axis2 server, the `samples/axis2Server/repository/conf/axis2.xml` file must be updated.

```
<!--Uncomment this and configure as appropriate for JMS transport support, after
setting up your JMS environment (e.g. ActiveMQ)-->
    <transportReceiver name="jms"
class="org.apache.axis2.transport.jms.JMSListener">
        <parameter name="myTopicConnectionFactory" locked="false">
                <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
                <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
                <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">TopicConnectionFactory</parameter>
        </parameter>

        <parameter name="myQueueConnectionFactory" locked="false">
                <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
                <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
                <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
        </parameter>

        <parameter name="default" locked="false">
                <parameter name="java.naming.factory.initial"
locked="false">org.apache.activemq.jndi.ActiveMQInitialContextFactory</parameter>
                <parameter name="java.naming.provider.url"
locked="false">tcp://localhost:61616</parameter>
                <parameter name="transport.jms.ConnectionFactoryJNDIName"
locked="false">QueueConnectionFactory</parameter>
        </parameter>
    </transportReceiver>
```

Setting up Mail Transport Sender

To enable the mail transport sender for samples, you need to uncomment the mail transport sender configuration in the `repository/conf/axis2.xml`. Uncomment the mail transport sender sample configuration and make sure it points to a valid SMTP configuration for any actual scenarios.

```
<transportSender name="mailto"
class="org.apache.synapse.transport.mail.MailTransportSender">
        <parameter name="mail.smtp.host">smtp.gmail.com</parameter>
        <parameter name="mail.smtp.port">587</parameter>
        <parameter name="mail.smtp.starttls.enable">true</parameter>
        <parameter name="mail.smtp.auth">true</parameter>
        <parameter name="mail.smtp.user">synapse.demo.0</parameter>
        <parameter name="mail.smtp.password">mailpassword</parameter>
        <parameter name="mail.smtp.from">synapse.demo.0@gmail.com</parameter>
    </transportSender>
```

Setting up Mail Transport Receiver

To enable the mail transport receiver for samples, you need to uncomment the mail transport receiver configuration in the `repository/conf/axis2.xml`. Uncomment the mail transport receiver sample configuration.

> **Note**
>
> You need to provide correct parameters for a valid mail account at service level.

```
<transportReceiver name="mailto"
class="org.apache.axis2.transport.mail.MailTransportListener">
</transportReceiver>
```

Configuring WSO2 ESB for the FIX Transport

Setting up the FIX Transport

To run the FIX samples used in this guide you need a local Quickfix/J (http://www.quickfixj.org) installation. Download Quickfix/J from: http://www.quickfixj.org/downloads.

Simply uncomment the FIX transport sender and FIX transport receiver configurations in the `repository/conf/axis2.xml`. Alternatively, if the FIX transport management bundle is in use you can enable the FIX transport listener and the sender from the WSO2 ESB management console. Login to the console and navigate to "Transports" on management menu. Scroll down to locate the sections related to the FIX transport. Simply click on the "Enable" links to enable the FIX listener and the sender.

Configuring the ESB for FIX Samples

In order to configure WSO2 ESB to run the FIX samples given in this guide, you need to create some FIX configuration files as specified below.

> **Note**
>
> You can find the config files in `$ESB_HOME/repository/conf/sample/resources/fix` folder.

The `FileStorePath property` in the following two files should point to two directories in your local file system. Once the samples are executed, Synapse will create FIX message stores in these two directories.

Put the following entries in a file called `fix-synapse.cfg`.

```
[default]
FileStorePath=repository/logs/fix/data
ConnectionType=acceptor
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ValidOrderTypes=1,2,F
SenderCompID=SYNAPSE
TargetCompID=BANZAI
UseDataDictionary=Y
DefaultMarketPrice=12.30

[session]
BeginString=FIX.4.0
SocketAcceptPort=9876
```

Put the following entries in a file called `synapse-sender.cfg`.

```
[default]
FileStorePath=repository/logs/fix/data
SocketConnectHost=localhost
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ReconnectInterval=5
SenderCompID=SYNAPSE
TargetCompID=EXEC
ConnectionType=initiator

 [session]
BeginString=FIX.4.0
SocketConnectPort=19876
```

Configuring Sample FIX Applications

If you use a binary distribution of Quickfix/J, the two samples and their configuration files are all packed to a single JAR file called `quickfixj-examples.jar`. You have to extract the JAR file, modify the configuration files and pack them to a JAR file again under the same name.

You can pass the new configuration file as a command line parameter too, in that case you do not need to modify the `quickfixj-examples.jar`. You can copy the `config` files from `$ESB_HOME/repository/conf/sample/resources/fix` folder to `$QFJ_HOME/etc` folder. Execute the sample apps from `$QFJ_HOME/bin`, `./banzai.sh/bat ../etc/banzai.cfg executor.sh/bat ../etc/executor.sh`.

Locate and edit the FIX configuration file of Executor to be as follows. This file is usually named `executor.cfg`.

```
[default]
FileStorePath=examples/target/data/executor
ConnectionType=acceptor
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ValidOrderTypes=1,2,F
SenderCompID=EXEC
TargetCompID=SYNAPSE
UseDataDictionary=Y
DefaultMarketPrice=12.30

[session]
BeginString=FIX.4.0
SocketAcceptPort=19876
```

Locate and edit the FIX configuration file of Banzai to be as follows. This file is usually named `banzai.cfg`.

```
[default]
FileStorePath=examples/target/data/banzai
ConnectionType=initiator
SenderCompID=BANZAI
TargetCompID=SYNAPSE
SocketConnectHost=localhost
StartTime=00:00:00
EndTime=00:00:00
HeartBtInt=30
ReconnectInterval=5

[session]
BeginString=FIX.4.0
SocketConnectPort=9876
```

The `FileStorePath` property in the above two files should point to two directories in your local file system. The launcher scripts for the sample application can be found in the bin directory of Quickfix/J distribution.

For more information regarding the FIX sample applications, please refer the Example Applications (http://www.quickfixj.org/quickfixj/usermanual/1.5.0/usage/examples.html) section in the Quickfix/J documentation. For more information on configuring Quickfix/J applications refer the Configuring Quickfix/J (http://www.quickfixj.org/quickfixj/usermanual/1.5.0/usage/configuration.html) section of the Quickfix/J documentation.

Configure WSO2 ESB for AMQP Transport

The samples used in this guide assumes the existence of a local QPid (1.0-M2 or higher) installation properly installed and started up. You also need to copy the following client JAR files into the `repository/components/lib` directory to support AMQP. These files are found in the `lib` directory of the QPid installation.

- qpid-client-1.0-incubating-M2.jar
- qpid-common-1.0-incubating-M2.jar
- geronimo-jms_1.1_spec-1.0.jar

- slf4j-api-1.4.0.jar **
- slf4j-log4j12-1.4.0.jar **

Note

To configure FIX (Quickfix/J 1.3) with AMQP (QPid-1.0-M2) copy the `sl4j-*` libraries that come with QPid and ignore the `sl4j-*` libraries that come with Quickfix/J.

To enable the AMQP over JMS transport, you need to uncomment the JMS transport listener configuration. To enable AMQP over JMS for ESB, the `repository/conf/axis2.xml` must be updated, while to enable JMS support for the sample Axis2 server the `samples/axis2Server/repository/conf/axis2.xml` file must be updated.

```
<\!--Uncomment this and configure as appropriate for JMS transport support, after
setting up your JMS environment \-->
<transportReceiver name="jms">
</transportReceiver>

<transportSender name="jms">
</transportReceiver>
```

Locate and edit the AMQP connection settings file for the message consumer, this file is usually named `direct.properties` and can be found in the `repository/samples/resources/fix` directory.

```
java.naming.factory.initial =
org.apache.qpid.jndi.PropertiesFileInitialContextFactory
# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.qpidConnectionfactory =
amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672'
# Register an AMQP destination in JNDI
# destination.[jniName] = [BindingURL]
destination.directQueue =
direct://amq.direct//QpidStockQuoteService?routingkey='QpidStockQuoteService'
destination.replyQueue = direct://amq.direct//replyQueue?routingkey='replyQueue'
```

Locate and edit the AMQP connection settings file for WSO2 ESB, this file is usually named `con.properties` and can be found in the `repository/samples/resources/fix` directory.

```
#initial context factory
\#java.naming.factory.initial
=org.apache.qpid.jndi.PropertiesFileInitialContextFactory
# register some connection factories
# connectionfactory.[jndiname] = [ConnectionURL]
connectionfactory.qpidConnectionfactory=amqp://guest:guest@clientid/test?brokerlist=
'tcp://localhost:5672'
# Register an AMQP destination in JNDI
# destination.[jndiName] = [BindingURL]
destination.directQueue=direct://amq.direct//QpidStockQuoteService
```

Configure WSO2 ESB for TCP Transport

To enable the TCP transport for samples, simply open up the `repository/conf/axis2.xml` file in a text editor and add the following transport receiver configuration and sender configuration. TCP transport module is shipped with ESB by default.

```
<transportReceiver name="tcp"
class="org.apache.axis2.transport.tcp.TCPTransportListener">
    <parameter name="transport.tcp.port">6060</parameter>
</transportReceiver>

<transportSender name="tcp"
class="org.apache.axis2.transport.tcp.TCPTransportSender"/>
```

If you wish to use the sample Axis2 client to send TCP messages, you have to uncomment the TCP transport sender configuration in the `samples/axis2Client/client_repo/conf/axis2.xml` file.

Configure WSO2 ESB for UDP Transport

To enable the UDP transport for samples, open the file `repository/conf/axis2.xml` in a text editor and add the following transport configurations. UDP transport component is shipped with WSO2 ESB by default.

```
<transportReceiver name="udp" class="org.apache.axis2.transport.udp.UDPListener"/>
<transportSender name="udp" class="org.apache.axis2.transport.udp.UDPSender"/>
```

If you wish to use the sample Axis2 client to send UDP messages, you have to uncomment the UDP transport sender configuration in the `samples/axis2Client/client_repo/conf/axis2.xml` file.

Configuring the ESB for Script Mediator Support

The Script Mediator is a Synapse extension, and thus all pre-requisites for all BSF supported scripting languages may not be bundled by default with the ESB distribution. Before you use some script mediators, you may need to manually add the required JAR files to the `lib` directory (`lib/extensions`), and optionally perform other installation tasks as may be required by the individual scripting language. This is detailed in the following sections.

JavaScript Support

The JavaScript/E4X support is enabled by default and comes ready-to-use with the WSO2 ESB distribution.

Ruby Support

For Ruby support you need to install the "WSO2 Carbon - JRuby Engine for Mediation" feature. This feature is available in the WSO2 P2 repository. Use the WSO2 Carbon Component Manager UI in the management console to connect to the WSO2 P2 repository and install the above feature. This feature will install a JRuby 1.3 engine in the ESB runtime.

Alternatively, you can also download and install the JRuby engine manually. Download the file named `jruby-complete-1.3.0.wso2v1.jar` from the WSO2 P2 repository and copy it into `repository/components/dropins` directory.

Setting up Derby database server

1. Download Apache Derby distribution from http://db.apache.org/derby.
2. Set up and start the Derby network server.
3. Create and open a connection to the database using the Derby client driver.

```
CONNECT 'jdbc:derby://localhost:1527/esbdb;user=esb;password=esb;create=true';
```

4. Create a table using the following statement.

```
CREATE table company(name varchar(10), id varchar(10), price double);
```

5. Inserts some data using following statements.

```
INSERT into company values ('IBM','c1',0.0);
INSERT into company values ('SUN','c2',0.0);
INSERT into company values ('MSFT','c3',0.0);
```

When using Derby, you need to add `derby.jar`, `derbyclient.jar` and `derbynet.jar` to the classpath. This can be done by putting the above three JARs into the ESB `lib/extensions` directory. For testing these samples Derby10.3.2.1 binary distribution was used.

You can use any other database product instead of Derby. Then you have to change the database connection details accordingly. Also you have to copy the required database driver JARs to the ESB classpath.

Setting up Synapse DataSources

Definition of the reusable database connection pool or datasources can be done using `datasources.properties` file. It is possible to configure any number of datasources. Currently this only supports two types of datasources and those are based on Apache DBCP datasources. Those types are `BasicDataSource` and `PerUserPoolDataSource` (based on Apache DBCP). The following configuration includes both definition.

Configuration is similar to the `log4j` appender configuration.

It requires two databases, follow the above steps to create the two databases `jdbc:derby://localhost:1527/lookupdb`, `jdbc:derby://localhost:1527/reportdb` using the user name and password as `esb`. Fill in the data for those two databases as per described in the above section

`datasources.properties configuration`

```
\################################################################################
# DataSources Configuration
\################################################################################
synapse.datasources=lookupds,reportds
synapse.datasources.icFactory=com.sun.jndi.rmi.registry.RegistryContextFactory
synapse.datasources.providerPort=2199
# If following property is present , then assumes that there is an external JNDI
provider and will not start a RMI registry
\#synapse.datasources.providerUrl=rmi://localhost:2199

synapse.datasources.lookupds.registry=Memory
synapse.datasources.lookupds.type=BasicDataSource
synapse.datasources.lookupds.driverClassName=org.apache.derby.jdbc.ClientDriver
synapse.datasources.lookupds.url=jdbc:derby://localhost:1527/lookupdb;create=false
# Optionally you can specifiy a specific password provider implementation which
overrides any globally configured provider
synapse.datasources.lookupds.secretProvider=org.apache.synapse.commons.security.secr
et.handler.SharedSecretCallbackHandler
synapse.datasources.lookupds.username=esb
# Depending on the password provider used, you may have to use an encrypted password
here\!
synapse.datasources.lookupds.password=esb
synapse.datasources.lookupds.dsName=lookupdb
synapse.datasources.lookupds.maxActive=100
synapse.datasources.lookupds.maxIdle=20
synapse.datasources.lookupds.maxWait=10000

synapse.datasources.reportds.registry=JNDI
synapse.datasources.reportds.type=PerUserPoolDataSource
synapse.datasources.reportds.cpdsadapter.factory=org.apache.commons.dbcp.cpdsadapter
.DriverAdapterCPDS
synapse.datasources.reportds.cpdsadapter.className=org.apache.commons.dbcp.cpdsadapt
er.DriverAdapterCPDS
synapse.datasources.reportds.cpdsadapter.name=cpds
synapse.datasources.reportds.dsName=reportdb
synapse.datasources.reportds.driverClassName=org.apache.derby.jdbc.ClientDriver
synapse.datasources.reportds.url=jdbc:derby://localhost:1527/reportdb;create=false
# Optionally you can specifiy a specific password provider implementation which
overrides any globally configured provider
synapse.datasources.reportds.secretProvider=org.apache.synapse.commons.security.secr
et.handler.SharedSecretCallbackHandler
synapse.datasources.reportds.username=esb
# Depending on the password provider used, you may have to use an encrypted password
here\!
synapse.datasources.reportds.password=esb
synapse.datasources.reportds.maxActive=100
synapse.datasources.reportds.maxIdle=20
synapse.datasources.reportds.maxWait=10000
```

To secure data sources password, you should use the mechanism for `securing secret Information`. If that mechanism is used, then passwords that have been specified are considered as aliases and those are used for picking actual passwords. To get password securely, you should set the password provider for each data source. The password provider should be an implementation of the following:

- `org.apache.synapse.commons.security.secret.SecretCallbackHandler`.

For run this sample, you just need to uncomment `secret-conf.properties`, `cipher-text.properties` and `datasources.properties`. Those files are in conf directory.