



MuleSoft™

## Mule 3 User Guide

1. Home .....	6
1.1 Essentials of Using Mule ESB 3 .....	6
1.1.1 Understanding Mule Configuration .....	7
1.1.1.1 About the XML Configuration File .....	23
1.1.2 Choosing Between Flows, Patterns, or Services .....	24
1.1.2.1 Using Flows for Service Orchestration .....	25
1.1.2.2 Using Mule Services .....	29
1.1.2.2.1 Service Messaging Styles .....	29
1.1.2.2.2 Configuring the Service .....	34
1.1.2.2.3 Models .....	37
1.1.2.2.4 Using Message Routers .....	38
1.1.2.3 Using Mule Configuration Patterns .....	61
1.1.2.3.1 Pattern-Based Configuration .....	62
1.1.2.3.2 Simple Service Pattern .....	62
1.1.2.3.3 Bridge Pattern .....	65
1.1.2.3.4 Validator Pattern .....	67
1.1.2.3.5 Web Service Proxy Pattern .....	69
1.1.3 Message Sources and Message Processors .....	71
1.1.3.1 Routing Message Processors .....	74
1.1.3.2 Custom Message Processors .....	83
1.1.3.3 Message Enricher .....	84
1.1.3.4 Logger Element for Flows .....	86
1.1.4 Configuring Components .....	87
1.1.4.1 Configuring Java Components .....	89
1.1.4.2 Developing Components .....	91
1.1.4.2.1 Entry Point Resolver Configuration Reference .....	94
1.1.4.3 Component Bindings .....	98
1.1.4.4 Using Interceptors .....	101
1.1.5 Connecting Using Transports .....	104
1.1.5.1 Configuring a Transport .....	105
1.1.6 Configuring Endpoints .....	108
1.1.6.1 Mule Endpoint URLs .....	115
1.1.7 Using Filters .....	116
1.1.8 Using Transformers .....	120
1.1.8.1 Transformers Configuration Reference .....	124
1.1.8.2 Native Support for JSON .....	136
1.1.8.3 XmlPrettyPrinter Transformer .....	139
1.1.8.4 Creating Custom Transformers .....	140
1.1.8.4.1 Creating Service Objects and Transformers Using Annotations .....	140
1.1.8.4.2 Creating Custom Transformer Class .....	152
1.1.9 Connecting SaaS, Social Media, and E-Commerce Using Mule Cloud Connect .....	156
1.1.9.1 Integrating with Cloud Connect .....	157
1.1.9.2 Available Cloud Connectors .....	160
1.1.9.2.1 Authorize.Net .....	161
1.1.9.2.2 Cybersource .....	166
1.1.9.2.3 Flickr .....	172
1.1.9.2.4 Magento .....	177
1.1.9.2.5 SalesForce .....	204
1.1.9.2.6 Twitter .....	215
1.1.9.3 Getting Started with Cloud Connect .....	220
1.1.9.4 How to Build a Cloud Connector .....	221
1.1.9.4.1 Creating a Cloud Connector .....	222
1.1.9.4.2 Connector Annotations .....	226
1.1.9.4.3 Using Eclipse .....	228
1.1.9.4.4 Using IntelliJ .....	231
1.1.9.4.5 Testing Your Connector .....	237
1.1.9.4.6 Documenting Your Connector .....	239
1.1.9.4.7 Integration with Mule .....	240
1.1.9.4.8 Submitting a Cloud Connector .....	241
1.1.9.4.9 Recipes .....	241
1.1.10 Mule Query Language .....	242
1.1.10.1 MQL Download .....	244
1.1.10.2 MQL Enrich Data .....	244
1.1.10.3 MQL Merge Datasets .....	246
1.1.10.4 MQL Mule Integration .....	248
1.1.10.5 MQL Query Java Objects .....	250
1.1.10.6 MQL Reference Guide .....	251
1.1.10.7 MQL Roadmap .....	253
1.1.10.8 MQL Service Versioning .....	254
1.1.10.9 MQL Spring Integration .....	256
1.1.11 Using Expressions .....	257
1.1.11.1 Creating Expression Evaluators .....	260
1.1.11.2 Mule Expression Language .....	262
1.1.12 Message Property Scopes .....	263
1.1.13 Transaction Management .....	265

1.1.13.1 Shared Transactions .....	269
1.1.14 Configuring Security .....	270
1.1.14.1 Configuring the Spring Security Manager .....	271
1.1.14.2 Configuring the Acegi Security Manager .....	272
1.1.14.3 Component Authorization Using Spring Security .....	274
1.1.14.4 Component Authorization Using Acegi .....	276
1.1.14.5 Setting up LDAP Provider for Spring Security .....	278
1.1.14.6 Setting up LDAP Provider for Acegi .....	280
1.1.14.7 Upgrading from Acegi to Spring Security .....	282
1.1.14.8 Encryption Strategies .....	285
1.1.14.9 PGP Security .....	285
1.1.14.10 Jaas Security .....	289
1.1.14.11 SAML Module .....	291
1.1.15 Error Handling .....	293
1.1.16 Using Web Services .....	295
1.1.16.1 Proxying Web Services .....	296
1.1.16.2 Using .NET Web Services with Mule .....	298
1.1.16.3 Web Service Wrapper .....	299
1.2 Advanced Usage of Mule ESB 3 .....	300
1.2.1 Object Scopes .....	301
1.2.2 Storing Objects in the Registry .....	302
1.2.3 Using Mule with Spring .....	303
1.2.3.1 Sending and Receiving Mule Events in Spring .....	303
1.2.3.2 Spring Application Contexts .....	305
1.2.3.3 Using Spring Beans as Service Components .....	306
1.2.4 Configuring Properties .....	308
1.2.5 Mule Agents .....	310
1.2.5.1 JMX Management .....	311
1.2.6 About Configuration Builders .....	317
1.2.7 Tuning Performance .....	317
1.2.8 Configuring Reconnection Strategies .....	327
1.2.9 Streaming .....	331
1.2.10 Bootstrapping the Registry .....	332
1.2.11 Configuring Queues .....	333
1.2.12 Internationalizing Strings .....	334
1.2.13 Using the Mule Client .....	336
1.2.14 Mule Object Stores .....	342
1.2.15 Flow Processing Strategies .....	343
1.3 Extending Mule ESB 3 .....	346
1.3.1 Extending Components .....	347
1.3.2 Creating Example Archetypes .....	348
1.3.3 Creating a Custom XML Namespace .....	351
1.3.4 Creating Module Archetypes .....	358
1.3.5 Creating Catalog Archetypes .....	361
1.3.6 Creating Project Archetypes .....	364
1.3.7 Creating Transports .....	368
1.3.7.1 Transport Archetype .....	373
1.3.7.2 Transport Service Descriptors .....	378
1.3.8 Creating Custom Routers .....	379
1.4 Deploying Mule ESB 3 .....	380
1.4.1 Deployment Scenarios .....	381
1.4.1.1 Choosing the Right Topology .....	381
1.4.1.2 Embedding Mule in a Java Application or Webapp .....	384
1.4.1.3 Deploying Mule to JBoss .....	386
1.4.1.3.1 Mule as MBean .....	386
1.4.1.4 Deploying Mule to WebLogic .....	389
1.4.1.5 Deploying Mule to WebSphere .....	391
1.4.1.6 Deploying Mule as a Service to Tomcat .....	393
1.4.1.7 Application Server Based Hot Deployment .....	393
1.4.1.8 Classloader Control in Mule .....	395
1.4.1.2 Mule Deployment Model .....	398
1.4.2.1 Hot Deployment .....	399
1.4.2.2 Application Deployment .....	399
1.4.2.3 Application Format .....	401
1.4.2.4 Deployment Descriptor .....	401
1.4.3 Configuring Logging .....	402
1.4.4 Mule Server Notifications .....	403
1.4.5 Profiling Mule .....	407
1.4.6 Hardening your Mule Installation .....	408
1.4.7 Mule High Availability .....	409
1.4.8 Configuring Mule for Different Deployment Scenarios .....	413
1.4.8.1 Configuring Mule as a Linux or Unix Daemon .....	414
1.4.8.2 Configuring Mule as a Windows Service .....	414
1.4.8.3 Configuring Mule to Run From a Script .....	414
1.4.8.4 Configuring Mule to Run From Maven .....	415

1.5 Testing With Mule ESB 3 .....	415
1.5.1 Introduction to Testing Mule .....	415
1.5.2 Using IDEs .....	416
1.5.3 Unit Testing .....	416
1.5.4 Functional Testing .....	417
1.5.5 Using Dynamic Ports in Mule Test Cases .....	424
1.5.6 Testing Strategies .....	424
1.6 Troubleshooting .....	432
1.6.1 Configuring Mule Stacktraces .....	433
1.6.2 Logging .....	433
1.6.2.1 Logging With Mule ESB 3.x .....	434
1.6.3 Step Debugging .....	434
1.7 Team Development with Mule .....	435
1.7.1 Modularizing Your Configuration Files for Team Development .....	435
1.7.2 Using Side-by-Side Configuration Files .....	436
1.7.3 Using Parameters in Your Configuration Files .....	437
1.7.4 Using Modules In Your Application .....	438
1.7.5 Sharing Custom Code .....	438
1.7.6 Sharing Custom Configuration Fragments .....	439
1.7.7 Sharing Custom Configuration Patterns .....	440
1.7.8 Sharing Applications .....	440
1.8 Sustainable Software Development Practices with Mule .....	440
1.8.1 Reproducible Builds .....	440
1.8.2 Continuous Integration .....	441
1.8.3 Repeatable Deploys .....	441
1.9 Reference Materials for Mule ESB 3 .....	441
1.9.1 Configuration Reference .....	442
1.9.1.1 Asynchronous Reply Router Configuration Reference .....	442
1.9.1.2 Catch-all Strategy Configuration Reference .....	443
1.9.1.3 Component Configuration Reference .....	444
1.9.1.4 Endpoint Configuration Reference .....	450
1.9.1.5 Exception Strategy Configuration Reference .....	453
1.9.1.6 Filters Configuration Reference .....	455
1.9.1.7 Global Settings Configuration Reference .....	461
1.9.1.8 Inbound Router Configuration Reference .....	461
1.9.1.9 Model Configuration Reference .....	465
1.9.1.10 Notifications Configuration Reference .....	466
1.9.1.11 Outbound Router Configuration Reference .....	468
1.9.1.12 Properties Configuration Reference .....	478
1.9.1.13 Security Manager Configuration Reference .....	480
1.9.1.14 Service Configuration Reference .....	482
1.9.1.15 Transactions Configuration Reference .....	485
1.9.1.16 BPM Configuration Reference .....	490
1.9.2 Transports Reference .....	491
1.9.2.1 AJAX Transport Reference .....	495
1.9.2.2 EJB Transport Reference .....	508
1.9.2.3 Email Transport Reference .....	510
1.9.2.3.1 Email Transport Filters .....	510
1.9.2.3.2 Email Transport Limitations .....	511
1.9.2.3.3 Email Transport Transformers .....	511
1.9.2.3.4 SMTP Transport Reference .....	512
1.9.2.4 File Transport Reference .....	522
1.9.2.5 FTP Transport Reference .....	536
1.9.2.6 HTTPS Transport Reference .....	547
1.9.2.7 HTTP Transport Reference .....	549
1.9.2.8 IMAP Transport Reference .....	562
1.9.2.9 JDBC Transport Reference .....	573
1.9.2.9.1 JDBC Transport Configuration Reference .....	599
1.9.2.9.2 JDBC Transport Performance Benchmark Results .....	603
1.9.2.10 Jetty Transport Reference .....	604
1.9.2.10.1 Jetty SSL Transport .....	605
1.9.2.11 JMS Transport Reference .....	606
1.9.2.11.1 Open MQ Integration .....	620
1.9.2.11.2 Fiorano Integration .....	622
1.9.2.11.3 JBoss Jms Integration .....	623
1.9.2.11.4 SeeBeyond JMS Server Integration .....	626
1.9.2.11.5 Sun JMS Grid Integration .....	626
1.9.2.11.6 Tibco EMS Integration .....	627
1.9.2.11.7 SonicMQ Integration .....	628
1.9.2.11.8 OpenJms Integration .....	629
1.9.2.11.9 HornetQ Integration .....	629
1.9.2.11.10 WebLogic JMS Integration .....	631
1.9.2.11.11 SwiftMQ Integration .....	634
1.9.2.11.12 ActiveMQ Integration .....	636
1.9.2.11.13 MuleMQ Integration .....	643

1.9.2.12 Mule MSMQ Transport Reference .....	652
1.9.2.13 Mule WMQ Transport Reference .....	658
1.9.2.14 Multicast Transport Reference .....	668
1.9.2.15 POP3 Transport Reference .....	674
1.9.2.16 Quartz Transport Reference .....	684
1.9.2.17 RMI Transport Reference .....	692
1.9.2.18 Servlet Transport Reference .....	693
1.9.2.19 SFTP Transport Reference .....	695
1.9.2.20 STUDIO Transport Reference .....	710
1.9.2.21 TCP Transport Reference .....	711
1.9.2.22 UDP Transport Reference .....	729
1.9.2.23 VM Transport Reference .....	734
1.9.2.24 WSDL Connectors .....	744
1.9.2.25 XMPP Transport Reference .....	746
1.9.2.26 TCP, SSL, and TLS Transports Reference	759
1.9.2.26.1 Choosing a Transport .....	760
1.9.2.26.2 Custom TCP Protocol .....	760
1.9.2.26.3 Protocol Tables .....	762
1.9.2.26.4 Protocol Types .....	763
1.9.2.26.5 SSL and TLS Transports Reference .....	763
1.9.2.26.6 TCP and SSL Debugging Notes .....	775
1.9.2.26.7 TCP Connector Attributes .....	775
1.9.2.27 BPM Transport Reference .....	775
1.9.3 Modules Reference .....	778
1.9.3.1 CXF Module Reference .....	779
1.9.3.1.1 CXF Module Configuration Reference .....	780
1.9.3.1.2 CXF Module Overview .....	784
1.9.3.1.3 Building Web Services with CXF .....	784
1.9.3.1.4 Consuming Web Services with CXF .....	788
1.9.3.1.5 Enabling WS-Addressing .....	790
1.9.3.1.6 Enabling WS-ReliableMessaging .....	792
1.9.3.1.7 Enabling WS-Security .....	793
1.9.3.1.8 Proxying Web Services with CXF .....	796
1.9.3.1.9 Supported Web Service Standards .....	797
1.9.3.1.10 Upgrading CXF from Mule 2 .....	798
1.9.3.1.11 Using a Web Service Client Directly .....	798
1.9.3.1.12 Using HTTP GET Requests .....	799
1.9.3.1.13 Using MTOM .....	799
1.9.3.2 Jersey Module Reference .....	801
1.9.3.3 JSON Module Reference .....	803
1.9.3.4 Acegi Module Reference .....	807
1.9.3.5 JAAS Module Reference .....	813
1.9.3.6 JBoss Transaction Manager Reference .....	814
1.9.3.7 Scripting Module Reference .....	815
1.9.3.8 Spring Extras Module Reference .....	819
1.9.3.9 SXC Module Reference .....	819
1.9.3.10 XML Module Reference .....	820
1.9.3.10.1 DomToXml Transformer .....	822
1.9.3.10.2 JAXB Bindings .....	823
1.9.3.10.3 JAXB Transformers .....	825
1.9.3.10.4 JXPath Extractor Transformer .....	826
1.9.3.10.5 XML Namespaces .....	826
1.9.3.10.6 XmlObject Transformers .....	827
1.9.3.10.7 XmItoXMLStreamReader Transformer .....	828
1.9.3.10.8 XPath Extractor Transformer .....	829
1.9.3.10.9 XQuery Support .....	830
1.9.3.10.10 XQuery Transformer .....	833
1.9.3.10.11 XSLT Transformer .....	837
1.9.3.11 Data Bindings Reference .....	842
1.9.3.12 BPM Module Reference .....	842
1.9.3.13 JBoss jBPM Module Reference .....	847
1.9.3.14 Atom Module Reference .....	851
1.9.3.15 Drools Module Reference .....	860
1.9.3.16 ATOM Module .....	864
1.9.3.17 RSS Module Reference .....	870
1.9.4 Expressions Configuration Reference .....	874
1.9.5 Schema Documentation .....	877
1.9.5.1 Notes on Mule 3.0 Schema Changes .....	877
1.9.6 Mule ESB 3 and Test API Javadoc .....	878
1.9.7 Release and Migration Notes .....	878
2. Suggested Reading .....	879
3. Using the Mule RESTpack .....	880

# Home

## Mule ESB 3 User Guide



This User Guide is for Mule ESB version 3.2.0 and 3.1.x. Information specific to Mule 3.2.0 is marked as [Mule 3.2].

The User Guide describes how to use Mule for software development. Some of the documentation on this site is under construction for the current release. If you have any feedback on the content, or would like to sign up to become a site contributor or editor, please [contact us](#).

Download the zipped PDF version of this book [here](#).

## How Mule Works

Have you ever wondered how Mule accomplishes its task as an Enterprise Service Bus? This topic explains how Mule works.

## Using Mule with Web Services

This topic, written by popular demand, provides a detailed overview of how to integrate Web services into your Mule application.

## Using the Mule ESB Management Console

Deploy applications, monitor your Mule ESB servers, get alerts, and perform admin server tasks using the Mule Management Console. Read about all the features of the console and how to use them.

## Connecting SaaS, Social Media, and E-Commerce Using Mule Cloud Connect

Mule Cloud Connect is a powerful way to move data between different endpoints such as Salesforce.com, Twitter, and Amazon. Read here how to accomplish this.

## Creating Transports

If you're ready to extend Mule to create your own transport, read this topic, which explains how to accomplish this advanced task.

## Introduction to Testing Mule

Testing is part and parcel of getting ready to deploy your Mule application. Read this topic for an introduction into testing Mule.

## Team Development with Mule

You may be working on a team that is developing a Mule application. Read this section for tips on how to structure your development effort to support team development.

## Reference

Everything you need to look up important information when developing with Mule.

## Essentials of Using Mule ESB 3

# Essentials of Using Mule ESB 3

Mule is a flexible message processing and integration framework. The way you use it depends on the problem you are trying to solve. Mule 3 offers multiple configuration constructs that can be mixed and matched as required to implement your solution.

- Understanding Mule Configuration
- Choosing Between Flows, Patterns, or Services
- Message Sources and Message Processors
- Configuring Components
- Connecting Using Transports
- Configuring Endpoints
- Using Filters
- Using Transformers
- Connecting SaaS, Social Media, and E-Commerce Using Mule Cloud Connect
- Mule Query Language
- Using Expressions
- Message Property Scopes
- Transaction Management
- Configuring Security
- Error Handling
- Using Web Services

## Understanding Mule Configuration

### Understanding Mule Configuration

[ About XML Configuration ] [ Schema References ] [ Default Values ] [ Enumerated Values ] [ Advantages of a Strongly Typed Language ] [ About Spring Configuration ] [ Spring Beans ] [ Spring Properties ] [ Property Placeholders ] [ About Mule Configuration ] [ Global Elements ] [ Flows ] [ Configuration Patterns ] [ Services ] [ Custom Elements ] [ System-level Configuration ] [ Managers ] [ Agents ]

### About XML Configuration

Mule uses an XML configuration to define each application, by fully describing the constructs required to run the application. A basic Mule application can use a very simple configuration, for instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/stdio/3.1/mule-stdio.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <stdio:connector name="stdio" promptMessage="Yes? " messageDelayTime="1000" />

    <flow name="echo">
        <stdio:inbound-endpoint system="IN" />
        <stdio:outbound-endpoint system="OUT" />
    </flow>

</mule>
```

We will examine all the pieces of this configuration in detail below. Note for now that, simple as it is, this is a complete application, and that it's quite readable: even a brief acquaintance with Mule makes it clear that it copies messages from standard input to standard output.

### Schema References

The syntax of Mule configurations is defined by a set of XML schemas. Each configuration lists the schemas it uses and give the URLs where they are found. The majority of them will be the Mule schemas for the version of Mule being used, but in addition there might be third-party schemas, for instance:

- Spring schemas, which define the syntax for any Spring elements (such as Spring beans) being used

- CXF schemas, used to configuration for web services processed by Mule's CXF module

Every schema referenced in a configuration is defined by two pieces of data:

- Its namespace, which is a URI
- Its location, which is a URL

The configuration both defines the schema's namespace URI as an XML namespace and associates the schema's namespace and location. This is done in the top-level `mule` element, as we can see in the configuration above:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/stdio/3.1/mule-stdio.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd" >
```

Shows the core mule schema's namespace being defined as the default namespace for the configuration. This is the best default namespace, because so many of the configuration's elements are part of the core namespace.

Shows the namespace for Mule's stdio transport, which allows communication using standard I/O, being given the "stdio" prefix. The convention for a Mule module or transport's schema is to use its name for the prefix.

The `xsi:schemaLocation` attribute associates schemas' namespaces with their locations. gives the location for the stdio schema and for the core schema.

It is required that a Mule configuration contain these things, because they allow the schemas to be found so that the configuration to be validated against them.

## Default Values

Besides defining the syntax of the elements and attributes that they define, schemas can also define default values for them. Knowing these can be extremely useful in making your configurations readable, since they won't have to be cluttered with unnecessary information. Default values can be looked up in the schemas themselves, or in the Mule documentation for the modules and transports that define them. For example, the definitions of the `<poll>` element, which polls an endpoint repeatedly, contains the following attribute definition:

```
<xsd:attribute name="frequency" type="substitutableLong" default="1000">
  <xsd:annotation>
    <xsd:documentation>Polling frequency in milliseconds. Default frequency is 1000ms (1s).
  </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
```

It is only necessary to specify this attribute when overriding the default value of 1 second.

## Enumerated Values

Many attributes in Mule can take only a limited set of values. These are defined in the schema as enumerated values, to ensure that only those values can be specified. Here's an example from the stdio transport:

```
<xsd:attribute name="system">
  <xsd:simpleType>
    <xsd:restriction base="xsd:NMTOKEN">
      <xsd:enumeration value="IN"/>
      <xsd:enumeration value="OUT"/>
      <xsd:enumeration value="ERR"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

This enforces that the only values allowed are IN, OUT, and ERR, corresponding to standard input, output, and error respectively.

## Advantages of a Strongly Typed Language

The requirement to reference all of the schemas may seem a bit cumbersome, but it has two advantages that far outweigh the effort.

First, it helps you create a valid configuration the first time. The major integrated development environments all provide schema-aware XML editors. Thus, as you create and edit your configuration, the IDE can prompt you with the elements and attributes that are allowed at each point, complete their names after you've typed a few characters, and highlight any typing errors that need correction. Likewise, it can provide the same help for filling in enumerated values.

Second, it allows Mule to validate your configuration as the application starts up. Unlike some other configuration-based systems that silently ignore elements or attributes that they don't recognize, Mule catches these errors so that you can correct them. For example, suppose that in the configuration above, we had misspelled "outbound-endpoint". As soon as the application tries to start up, the result would be the error:

```
org.mule.api.lifecycle.InitialisationException: Line 14 in XML document is invalid;
nested exception is org.xml.sax.SAXParseException: cvc-complex-type.2.4.a:
Invalid content was found starting with element 'stdio:outbound-endpint'.
```

This points directly to the line that needs to be corrected. It is much more useful than simply ignoring the problem and leaving you to wonder why no output is ever written.

## About Spring Configuration

The Mule facility for parsing configurations embeds Spring, so that a Mule configuration can, in addition to defining Mule-specific constructs, do anything a Spring configuration can do: create Spring Beans, configure lists and maps, define property placeholders, and so on. We will look at this in more detail in the following sections. Note that, as always, it will be necessary to reference the proper schemas.

### Spring Beans

The simplest use of Spring in a Mule configuration is to define Spring Beans. These beans are placed into the Mule registry along with the Mule-specific objects, where they can be looked up by name by any of your custom Java objects, for instance, custom components. You can use the full range of Spring capabilities to create them. For example:

```
<spring:beans>
  <spring:bean name="globalCache" class="com.mycompany.utils.LRUCache" >
    <spring:property name="maxItems" value="200"/>
  </spring:bean>
</spring:beans>
```

### Spring Properties

There are many places in a Mule configuration when a custom Java object can be used: custom transformers, filters, message processors, etc. In each case, one possibility is to specify the class to instantiate and a set of Spring properties to configure the resulting object. Once again, you can use the full range of Spring syntax within the properties, including lists, maps, etc.

Here's an example:

```
<custom-processor class="com.mycompany.utils.CustomerClassChecker">
  <spring:property name="highPriorities">
    <spring:list>
      <spring:value>Gold</spring:value>
      <spring:value>Platinum</spring:value>
      <spring:value>Executive</spring:value>
    </spring:list>
  </spring:property>
</custom-processor>
```

The syntax for creating custom components is a bit different, to allow more control over how the Java object is created. For instance, to create a singleton:

```

<component>
  <singleton-object class="com.mycompany.utils.ProcessByPriority">
    <properties>
      <spring:entry key="contents">
        <spring:list>
          <spring:value>Gold</spring:value>
          <spring:value>Platinum</spring:value>
          <spring:value>Executive</spring:value>
        </spring:list>
      </spring:entry>
    </properties>
  </singleton-object>
</component>

```

## Property Placeholders

Mule configurations can contain references to property placeholders, to allow referencing values specified outside the configuration file. One important use case for this is usernames and passwords, which should be specified in a more secure fashion. The syntax for property placeholders is simple: \${name}, where name is a property in a standard Java property file.

Here is an example of a configuration that uses property placeholders, together with the properties it references:

Configuration:

```

<spring:beans>
  <context:property-placeholder
    location="classpath:my-mule-app.properties,
              classpath:my-mule-app-override.properties" />
</spring:beans>

<http:endpoint name="ProtectedWebResource"
  user="${web.rsc.user}"
  password="${web.rsc.password}"
  host="${web.rsc.host}"
  port="80"
  path="path/to/resource" />

```

Properties file:

```

web.rsc.user=alice
web.rsc.password=s3cr3t
web.rsc.host=www.acme.com

```

Note the the location given for the file is a location in the classpath. Another alternative would be a URL, for instance <file:///etc/mule/conf/my-mule-app-override.properties>. As shown above, it is also possible to specify a list of properties files, comma-separated.

## About Mule Configuration

### Global Elements

Many Mule elements can be specified at the global level, that is, as direct children of the outermost `mule` element. These global elements always have names, which allows them to be referenced where they're used. Note that a Mule configuration uses a single, flat namespace for global elements. No two global elements can share the same name, even if they are entirely different sorts of things, say an endpoint and a filter.

Let's examine the most common global elements:

### Connectors

A connector is a concrete instance of a Mule transport, whose properties describe how that transport is used. All Mule endpoints use a connector with its same transport inherit the connector's properties.

Here are some examples of connectors:

```
<vm:connector name="persistentConnector">
  <vm:queueProfile persistent="true" />
</vm:connector>

<file:connector name="fileConnector"
  pollingFrequency="1000" moveToDirectory="/tmp/test-data/out" />
```

The vm connector specifies that all of its endpoints will use persistent queues. The file connector specifies that each of its endpoints will be polled once a second, and also the directory that files will be moved to once they are processed.

Note that properties may be specified either by attributes or by child elements. You can determine how to specify connector properties by checking the reference for that connector's transport.

The relationship between an endpoint and its connector is actually quite flexible:

- If an endpoint specifies a connector by name, it uses that connector. It is of course an error if the endpoint and the connector use different transports.
- If an endpoint does not name a connector, and there is exactly one connector for its transport, the endpoint uses that connector.
- If an endpoint does not name a connector, and there is no connector for its transport, Mule creates a default connector for all endpoints of that transport to use.
- It is an error if an endpoint does not name a connector, and there is more than one connector for its transport.

For transport-specific information about connector and endpoint configuration, see MULE3USER:Available Transports.

## Endpoints

A Mule endpoint is an object that messages can be read from (inbound) or written to (outbound), and that specifies properties that define how that will be done. Endpoints can be specified two different ways:

- An endpoint specified as a global element is called a global endpoint. An inbound or outbound endpoint, specified in a flow or service, can refer to a global endpoint using the `ref` attribute.
- An inbound or outbound endpoint, specified in a flow or service, can be configured without referring to a global endpoint.

A global endpoint specifies a set of properties, including its location. Inbound and outbound endpoints that reference the global endpoint inherit its properties. Here are some examples of global endpoints:

```
<vm:endpoint name="in" address="vm://in" connector-ref="persistentConnector" />
<http:endpoint name="in" host="localhost" port="8080" path = "services/orders" />
<endpoint name="inFiles" address="file:///c:/Orders" />
```

The vm endpoint at `in` specifies its location and refers to the connector shown above. It uses the generic `address` attribute to specify its location. The http endpoint at `in` uses the default http connector. Because it is explicitly configured as an http endpoint, it can use the http-specific attributes `host`, `port`, and `path` to specify its location. The file endpoint at `inFiles` specifies the directory it will read from (or write to), and will use the default file connector. Because it is configured as a generic endpoint, it must specify its location via `address`.

Note that every endpoint uses a specific transport, but that this can be specified in two different ways:

- If the element has a prefix, it uses the transport associated with that prefix. (, )
- If not, the prefix is determined from the element's `address` attribute. ( )

The prefix style is preferred, particularly when the location will be complex. Compare

```
<http:endpoint name="in" host="localhost" port="8080" path = "services/orders" user="${user.name}"
password ="${user.password}" />
```

and

```
<endpoint address="http://${user.name}:${user.password}@localhost:8080/services/orders"/>
```

One of the most important attributes of an endpoint is its message exchange pattern (MEP, for short), that is, whether message go only one way

or if requests return responses. This can be specified at several levels:

- some transports only support one MEP. For instance, imap is one way, because no response can be sent when it reads an e-mail message. servlet, on the other hand, is always request-response.
- every transport has a default MEP. jms is one-way by default, since JMS message are not usually correlated with responses. http defaults to request-response, since the HTTP protocol does naturally have a response for every request.
- endpoints can define MEPs, though (of course), only the MRPs that are legal for their transport are allowed

## Transformers

A transformer is an object that transforms the current Mule message. The Mule core defines a basic set of transformers, and many of the modules and transports define more, for instance the JSON module defines transformers to convert an object to JSON and vice-versa, while the Email transport defines transformers that convert between byte arrays and MIME messages. Each type of transformer defines XML configuration to define its properties. Here are some examples of transformers:

```
<json:json-to-object-transformer
    name="jsonToFruitCollection" returnClass="org.mule.module.json.transformers.FruitCollection">
    <json:deserialization-mixin
        mixinClass="org.mule.module.json.transformers.OrangeMixin" targetClass=
        "org.mule.tck.testmodels.fruit.Orange"/>
</json:json-to-object-transformer>

<message-properties-transformer name="SetInvocationProperty" scope="invocation">
    <add-message-property key="processed" value="yes" />
</message-properties-transformer>
```

The transformer at converts the current message to JSON, specifying special handling for the conversion of the org.mule.tck.testmodels.fruit.Orange class. The transformer at adds an invocation-scoped property to the current message.

Like endpoints, transformers can be configured as global elements and referred to where they are used, or configured at their point of use.

For more about Mule transformers, see [Using Transformers](#).

## Filters

A filter is an object that determines whether a message should be processed or not. As with transformers, the Mule core defines a basic set of filters, and many of the modules and transports define more. Here are some examples of filters:

```
<wildcard-filter pattern="* header received"/>

<mxml:is-xml-filter/>
```

The filter at continues processing of the current message only if it matches the specified pattern. The filter at continues processing of the current message only if it is an XML document.

There are a few special filters that extend the power of the other filters. The first is message-filter:

```
<message-filter onUnaccepted="deadLetterQueue">
    <wildcard-filter pattern="* header received"/>
</message-filter>

<message-filter throwOnUnaccepted="true">
    <mxml:is-xml-filter/>
</message-filter>
```

As above, continues processing of the current message only if it matches the specified pattern. But now any messages that don't match, rather than being dropped, are sent to a dead letter queue for further processing. continues processing of the current message only if it is an XML document, but throws an exception otherwise.

Other special filters are and-filter, or-filter, and not-filter, which allow you to combine filters into a logical expression:

```

<or-filter>
  <wildcard-filter pattern="*priority:1*" />
  <and-filter>
    <not-filter>
      <wildcard-filter pattern="*region:Canada*" />
    </not-filter>
    <wildcard-filter pattern="*priority:2*" />
  </and-filter>
</or-filter>

```

This processes a message only if it's either priority 1 or a priority 2 message from a country other than Canada.

Filters once again can be configured as global elements and referred to where they are used, or configured at their point of use. For more about Mule filters see [Using Filters](#)

## Expressions

Mule has a powerful expression facility that allows information for many different parts of the system to be used to affect message processing. Because many different parts of Mule can evaluate expressions, specifying an expression requires two things:

- The **evaluator**, which evaluates the expression. Mule supplies a long list of evaluators, or you can add your own.
- The **expression proper**, which is what's evaluated. The syntax of an expression is evaluator-specific.

There are two ways of specifying expressions depending on where the expression is being used. Typically, expression-based elements such as the expression transformer, expression filter, and expression-based routers such as the expression message splitter, will have specific attributes for expression, evaluator, and custom-evaluator. For example:

```
<expression-filter evaluator="header" expression="my-header!=null" />
```

When substituting expressions into string values, you use the syntax #[evaluator:expression], for instance:

```

<message-properties-transformer>
  <add-message-property name="GUID" value=
  "#[string:[xpath:/msg/header/ID]-#[xpath:/msg/body/@ref]]" />
</message-properties-transformer>

```

This nests expression calls: first the xpath evaluator is used twice to extract data from the current message, then the string evaluator is used to construct a string from them and a literal hyphen.

Expressions and property placeholders may seem similar, but they're actually quite different. Property placeholders are substituted at configuration time, and can be used to construct information that needs to be static. Expressions are substituted at run-time, so anything that uses them is dynamic. consider the following:

```

<vm:inbound-endpoint path="${vm.path}" />
<vm:inbound-endpoint path="#{header:vm.path}" /> <!-- illegal! -->
<vm:outbound-endpoint path="${vm.path}" />
<vm:outbound-endpoint path="#{header:vm.path}" />

```

is fine – it determines the endpoint's location at configuration time. (The property vm.path must be set, of course. is illegal. The address of an inbound endpoint must be set at configuration time, and the expression cannot be evaluated before the configuration is built.

is precisely like .

is something new – a dynamic endpoint. The location to which that endpoint will send a message is determined when that message is processed, and might be different each time. A message that doesn't define the property vm.path will cause an error, of course. For more about Mule expressions, see [Using Expressions](#).

## Names and References

As we've seen, many Mule objects can be defined globally. The advantage of this is that they can be reused throughout the application, by referring to them where they're needed. There's a common pattern for this:

- The global object is given a name using the `name` attribute
- It is referred to using the `"ref"` attribute

For each type of object, there is a generic element used to refer to it.

- All global transformers are referred to by the `transformer` element
- All global message processors are referred to by the `processor` element
- All global endpoints are referred to by the `inbound-endpoint` or `outbound-endpoint` elements
- All global filters are referred to by the `filter` element

For example

```
<vm:endpoint name="in" address="vm://in" connector-ref="persistentConnector" />
<expression-filter name="checkMyHeader" evaluator="header" expression="my-header!>
<message-properties-transformer name="SetInvocationProperty" scope="invocation">
  <add-message-property key="processed" value="yes" />
</message-properties-transformer>

<flow name="useReferences">
  <vm:inbound-endpoint ref="in"/>
  <filter ref="checkSetInvocationPropertyMyHeader" />
  <transformer ref="" />
</flow>
```

In addition, there are places where the names of global objects are the values of an attribute, for instance:

```
<vm:endpoint name="in" address="vm://in" transformer-refs="canonicalize sort createHeaders" />
```

## Flows

The flow is the basic unit of processing in Mule. A flow begins with an inbound endpoint from which messages are read and continues with a list of message processors, optionally ending with an outbound endpoint, to which the fully processed message is sent. We've already met some types of message processors: transformers and filters. Other types include components, which process messages using languages like Java or Groovy, cloud connectors, which call cloud services, and routers, which can alter the message flow as desired. Below is a simple flow, which we'll be referring to as we examine its parts:

```

<flow name="acceptAndProcessOrder">
    <http:inbound-endpoint ref="in"/>
    <byte-array-to-string-transformer/>
    <jdbc:outbound-endpoint ref="getOrdersById" exchange-pattern="request-response" />
    <mxml:object-to-xml-transformer/>
    <expression-filter evaluator="xpath" expression="/status = 'ready' />
    <logger level="DEBUG" message="fetched orders: #[payload]" />
    <splitter evaluator="xpath" expression="/order"/>

    <enricher>
        <authorize:authorization-and-capture amount="#[xpath:/amount]"
            cardNumber="#[xpath:/card/number]"
            expDate="#[xpath:/card/expire]" />
        <enrich target="#[variable:PaymentSuccess]" source="#[bean:responseCode]" />
    </enricher>
    <message-properties-transformer scope=:invocation>
        <add-message-property key="user-email-address" value="#[xpath:/user/email]" />
    </message-properties-transformer>
    <component class="org.mycompany.OrderPreProcessor" />
    <flow-ref name="processOrder" />
    <smtp:outbound-endpoint subject="Your order has been processed"
        to="#[header:INVOCATION:user-email-address]" />

    <default-exception-strategy>
        <processor-chain>
            <object-to-string-transformer/>
            <jms:outbound-endpoint ref="order-processing-errors" />
        </processor-chain/>
    </default-exception-strategy>
</flow>

```

This flow, as you would expect from its name, accepts and processes orders. Note as we go through it, how the flow configuration's maps exactly to the logic it describes:

A message is read from an HTTP endpoint.

The message is transformed to a string.

This string is used as a key to look up the list of orders in a database.

The order is now converted to XML.

If the order is not ready to be processed, it is skipped.

The list is optionally logged, for debugging purposes.

Each order in the list is split into a separate message

A message enricher is used to add information to the message

Authorize.net is called to authorize the order

The email address in the order is saved for later use.

A Java component is called to preprocess the order.

Another flow, named processOrder, is called to process the order.

The confirmation returned by processOrder is e-mailed to the address in the order.

If processing the order caused an exception, the exception strategy at is called:

All the message processors in this chain are called to handle the exception

First, the message is converted to a string.

Last, this string is put on a queue of errors to be manually processed.

Each step in this flow is described in more detail below, organized by construct.

## Endpoints

Previously, we looked at declarations of global endpoints. Here we see endpoints in flows, where they are used to receive (inbound) and send (outbound) messages. Inbound endpoints appear only at the beginning of the flow, where they supply the message to be processed. Outbound endpoints can appear anywhere afterward. The path of a message through a flow depends upon the MEPs of its endpoints:

- If the inbound endpoint is request-response, the flow will, at its completion, return the current message to its caller.
- If the inbound endpoint is one-way, the flow will, at its completion, simply exit
- When the flow comes to a request-response outbound endpoint, it will send the current message to that endpoint, wait for a response, and make that response the current message
- When the flow comes to a one-way outbound endpoint, it will send the current message to that endpoint and continue to process the current message

This receives a message over an HTTP connection. The message payload is set to an array of the bytes received, while all HTTP headers become inbound message properties. Because this endpoint is request-response (the default for http), at the end of the flow, the current message will be returned to the caller.

The calls a JDBC query, using the current message as a parameter, and replaces the current message with the query's result. Because this endpoint is request-response, the result of the query will become the current message.

The confirmation for a completed order, which was returned from the sub-flow, is e-mailed. Note that we use the email-address that had previously been saved in a message property. Because this endpoint is one-way (the only MEP for email transports), the current message will not change.

Any orders that were not processed correctly are put on a JMS queue for manual examination. Because this endpoint is one-way (the default for jms), the current message will not change.

Thus the message sent back to the caller will be the confirmation message, in case of success, or the same string sent to the JMS error queue in case of failure.

## **Transformers**

As described above, transformers change the current message. There are a few examples here. Note that they are defined where used. They could also have been defined globally and referred to where used.

The message, which is a byte array, is converted to a string, allowing it to be the key in a database look-up.

The order read from the database is converted to an XML document.

The email address is stored in a message property. Note that, unlike most transformers, the message-properties-transformer does not affect the message's payload, only its properties.

The message that caused the exception is converted to a string. Note that since the same strategy is handling all exceptions, we don't know exactly what sort of object the message is at this point. It might be a byte array, a string, or an XML document. Converting all of these to strings allows its receiver to know what to expect.

## **Message Enrichment**

Message enrichment is done using the `enricher` element. Unlike message transformation, which alters the current message's payload, enrichment adds additional properties to the message. This allows the flow to build up a collection of information for later processing. For more about enriching messages see [Message Enricher](#).

The enricher calls a cloud connector to retrieve information that will be stored as a message property. Because the cloud connector is called within an enricher, its return value is processed by the enricher rather than becoming the message.

## **Logger**

The `logger` element allows debugging information to be written from the flow. For more about the logger see [Logger Element for Flows](#)

Each order fetched from the database is output, but only if DEBUG mode is enabled. This means that the flow will ordinarily be silent, but debugging can easily be enabled when required.

## **Filters**

Filters determine whether a message is processed or not.

If the status of the document fetched is not "ready", its processing is skipped.

## **Routers**

A router changes the flow of the message. Among other possibilities, it might choose among different message processors, split one message into many, join many messages into one. For more about routers, see [Routing Message Processors](#).

Split the document retrieved from the database into multiple orders, at the XML element `order`. The result is zero or more orders, each of which is processed by the rest of the flow. That is, for each HTTP message received, the flow is processed once up through the splitter. The rest of the flow might be processed zero, one, or more times, depending on how many orders the document contains.

## **Components**

A component is a message processor written in Java, groovy, or some other language. Mule determines which method to call on a component by finding the best match to the message's type. For instance, if the message is an `InputStream`, it searches for a public method that takes one argument of type `InputStream` (or `Stream`, or `Object`). To help tailor this search, Mule uses objects calls Entry Point Resolvers, which are configured on the component. Here are some examples of that:

```

<component class="org.mycompany.OrderPreProcessor">
<entry-point-resolver-set>
    <method-entry-point-resolver>
        <include-entry-point method="preProcessXMLOrder" />
        <include-entry-point method="preProcessTextOrder" />
    </method-entry-point-resolver>
    <reflection-entry-point-resolver/>
</entry-point-resolver-set>
</component>

<component class="org.mycompany.OrderPreProcessor">
    <property-entry-point-resolver property="methodToCall" />
</component>

<component class="org.mycompany.generateDefaultOrder">
    <no-arguments-entry-point-resolver>
        <include-entry-point method="generate" />
    </no-arguments-entry-point-resolver>
</component>

```

causes the two methods `preProcessXMLOrder` and `preProcessTextOrder` to become candidates. Mule chooses between them by doing reflection, using the type of the message.

calls the method whose name is in the message property `methodToCall`.  
call the `generate` method, even though it takes no arguments

Entry point resolvers are for advanced use. Almost all of the time, Mule will find the right method to call without needing any special guidance.

This is a Java component, specified by its class name, which is called with the current message. In this case, it preprocesses the message. For more about entry point resolvers, see [Entry Point Resolver Configuration Reference](#).

### **Cloud Connectors**

A cloud connector calls a cloud service.

calls authorize.net to authorize a credit card purchase, passing it information from the message. For more about Cloud Connectors, see [Integrating with Cloud Connect](#).

### **Processor Chain**

A processor chain is a list of message processors, which will be executed in order. It allows you to use more than one processor where a configuration otherwise allows only one, exactly like putting a list of Java statements between curly braces.

is used to perform two steps as part of the exception strategy. It first transforms and then mails the current message.

### **Sub-flow**

A sub-flow is a flow that can be called from another flow. It represents a reusable processing step. Calling it is much like calling a Java method, the sub-flow is passed the current message, and when it returns the calling flow resumes processing with the message that the sub-flow returns.

calls a flow to process an order that has already been pre-processed and returns a confirmation message..

### **Exception Strategies**

An exception strategy is called whenever an exception occurs in its scope, much like an exception handler in Java. It can define what to do with any pending transactions and whether the exception is fatal for the flow, as well as logic for handling the exception.

writes the message that caused the exception to a JMS queue, where it can be examined. For more about exception strategies, see [MULE3USER:Exception Handling](#).

### **Configuration Patterns**

Flows have the advantages of being powerful and flexible. Anything that Mule can do can be put into a flow. Mule also comes with configuration patterns, each of which is designed to simplify a common use of Mule. It's worthwhile to become familiar with the patterns and use them when possible, for the same reasons that you would use a library class rather than built the same functionality from scratch. There are currently four configuration patterns:

- `pattern:bridge` bridges between an inbound endpoint and an outbound endpoint
- `pattern:simple-service` is a simple flow from one inbound endpoint to one component
- `pattern:validator` is like a one-way bridge, except that it validates the message before sending it to the outbound endpoint
- `pattern:web-service-proxy` is a proxy for a web service.

As of mule 3.1.1, all are in the pattern namespace as shown. In earlier Mule 3 releases, they are in the core namespace, except for `web-service-proxy` which is `ws:proxy`. These older names will continue to work for the Mule 3.1.x releases, but will be removed after that.

### **Common features.**

For flexibility, all of the patterns allow endpoints to be specified in a variety of ways:

- local endpoints can be declared as sub-elements, as in flow
- references to global elements can be declared as sub-elements, as in flow
- references to global elements can be declared as values of the attributes `inboundEndpoint-ref` and `outboundEndpoint-ref`
- the endpoint's address can be given as the value of the attributes `inboundAddress` and `outboundAddress`

All configuration patterns can specify exception strategies, just as flows can.

### **Bridge**

This allows you to configure, in addition to the inbound and outbound endpoints

- a list of transformers to be applied to requests
- a list of transformers to be applied to response
- whether to process messages in a transaction.

Here are some examples:

```
<pattern:bridge name="queue-to-topic"
    transacted="true"
    inboundAddress="jms://myQueue"
    outboundAddress="jms://topic:myTopic" />

<pattern:bridge name="transforming-bridge"
    inboundAddress="vm://transforming-bridge.in"
    transformer-refs="byte-array-to-string"
    responseTransformer-refs="string-to-byte-array"
    outboundAddress="vm://echo-service.in" />
```

copies message from a JMS queue to a JMS topic, using a transaction. reads byte arrays from an inbound vm endpoint, transforms them to strings, and writes them to an outbound vm endpoint. The response are strings, which are transformed to byte arrays, and them written to the inbound endpoint.

### **Simple Service**

This allows you to configure, in addition to the inbound endpoint

- a list of transformers to be applied to requests
- a list of transformers to be applied to response
- a component
- a component type, which allows you to use Jersey and CXF components.

Here are some examples:

```
<pattern:simple-service name="echo-service"
    endpoint-ref="echo-service-channel"
    component-class="com.mycompany.EchoComponent" />

<pattern:simple-service name="weather-forecaster-ws"
    address="http://localhost:6099/weather-forecast"
    component-class="com.mycompany.WeatherForecaster"
    type="jax-ws" />
```

Is a simple service that echos requests. is a simple web service that uses a CXF component. Note how little configuration is required to create them.

### Validator

The allows you to configure, in addition to the inbound and outbound endpoints

- a list of transformers to be applied to requests
- a list of transformers to be applied to response
- a filter to perform the validation
- expressions to create responses to indicate that the validation succeeded or failed

Here is an example:

```
<pattern:validator name="validator"
    inboundAddress="vm://services/orders"
    ackExpression="#[string:OK]"
    nackExpression="#[string:illegal payload type]"
    outboundAddress="vm://OrderService">
    <payload-type-filter expectedType="com.mycompany.Order"/>
</pattern:validator>
```

validates the the payload is of the correct type before calling the order service, using the filter at .

### Web service proxy

This create a proxy for a web service. It modifies the advertised WSDL to contain the proxy's URL.

The allows you to configure, in addition to the inbound and outbound endpoints

- a list of transformers to be applied to requests
- a list of transformers to be applied to response
- The location of the service's WSDL, either as a URL as as a file name.

Here is an example:

```
<pattern:web-service-proxy name="weather-forecast-ws-proxy"
    inboundAddress="http://localhost:8090/weather-forecast"
    outboundAddress="http://server1:6090/weather-forecast"
    wsdlLocation="http://server1:6090/weather-forecast?wsdl" />
```

This creates a proxy for the weather forecasting service located on server1.

For more about configuration patterns, see [Using Mule Configuration Patterns](#) .

## Services

Services are an older feature of Mule. They are not as flexible as flows are nor as friendly as configuration patterns. While services remain fully supported, it is recommended that new development be done with flows and patterns. That having been said, services use many of the same ideas as flows and are not difficult to use or construct.

A service is divided into three parts:

- Inbound. This contains the inbound endpoint plus any processing that precedes the single component that a service is allowed. This can consist of inbound routers, transformers, filters, and other message processors.
- Component. This is the same component found in flows. It is optional
- Outbound. This is all of the processing that follows the component. It consists of a set of outbound routers. The simplest of these is the pass-through router, which simply passes the message to the outbound endpoint.

Services, like flows and pattern, can also define exception strategies.

Services live inside a construct called a model, which groups service and allows them to share some configuration:

- exception strategies
- entry point resolves for components

Putting all of this together, we can compare using flows, configuration patterns, and services to create a simple application that copies standard

output:

```
<stdio:outbound-endpoint name="out" system="OUT"/>
<stdio:outbound-endpoint name="in" system="IN"/>

<!-- flow-->
<flow name="echoFlow">
    <stdio:inbound-endpoint ref="in"/>
    <stdio:outbound-endpoint ref="out"/>
</flow>

<!-- bridge pattern -->
<pattern:bridge name="echoBridge"
    inboundEndpoint-ref="in"
    outboundEndpoint-ref="out" />

<!-- service -->
<model name="echoModel">
    <service name="echoService">
        <inbound>
            <stdio:inbound-endpoint ref="in"/>
        </inbound>
        <outbound>
            <pass-through-router>
                <stdio:outbound-endpoint ref="out"/>
            </pass-through-router>
        </outbound>
    </service>
</model>
```

The bridge pattern is the simplest, with flows not far behind. The service isn't hard to read, but it's significantly longer and more complicated for no real gain. For more about services, see [Service Configuration Reference](#).

## Custom Elements

Mule is extensible, meaning that you can create your own objects (often by extending Mule classes). After you've done this, there are standard way to place them into the configuration. Assume, for instance, that you've created `com.mycompany.HTMLCreator`, which converts a large variety of document types to HTML. It should be a Spring bean, meaning

- It has a default constructor
- It is customized by setting bean properties

You can now put it into your configuration using the `custom-transformer` element:

```
com.mycompany.HTMLCreator
<custom-transformer mimeType="text/html" returnType="java.lang.String" class=
"com.mycompany.HTMLCreator">
    <spring:property name="level" value="HTML5" />
    <spring:property name="browser" value="Firefox" />
</custom-transformer>
```

Note that the standard Mule properties for a transformer are specified the usual way. The only differences are that the object itself is created via its class name and Spring properties rather than via schema-defined elements and attributes. Each type of Mule object has an element used for custom extensions:

- `custom-connector` for connectors
- `custom-entry-point-resolver` for entry point resolvers
- `custom-exception-strategy` for exception strategies
- `custom-filter` for filters
- `custom-processor` for message processors
- `custom-router` for routers
- `custom-transformer` for transformers

## System-level Configuration

The configuration contains several global settings that affect the entire mule application. All are children of the `configuration` element, which itself is a top-level child of `mule`. They fall into two groups: threading profiles and timeouts.

## **Threading Profiles**

Threading profiles determine how Mule manages its thread pools. In most cases the default will perform well, but if you determine that, for instance, your endpoints are receiving so much traffic that they need additional threads to process all of it, you can adjust this, either for selected endpoint or, by changing the default, for all endpoints. The defaults that can be adjusted are and their corresponding elements are:

- `default-threading-profile` all thread pools
- `default-dispatcher-threading-profile` for the thread pools used to dispatch (send) messages
- `default-receiver-threading-profile` for the thread pools used to receive messages
- `default-service-threading-profile` for the thread pools used to process services

## **Timeouts**

Again, the default timeouts will usually perform well, but if you want to adjust them, you can do say either per use or globally. The timeouts that can be adjusted and their corresponding attributes are:

- `defaultResponseTimeout` How long, in milliseconds, to wait for a synchronous response. The default is 10 seconds.
- `defaultTransactionTimeout` How long, in milliseconds, to wait for a transaction to complete. The default is 30 seconds.
- `shutdownTimeout` How long, in milliseconds, to wait for Mule to shut down gracefully. The default is 5 seconds.

## **Managers**

There are several global objects used to manage system-level facilities used by Mule. They are discussed below.

### **Transaction manager**

Mule uses JTA to manage XA transactions; thus, to use XA transactions, a JTA transaction manager is required, and must be specified in the configuration. Mule has explicit configuration for many of these, and, as usual, also allows you to specify a custom manager. The element used to specify a transaction manager is a direct child of `mule`.

- `websphere-transaction-manager` for the WebSphere transaction manager
- `jboss-transaction-manager` for the JBoss transaction manager
- `* weblogic-transaction-manager` for the WebLogic transaction manager
- `jrun-transaction-manager` for the JRun transaction manager
- `resin-transaction-manager` for the Resin transaction manager
- `*jndi-transaction-manager` to look up a transaction manager in JNDI
- `*custom-transaction-manager` for a custom lookup of the transaction manager

The starred transaction managers allow you to configure a JNDI environment before performing the lookup. For more about transaction managers, see [Transaction Management](#).

### **Security Manager**

The Mule security manager can be configured with one or more encryption strategies that can then be used by encryption transformers, security filters, or secure transports such as SSL or HTTPS. These encryption strategies can greatly simplify configuration for secure messaging as they can be shared across components. This security manager is set with the global `security-manager` element, which is a direct child of `mule`.

For example, here is an example of a password-based encryption strategy (PBE) that provides password-based encryption using JCE. Users must specify a password and optionally a salt and iteration count as well. The default algorithm is `PBEWithMD5AndDES`, but users can specify any valid algorithm supported by JCE.

```
<security-manager>
  <password-encryption-strategy name="PBE" password="mule" />
</security-manager>
```

This strategy can then be referenced by other components in the system such as filters or transformers.

```

<decrypt-transformer name="EncryptedToByteArray" strategy-ref="PBE"/>

<flow name="testOrderService">
  <inbound-endpoint address="vm://test">
    <encryption-security-filter strategy-ref="PBE" />
  </inbound-endpoint>
  ...
</flow>

```

For more about Mule security, see [Configuring Security](#).

### **Notifications Manager**

Mule can generate notifications whenever a message is sent, received, or processed. For these notifications to actually be created and sent, objects must register to receive them. This is done via the `global{{notifications}}` element, which is a direct child of `mule`. It allows you to specify an object to receive notifications as well as specify which notifications to send it. Note that an object will only receive notifications for which it implements the correct interface (these interfaces are defined in the `org.mule.api.context.notification.package`.) Here is an example:

```

<spring:bean name="componentNotificationLogger"
  class="org.myfirm.ComponentMessageNotificationLogger"/>

<spring:bean name="endpointNotificationLogger"
  class="org.myfirm.EndpointMessageNotificationLogger"/>

<notifications>
  <notification event="COMPONENT-MESSAGE"/>
  <notification event="ENDPOINT-MESSAGE"/>
  <notification-listener ref="componentNotificationLogger"/>
  <notification-listener ref="endpointNotificationLogger"/>
</notifications>

```

Assume that `ComponentMessageNotificationLogger` implements the `ComponentMessageNotificationListener` interface and `EndpointMessageNotificationLogger` implements `EndpointMessageNotificationListener`.

and create the listener beans. appears to register both beans for both component and endpoint notifications. But since `ComponentMessageNotificationLogger` only implements the interface for component notification, those are all it will receive (and likewise for `EndpointMessageNotificationLogger`).

For more about notifications, see [Notifications Configuration Reference](#).

### **Agents**

Mule allows you to define Agents to extend the functionality of Mule. Mule will manage the agents' lifecycle (initialize them and start them on startup, and stop them and dispose of them on shutdown). These agents can do virtually anything; the only requirement is that they implement `org.mule.api.agent.Agent`, which allows Mule to manage them. For more about Mule agents, see [Mule Agents](#).

### **Custom Agents**

To create a custom agent, simply declare it using the global `custom-agent` element, which is a direct child of `mule`. The agent is a Spring bean, so as usual it requires a class name and a set of Spring properties to configure it. In addition it requires a name, which Mule uses to identify it in logging output. Here's an example:

```

<custom-agent name="heartbeat-agent" class="com.mycompany.HeartbeatProvider">
  <spring:property name="frequency" value="30"/>
<custom-agent>

```

This create an agent that issues a heartbeat signal every 30 seconds. Since Mule will start it and stop it, the heartbeat is present precisely when the Mule server is running.

### **Management Agents**

Mule implements various management agents in the management namespace.

- management:jmx-server creates a JMX server that allows local or remote access to Mule's JMX beans
- management:jmx-mx4j-adaptor creates a service that allows HTTP access to the JMX beans
- management:rmi-server creates a service that allows RMI access to the JMX beans
- management:jmx-notifications creates an agent that propagates Mule notifications to JMX
- management:jmx-log4j allows JMX to manage Mule's use of Log4J
- management:jmx-default-config allows creating all of the above at once
- management:log4j-notifications creates an agent that propagates Mule notifications to Log4J
- management:chainsaw-notifications creates an agent that propagates Mule notifications to Chainsaw
- management:publish-notifications creates an agent that publishes Mule notifications to a Mule outbound endpoint
- management:yourkit-profiler creates an agent that exposes YourKit profiling information to JMX

Your Rating: 

Results:  3 rates

## About the XML Configuration File

### About the XML Configuration File

[ XML Schema ] [ Namespaces ] [ Spring ] [ Merging Configuration Files ]

The most common way to configure Mule ESB is with Spring XML files that use custom Mule namespaces. This page describes the format of the file. Note that the XML schema and namespace declarations are filled in automatically when you use the [Mule IDE](#).

#### XML Schema

The configuration files are based on XML schemae, which are specified in the header:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">
```

Be sure to specify all the necessary schema files. This can be time-consuming when setting up the configuration file, but importing schemae provides the following time-saving benefits:

- Auto-completion and context-specific help in your favorite IDE
- Design-time configuration validation
- Typed properties

#### Namespaces

Each Mule module or transport has its own XML schema. When you import a schema, it has its own *namespace*. For example, the following lines from the previous header declare the `jms` namespace:

```
xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/jms
http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd"
```

This example binds the `mule-jms.xsd` schema to the `jms` namespace. Therefore, any XML element starting with `<jms:` will assume the element comes from the `mule-jms.xsd` schema.

#### Default Namespace

Typically, you set the Mule core schema as the default namespace for your configuration file. This means that any XML element without a prefix will come from the Mule core schema, (`mule.xsd`). To set the default namespace schema, specify `xmlns` immediately followed by the URL of the Mule schema, without the colon or namespace prefix you set in the previous example (e.g., `xmlns` instead of `xmlns:jms`):

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/core
      http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">
    ...config...
</mule>

```

## Spring

Although your configuration files appear to be Mule-specific, they are really just Spring configuration files with Mule-specific extensions. This approach allows you to use anything Spring offers within your Mule configuration, such as beans, factory beans, resource loaders, EJBs, JNDI, AOP, and even integration with other software such as jBPM, Gigaspaces, JBoss Rules, etc.

To use the standard Spring elements, import the standard Spring namespaces:

```

xmlns:spring="http://www.springframework.org/schema/beans"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd"
...
<spring:bean id="myBean" class="com.acme.CoolBean">
  <spring:property name="sessionFactory">
    <spring:ref local="mySessionFactory" />
  </spring:property>
  <spring:property name="configuration">
    <spring:value>my-config.xml</spring:value>
  </spring:property>
</spring:bean>

```

For complete information on Spring configuration, see the [Spring Framework reference documentation](#).

## Merging Configuration Files

If you have multiple configuration files, you can import them into one configuration file so that you only have to specify one configuration. For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns=http://www.mulesoft.org/schema/mule/core ....>
  <spring:beans>
    <spring:import resource="mule-sub-config1.xml" />
    <spring:import resource="mule-sub-config2.xml" />
  </spring:beans>
...

```

Your Rating: 

Results:  3 rates

## Choosing Between Flows, Patterns, or Services

### Choosing Between Flows, Patterns, or Services

Because of the emphasis on ease of use, simplicity, and flexibility that guided the creation of Mule 3, a number of choices are open at the outset.

#### Choosing Configuration Patterns and Flows

Take a look at [configuration patterns](#) and [flows](#). These will allow you to go faster and further than you could using services alone. If you're a new user, start first by looking at configuration patterns.

Mule 3 includes a number of configuration patterns designed to solve common integration problems with minimal XML. These patterns are similar to Enterprise Integration Patterns which are also widely supported in Mule but tackle bigger use cases, like creating a web service proxy. Take a moment to inspect the list of patterns.

If your project requires a more complex set of operations, you might want to consider flows, which are introduced in Mule 3.

For more information about patterns, read the Configuration Reference.

- [Asynchronous Reply Router Configuration Reference](#)
- [Catch-all Strategy Configuration Reference](#)
- [Component Configuration Reference](#)
- [Endpoint Configuration Reference](#)
- [Exception Strategy Configuration Reference](#)
- [Filters Configuration Reference](#)
- [Global Settings Configuration Reference](#)
- [Inbound Router Configuration Reference](#)
- [Model Configuration Reference](#)
- [Notifications Configuration Reference](#)
- [Outbound Router Configuration Reference](#)
- [Properties Configuration Reference](#)
- [Security Manager Configuration Reference](#)
- [Service Configuration Reference](#)
- [Transactions Configuration Reference](#)
- [BPM Configuration Reference](#)

## Choosing Services

Mule's service-based architecture is well-suited for many scenarios where you are using Mule as a service container and exposing services over one or more endpoints. If you're familiar with using previous versions of Mule, you can continue - with very few changes - doing things the same way.

- [Using Flows for Service Orchestration](#)
- [Using Mule Services](#)
- [Using Mule Configuration Patterns](#)

## Using Flows for Service Orchestration

### Using Flows for Service Orchestration

[ [Introduction](#) ] [ [When to Use a Flow](#) ] [ [The Anatomy of a Flow](#) ] [ [Flow Configuration](#) ] [ [Example](#) ] [ [Flow Behavior](#) ] [ [Private Flows](#) ] [ [Further Reading](#) ]

#### Introduction

A **Flow** is a simple yet very flexible mechanism that enables orchestration of services using the sophisticated message flow capabilities of Mule ESB. Using Flow, you may automate integration processes and construct sophisticated integration solutions by simply building them from the basic [Message Sources and Message Processors](#) provided by Mule. Because of the flexibility of Flow, it is much easier to create solutions that more closely match your requirements. Flow is new in Mule 3.

#### When to Use a Flow



A flow is the most versatile and powerful integration mechanism available in Mule.

In contrast to the use of [Services](#), which define a component with explicit inbound and outbound phases that allow a limited amount of flexibility, a **Flow** does not define anything and is completely free-form. This means that if your solution requires multiple steps you won't need to use multiple services glued together with the vm transport or a chaining router. Instead you'll be able to configure everything in the same flow.

Flows are valuable in many situations, including:

- Simple integration tasks
- Scheduled data processing
- Connecting cloud and on-premise applications
- Event processing where multiple services need to be composed

Before you create flows be sure to take a look at the ready-to-use [configuration patterns](#). They can save you a great amount of configuration effort for common use cases.

You still may choose to use [Services](#) in some cases. For example, if all you wish to do is expose a single service backed by a service component and your requirements are more complex than Simple Service supports, then a Mule Service is still a good choice. Existing Mule services can also be easily integrated into a flow. Finally, flows do not yet support all features of Mule. In particular, services are still required if you want to do any of the following:

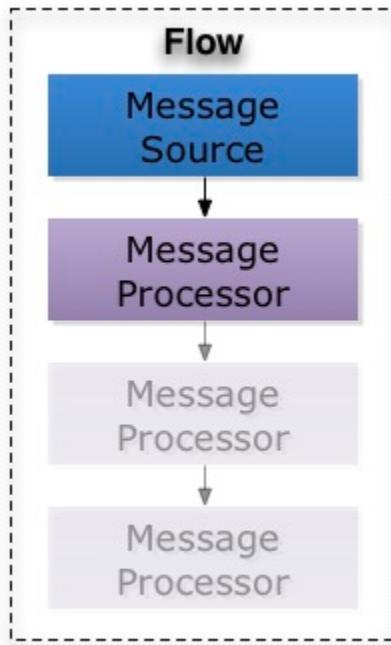
- Transactions configured anywhere other than on endpoints e.g. on routers (including local, multi-tx, and XA)

- Request-Reply

Future releases of Mule will enable this functionality for Flow.

### The Anatomy of a Flow

A flow is in essence just a chain of Message Processors. Think of each Message Processor as a Lego block where a Flow is something you build with them. A flow also has a [Message Source](#), the message source is the source of messages that are processed by the Message Processor chain.



### Flow Configuration

A Flow is configured in XML using the `<flow>` element. Each flow has a name attribute, a message source (unless it's a private flow), one or more message processors and an optional exception strategy.

#### Basic Structure

```

<flow name="">
    - 0..1 MessageSource
    - 1..n MessageProcessor(s)
    - 0..1 ExceptionStrategy
</flow>
  
```

Flows seem simple, yet can be quite powerful. In particular, when combined with expressions in Mule, they can allow for very sophisticated processing of the message contents. There are many elements that leverage expressions, including:

- Transformers
- Filters
- Routing
- Message Enricher

### Example

### Simple Book Order Processing Flow

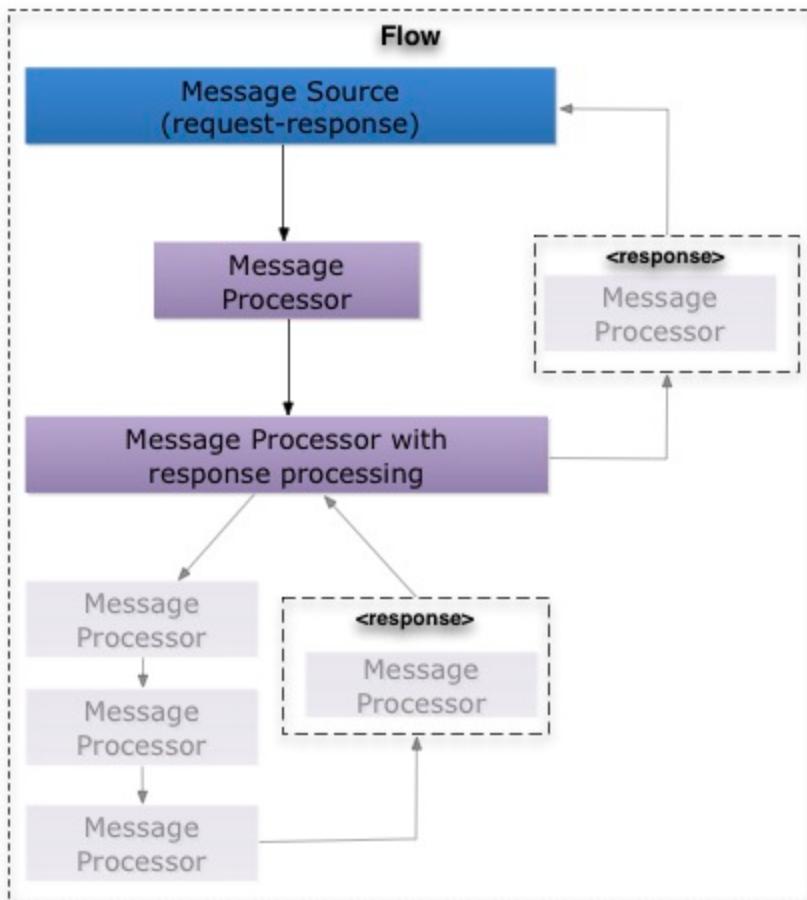
```
<flow>
    <file:inbound-endpoint path="/myDirectory">
        <file:filename-filter name="*.xml"/>
    </file:inbound-endpoint>
    <xml:xslt-transformer xsl-file="bookOrderTransformation.xsl"/>
    <splitter expression="xpath://order"/>
    <!-- The following message processors will be invoked for each order in the xml file -->
    <expression-filter expression="xpath://order[@type='book']"/>
    <component class="org.my.BookOrderProcessor"/>
    <smtp:outbound-endpoint subject="Order Confirmation" address="" />
    <jdbc:outbound-endpoint />
    <default-exception-strategy>
        <jms:outbound-endpoint queue="failedOrders"/>
    </default-exception-strategy>
</flow>
```

### Flow Behavior

When a message is received or generated by the message source the flow is started and the configured message processors are invoked in a chain in the same order as they are configured. Some message processors will accept child message processor elements, in this case these will be processed before returning and continuing processing the main list.

The above describes the behavior when the flow is *one-way*. If the flow is *request-response* because an inbound endpoint as a *request-response* exchange pattern defined then the result of the flow execution is return to the inbound endpoint and then in turn to the callee. If there are no *<response>* blocks in your flow and if none of the configured message processors perform any response processing then the response used is simply the result from the last Message Processor in the flow. If a *<response>* block is used then any message processors configured in this element will be used to process the response message. Some message processors (e.g. CXF) perform processing of the response message as part of their default configuration.

**Note:** When the last element in the flow configuration is a *one-way* *<outbound-endpoint>* there's no result of it's execution so the returned payload of the message is going to be NullPayload. If the *one-way* *<outbound-endpoint>* is followed by another processor it will receive as input the same message that the outbound-endpoint received instead of NullPayload.



## Private Flows

A private flow is one that cannot be accessed from outside the JVM via a [Mule Endpoint](#) because it has no Message Source defined.

Private Flows are therefore only used if they are referenced from another construct running in the same Mule instance. When configuring Mule using XML the `<flow-ref>` element is used to include one flow in another.

A private Flow differs from the use of a "Processor Chain" in that a Flow has its own context and Exception Strategy where as when a processor chain is referenced it is executed in the context of the flow or service that references it.

### Private Flow Example

```

<flow name="privateFlow">
    <append-string-transformer message="b" />
</flow>

<flow name="publicFlow">
    <http:inbound-endpoint address="http://localhost:8080" />
    <append-string-transformer message="a" />
    <flow-ref name="privateFlow"/>
    <append-string-transformer message="c" />
</flow>

```

## Further Reading

You can read more about the reason we added Flow for Mule 3 in the following blog posts:

[Mule 3 Architecture, Part 1: Back to Basics](#)

[Mule 3 Architecture, Part 2: Introducing the Message Processor](#)

## Using Mule Services

### Using Mule Services

#### When to Use a Service



Unless you specifically want to use Service, you should instead look at using [Configuration Patterns](#) or [Flows](#) instead.

More flexible than [Configuration Patterns](#) but less versatile than [Flows](#), services are the *Mule 2 way* of configuring integration.

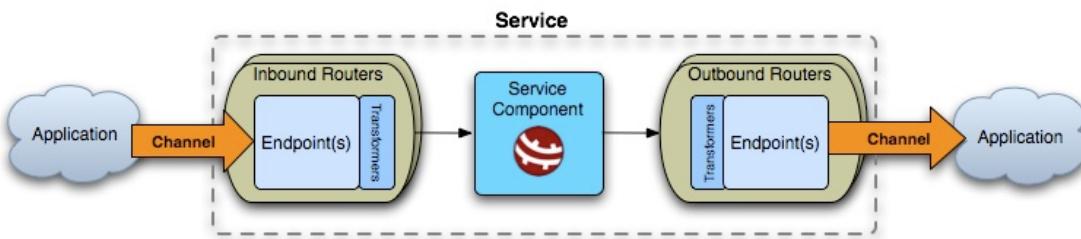
A service provides a fixed framework for building integration mechanisms around the concept of inbound routers and endpoints, a single service component and outbound routers and endpoints.

To achieve complex routing or processing chains, several services, tied together by [VM queues](#), are often needed. In contrast, a single [flow](#) element can achieve such complex integration paths.

#### Service Components

A *service component* is a class, web service, or other application that contains the business logic you want to plug in to the Mule ESB framework. For example, one service component could add information to an invoice from a customer database, and another service component could be the order fulfillment application that processes that invoice. You can use any existing application for a service component, or create a new one.

Your service component does not need to contain any Mule-specific code. Instead, you configure a *service*, which wraps the service component with the Mule-specific configuration. The service configuration points to the service component, as well as routers, filters, and transformers to use when carrying messages to and from the service component. It also specifies the endpoint on which this service will receive messages and the outbound endpoint where messages will go next.



To watch a demo of building a simple service, click [here](#).

A service component can be a POJO, spring bean or a script. Typically, you [configure](#) your own service components, but you can also use one of the several standard components included with Mule. For more information, see [Configuring Components](#).

#### Service Configuration

Most configuration happens at the service level. Services can be configured using globally defined endpoints, transformers, and filters, or these can be defined inline. For more information, see [Configuring the Service](#).

#### Service Behavior

When a service receives a message on an inbound endpoint, the [service model](#) (default is SEDA) determines the service's threading and queuing behavior, while the [messaging pattern](#) defines the inbound and outbound message exchange patterns that will be used.

#### Advanced Configuration

You can further configure the service with [security](#) (configured on endpoints), [transactions](#), and [error handling](#).

## Service Messaging Styles

## Mule Messaging Styles

[ Overview ] [ One-way ] [ Request Response ] [ Synchronous ] [ Async Request Response ]

### Overview

Mule ESB can send messages *asynchronously* (each stage of the message is on a different thread) or *synchronously* (after the message is received by the component, it uses a single thread throughout the rest of its lifecycle and supports request-response). You can set the request-response property on the connector, on the endpoint, and implicitly within the transport.

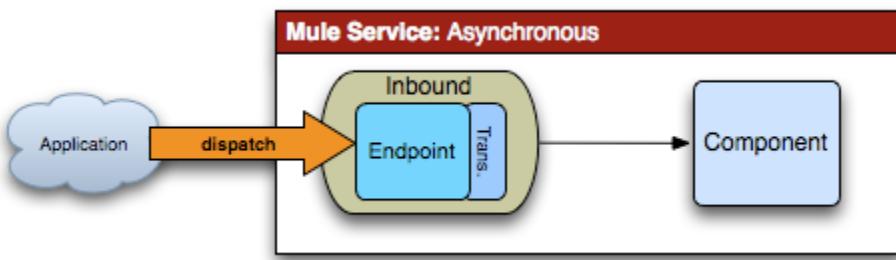
By default, a Mule service uses SEDA, which uses asynchronous, one-way staged queues. One thread is used for the inbound message, another thread is used for processing the message in the service component, and another thread is used for the outbound message. You can configure the message so that the inbound message is on one thread and the remaining stages are on a second thread, or so that all stages are on a single thread.

Mule services also support the request-response messaging style. In this case, there is no outbound router, so the message is sent back to the same endpoint as the inbound endpoint, providing a reply back to the sender.

You can use a mix of request-response and one-way messaging styles throughout Mule services. You can also use a mix of styles for a single service component. For example, a service component can have multiple outbound routers that route to different endpoints depending on filter criteria, and you might want the message to wait for a response in some cases but not in others.

The rest of this page describes the various messaging styles in more detail and how to configure them. It includes reference to the message exchange patterns (MEPs) that each message style supports. For more information on MEPs and Mule, see [MEPs](#).

### One-way



<b>Description</b>	Receives a message and puts it on a SEDA queue. The callee thread returns and the message is processed by the SEDA thread pool. Nothing gets returned from the result of the call.
<b>Error Handling</b>	If an error occurs it is handled by the Mule server. An error endpoint can be used to route errors and the client that initiated the call can listen on the error queue in a separate thread, or have a specific error handling client.
<b>Mule Config</b>	The Mule service must have an asynchronous inbound endpoint.
<b>Equivalent MEPs</b>	In-only
<b>Discussion Points</b>	We have no way of supporting Robust In-only MEP outside of web services (where in Mule you would use Request/Response) and define the MEP in the service contract.

### Example Configuration

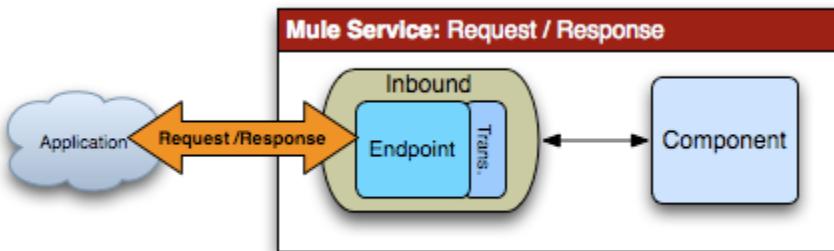
```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <model name="Asynchronous_Message_Pattern">
        <service name="AsynchronousService">
            <inbound>
                <jms:inbound-endpoint queue="test.in" exchange-pattern="one-way"/>
            </inbound>
            <test:component/>
        </service>
    </model>
</mule>

```

## Request Response



<b>Description</b>	Receives a message and the component returns a message. If the component call returns null, then a MuleMessage with a NullPayload is returned. If the call method is void the request message is returned.
<b>Mule Config</b>	The Mule service must have a synchronous inbound endpoint and no outbound endpoint configured. You set an endpoint as synchronous="true". HTTP/S, SSL, TCP, and Servlet endpoints are synchronous by default and do not require this setting.
<b>Error Handling</b>	A response message is always sent back. Clients can check the MuleMessage.getExceptionPayload() to get all information about the server-side error. If an exception originates from the client call, the exception will be thrown.
<b>Equivalent MEPs</b>	In-Out, In-Optional-Out
<b>Discussion Points</b>	In-Optional-Out returns the request message if there is no result from the call.

## Example Configuration

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <model name="Request-Response_Message_Pattern">
        <service name="SynchronousService">
            <inbound>
                <http:inbound-endpoint host="localhost" port="8080" path="/mule/services"
exchange-pattern="request-response"/>
            </inbound>
            <test:component/>
        </service>
    </model>
</mule>

```

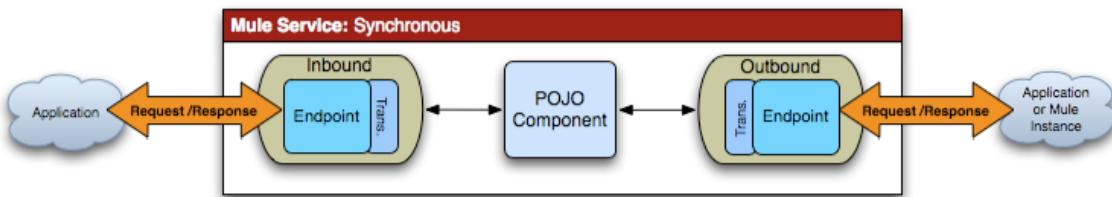
## Synchronous

**NOTE:** As of 3.0, the "synchronous" flag on endpoints has been replaced by the "exchange-pattern" attribute, which has these possible values:

- one-way
- request-response

A message is now processed synchronously if one of the following is true:

- endpoint is request-response
- a transaction is active
- the message property MULE\_FORCE\_SYNC is set to TRUE



<b>Description</b>	Receives a message and the component processes before sending it out on another endpoint. The request happens in the same thread. Mule blocks on the outbound endpoint to wait for a response from the remote application (if applicable) until the <code>responseTimeout</code> threshold is reached. If no response is received, it returns null. The synchronous call must be used if transactions are being used on the inbound endpoint. A synchronous call always returns a result, even if there is an outbound endpoint.
<b>Mule Config</b>	The Mule service must have a request-response inbound endpoint and an outbound endpoint configured. You set an endpoint as request-response using <code>message-exchange="request-response"</code> . HTTP/S, SSL, TCP, and Servlet endpoints are request-response by default and do not require this setting.
<b>Error Handling</b>	A response message is always sent back. Clients can check the <code>MuleMessage.getExceptionPayload()</code> to get all information about the server-side error. If an exception originates from the client call, the exception will be thrown.
<b>Equivalent MEPs</b>	In-Only, In-Optional-Out, In-Out
<b>Discussion Points</b>	Mule always returns the result from the component back to the caller, as well as sending it out via the outbound endpoint.

## Example Configuration

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

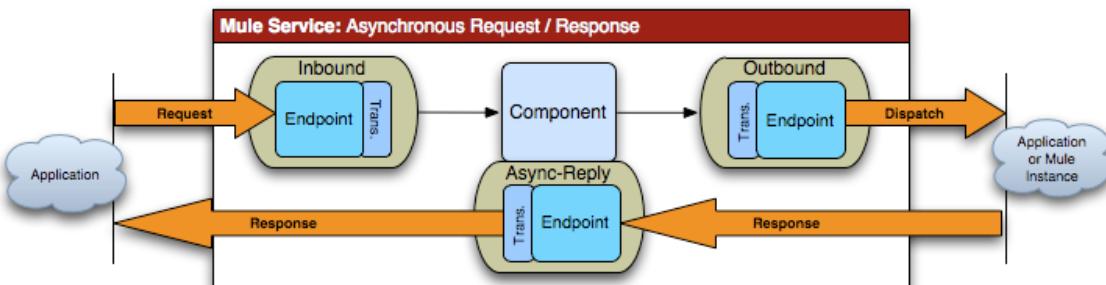
    <model name="Synchronous_Message_Pattern">
        <service name="SynchronousService">
            <inbound>
                <jms:inbound-endpoint queue="test.in" exchange-pattern="request-response"/>
            </inbound>

            <test:component/>

            <outbound>
                <pass-through-router>
                    <jms:outbound-endpoint queue="test.out" exchange-pattern="one-way"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>

```

## Async Request Response



<b>Description</b>	This pattern allows the back-end process to be forked to invoke other services and return a result based on the results of multiple service invocations. The <i>Async Reply Router</i> is used to listen on a <i>Reply To</i> endpoint for results.
<b>Mule Config</b>	Set the reply-to address on the outbound router, and set the <code>&lt;async-reply&gt;</code> element to listen on that reply endpoint. If you also want the caller to get a response, use a synchronous inbound endpoint by setting <code>message-exchange="request-response"</code> .
<b>Error Handling</b>	A response message is always sent back. Clients can check the <code>MuleMessage.getExceptionPayload()</code> to get all information about the server-side error. If an exception originates from the client call, the exception will be thrown.
<b>Equivalent MEPs</b>	In-Out, In-Optional-Out
<b>Discussion Points</b>	None

## Example Configuration

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xmlns:scripting="http://www.mulesoft.org/schema/mule/scripting"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.0/mule-http.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.mulesoft.org/schema/mule/scripting
          http://www.mulesoft.org/schema/mule/scripting/3.0/mule-scripting.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <model name="Async_Request-Response_Message_Pattern">
        <service name="AsyncRequestResponseService">
            <inbound>
                <http:inbound-endpoint host="localhost" port="8080" path="/mule/services"
                    exchange-pattern="request-response"/>
            </inbound>

            <test:component/>

            <outbound>
                <multicasting-router>
                    <jms:outbound-endpoint queue="service1" exchange-pattern="one-way"/>
                    <jms:outbound-endpoint queue="service2" exchange-pattern="one-way"/>
                    <reply-to address="jms://reply.queue"/>
                </multicasting-router>
            </outbound>

            <async-reply timeout="5000">
                <jms:inbound-endpoint queue="reply.queue" exchange-pattern="one-way"/>
                <collection-async-reply-router/>
            </async-reply>
        </service>
    </model>
</mule>

```

Your Rating: 

Results:  0 rates

## Configuring the Service

### Configuring the Service

[ Inbound ] [ Component ] [ Outbound ] [ Asynchronous Reply Router ] [ Exception Strategy ] [ Service Bridge ] [ Service Messaging Style ]

You configure services using a `<service>` element within a `<model>` element. Each `<service>` element represents and configures a Mule ESB service, providing a unique name that identifies it and optionally an initial state that determines whether the service and its endpoints are started when Mule starts (supported values are started, stopped, or paused).

```

<mule>
  <model>
    <service name="GreeterUMO">
      ...
    </service>

    <service name="GreeterUMO2" initialState="stopped" >
      ...
    </service>
  </model>
</mule>

```

Each service can be configured with the following optional elements:

- <description>: Describes the service
- <inbound>: Configures the inbound routers, their endpoints, and inbound transformers
- component: Configures the service component
- <outbound>: Configures the outbound routers, their endpoints, and outbound transformers
- <async-reply>: Configures an async reply router, which is used for asynchronous request-response messaging
- <exception-strategy>: Configures the exception strategy for the service

If you configure more than one of these elements, note that you must configure them in the order shown above. For detailed information on the <service> elements and attributes, see [Service Configuration Reference](#).

Following is a sample service configuration showing these elements:

```

<service name="GreeterUMO">
  <description>Adds some text to the string before passing it on</description>
  <inbound>
    <stdio:inbound-endpoint system="IN">
      <transformer ref="StdinToString"/>
    </stdio:inbound-endpoint>
  </inbound>
  <component class="org.mule.example.hello.Greeter" />
  <outbound>
    <filtering-router>
      <vm:outbound-endpoint path="chitchatter" />
      <payload-type-filter expectedType="org.mule.example.hello.NameString" />
    </filtering-router>
  </outbound>
  <default-service-exception-strategy>
    <vm:outbound-endpoint path="systemErrorHandler" />
  </default-service-exception-strategy>
</service>

```

The following sections describe these elements in more detail.

## **Inbound**

This element is used to configure inbound endpoints and inbound routers. Endpoints are used to receive incoming messages, and inbound routers determine how these messages are routed. Inbound endpoints and routers are configured separately within the <inbound> element.

### **Inbound Endpoints**

Inbound endpoints are used to receive incoming messages. An endpoint is simply a set of instructions indicating which transport and path/address to receive messages from, as well as any transformers, filters, or security that should be applied when receiving the messages. You can configure multiple inbound endpoints, each receiving message from different transports. For more information, see [Configuring Endpoints and Transports Reference](#).

### **Inbound Routers**

The <inbound> element configures inbound routers. Inbound routers control and manipulate messages received by a service before passing them to the service component. Typically, an inbound router is used to filter incoming messages, aggregate a set of incoming messages, or

re-sequence messages when they are received. Inbound routers are also used to register multiple inbound endpoints for a service. You can chain inbound routers together, so that each router must be matched before the message can be passed to the component. You can also configure a [catch-all strategy](#) that is invoked if none of the routers accept the current message.

For more information, see [Inbound Routers](#).

## **Component**

The `<component>` element configures the service component that will be invoked after the inbound message is processed by the inbound routers. If no component is configured, the service acts as a bridge and simply passes messages through to the outbound router.

There are several standard components you can use, such as `<log-component>`, which logs component invocations, outputting the received message as a string, and `<echo-component>`, which extends the log component to log and echo the incoming message. Typically, you will create your own component as a plain old Java object (POJO) and configure it using the `<component>` element.

For more information about component types and their configuration, see [Configuring Components](#). You can also implement new component types in your Mule modules and use them within your configuration. In Mule 2.0, it is now easier to implement and use new non-Java component types and configure them with their own custom component element.

## **Outbound**

The `<outbound>` element configures outbound routers and their endpoints. Because outbound routers are used to determine which endpoints to use for dispatching messages after the component has finished processing them, outbound endpoints are configured on the outbound routers, not directly on the `<outbound>` element. Outbound routers allow the developer to define multiple routing constraints for any given message. You can specify a [catch-all strategy](#) to invoke if none of the routers accept the current message. For more information, see [Outbound Routers](#).

## **Asynchronous Reply Router**

The `<async-reply>` element is used to configure the endpoints and routers that will be used to receive the response in asynchronous request-response scenarios where you must consolidate responses from a remote endpoint before the current service responds via its inbound endpoint. The classic example of this approach is where a request is made and then multiple tasks are executed in parallel. Each task must finish executing and the results processed before a response can be sent back to the requester. For an illustration of asynchronous request-response, click [here](#). For more information, see [Asynchronous Reply Routers](#) and [Configuring Endpoints](#).

## **Exception Strategy**

Exception strategies are used to handle exception conditions when an error occurs during the processing of a message. You can configure exception strategies on services. If no exception strategy is configured, the [DefaultServiceExceptionStrategy](#) is used.

For more information on exception strategies, see [Error Handling](#).

## **Service Bridge**

Service component configuration is optional in Mule 2.x. The default and implicit component used is [PassThroughComponent](#). This component automatically bridges inbound messages to the outbound phase and simply passes messages to the outbound routers. This approach is useful for bridging endpoints if you want to pass a message from one transport to another.



As of Mule 2.0, you no longer need to configure an explicit BridgeComponent.

The following example demonstrates reading a file and send its contents onto a JMS topic.

```

<service name="FileToJmsBridge">
  <inbound>
    <file:inbound-endpoint path="/data/in">
      <file:filename-wildcard-filter pattern="*.txt"/>
    </inbound-endpoint>
  </inbound>

  <!-- No need to configure a component here -->

  <outbound>
    <pass-through-router>
      <jms:outbound-endpoint topic="receivedFiles" />
    </pass-through-router>
  </outbound>
</service>

```

If you want to send a response back to the inbound endpoint, use the chaining router instead of the pass-through router in the outbound endpoint. The inbound endpoint must be synchronous.

```

<service name="HttpProxyService">
  <inbound>
    <inbound-endpoint address="http://localhost:8888" synchronous="true" />
  </inbound>
  <outbound>
    <chaining-router>
      <outbound-endpoint address="http://www.webservicex.net#[header:http.request]" synchronous="true" />
    </chaining-router>
  </outbound>
</service>

```

## Service Model

By default, Mule uses the staged event-driven architecture (SEDA) model. SEDA is an architecture model where applications consist of a network of event-driven stages connected by explicit queues. This architecture allows services to be well-conditioned to load, preventing resources from being overcommitted when demand exceeds service capacity. As a result, SEDA provides an efficient event-based queuing model that maximizes performance and throughput.

See [Models](#) for more information about alternative models and information about how you can implement your own.

## Service Messaging Style

The messaging style determines the message exchange pattern that is to be used on inbound and outbound endpoints and allows endpoints to be configured as synchronous request/response or asynchronous in-only as well as other patterns.

The messaging style is configured on endpoints, allowing multiple styles to be used with the same service. For more information, see [Service Messaging Styles](#).

Your Rating: 

Results:  1 rates

## Models

### About Models

[ [Configuring a Model](#) ] [ [Model Interface](#) ]

A *model* is a grouping of services. A model manages the runtime behavior of the service components that a Mule ESB instance hosts. The manner in which these components are invoked and treated is all encapsulated inside the current Mule model.

### Configuring a Model

To configure a model, you add the `<model>` element to your Mule configuration file. You then add services within the model. For configuration details, see the [Model Configuration Reference](#).

## Model Interface

Every Mule model implements the `Model` interface. This interface represents the behavior of a Mule server and works with the following objects. For a complete list of fields and methods, see in the Model interface `org.mule.api.model.Model`. Also, see `org.mule.model.AbstractModel`, which is an abstract class that implements this interface and is the parent for all models.

Object	Description
Name	A string that refers to the name of the model and is set as an attribute in the <code>&lt;model&gt;</code> element. If no name is given, a unique value is generated automatically.
ExceptionListener	The exception strategy to use for the entire model. The exception strategy is used to handle any exceptions that occur when a component is processing a message. For more information on exception strategies, see <a href="#">Error Handling</a> .
EntryPointResolverSet	A set of classes that implement the <code>EntryPointResolver</code> interface. These will be used to determine the entry point for any hosted component when a message is received. You can configure an entry point resolver using the <code>&lt;abstract-entry-point-resolver&gt;</code> element or configure an entry point resolver set using the <code>&lt;abstract-entry-point-resolver-set&gt;</code> element. For more information on entry point resolvers, see <a href="#">Developing Components</a> .
LifecycleAdapterFactory	Used by the model to create lifecycle adapters that are needed to translate Mule lifecycle events into messages that components registered with the model understand. You configure the lifecycle adapter on the <code>&lt;component&gt;</code> elements, not on the model itself. For more information on lifecycle adapters, see <a href="#">Developing Components</a> .

Your Rating: 

Results:  0 rates

## Using Message Routers

### Using Message Routers

[ [Quick Reference](#) ] [ [Overview](#) ] [ [Frequently Asked Questions](#) ]

Message routers are used within [Using Mule Services](#) to control how messages are sent and received by components in the system. (Within [Using Flows for Service Orchestration](#), the same job is performed by [message processors](#).) See [Overview](#) below for a description of the different types of routers. Click a link in the Quick Reference table below for details on a specific router.

### Quick Reference

Inbound Routers	Outbound Routers	Async-Reply Routers	Catch-all Strategies
No Router	Pass-through	Single	Forwarding
Selective Consumer	Filtering	Collection	Custom Forwarding
Idempotent Receiver	Recipient List Routers	Custom	Logging
Idempotent Secure Hash Receiver	Multicasting		Custom
Collection Aggregator	Chaining		
Message Chunking Aggregator	List Message Splitter		
Custom Correlation Aggregator	Filtering XML Message Splitter		
Correlation Resequencer	Expression Splitter Router		
Forwarding	Message Chunking Router		
WireTap	Exception Based Routers		
Custom	Template Endpoint		
	Round Robin Message Splitter		
	Custom		

### Overview

[Inbound routers](#) control how a service handles incoming messages, such as selectively consuming only those messages that meet specific

criteria or grouping messages together that share a group ID before forwarding them on.

**Outbound routers** control how a message is dispatched after the service has processed it, such as sending it to a list of recipients or splitting up the message and sending the parts to different endpoints.

**Asynchronous reply routers** are used in request/response scenarios where message traffic is triggered by a request and the traffic needs to be consolidated before a response is given. The classic example of this is where a request is made and tasks are executed in parallel. Each task must finish executing and the results processed before a response can be sent back.

**Catch-all strategies** are invoked if no routing path can be found for the current message. An inbound or outbound endpoint can be associated with a catch-all strategy so that any orphaned messages can be caught and routed to a common location.

Filters provide the logic used to invoke a particular router. Filters can be combined using the logic filters AndFilter, OrFilter, and NotFilter. Not all routers need to use filters, but all routers support them. See [Using Filters](#) for complete information.

Mule ESB provides flexible message routing support for your components. Routing features are based on the enterprise routing requirements described in [EIP](#). For information on how your Java or Script components can orchestrate messages, see [Component Bindings](#).

## Frequently Asked Questions

### When do I need an outbound router vs. simply getting a reply?

In simple scenarios, a service receives a request on a synchronous inbound endpoint, processes the request, and then sends it back to the caller as a reply. This is the **request-response message style**. For example, if a user enters a value in an HTML form, and you want to transform that value and display the results in the same page, you can simply configure a synchronous inbound endpoint on the service that does the transformation. This scenario does not use an outbound router.

If you need to pass the message to a second service for additional processing, you configure an outbound router on the first service to pass the message to the second service. You configure a synchronous inbound endpoint on the second service so that it passes the message back to the first service, which then sends it back to the caller as a reply. This is the **synchronous message style**.

In the most complex scenario, you can enable request-response messaging and allow the back-end process to be forked to invoke other services and return a result based on the results of multiple service invocations. This is the **asynchronous request response message style**.

See [Service Messaging Styles](#) for more information.

### How do I create a nested router that calls a web service with multiple methods without having to hardcode the method name in the outbound endpoint?

The **template endpoint router** allows endpoints to be altered at runtime based on properties set on the current message or fallback values set on the endpoint properties.

Your Rating: 

Results:  3 rates

## Inbound Routers

### Inbound Routers

[ [Overview](#) ] [ [No Router](#) ] [ [Selective Consumer](#) ] [ [Idempotent Message Filter](#) ] [ [Idempotent Secure Hash Message Filter](#) ] [ [Collection Aggregator](#) ] [ [Message Chunking Aggregator](#) ] [ [Custom Correlation Aggregator](#) ] [ [Correlation Resequencer](#) ] [ [Forwarding Router](#) ] [ [Wiretap Router](#) ] [ [Custom Inbound Router](#) ]

#### Overview

Inbound routers control and manipulate messages received by a service before passing them to the service component. Typically, an inbound router is used to filter incoming messages, aggregate a set of incoming messages, or re-sequence messages when they are received. Inbound routers are also used to register multiple inbound endpoints for a service. You can chain inbound routers together, so that each router must be matched before the message can be passed to the component. You can also configure a **catch-all strategy** that is invoked if none of the routers accept the current message.

Inbound routers are different from outbound routers in that the endpoint is already known (as the message has already been received), so the purpose of the router is to control how messages are given to the component.

All inbound routers are configured on a service within the `<inbound>` element. If no inbound routers are configured, by default an `InboundPassThroughRouter` is used to simply pass the incoming message to the component.

#### Matching Only the First Router

By default, a message must match and be processed by **all** inbound routers in a service before it is passed to the service component. If you want to configure the service so that the message is processed only by the first router whose conditions it matches, you set the `matchAll` attribute on the `<inbound>` element to false.



This behavior is new as of Mule 2.0. Previously, the message was processed only by the first matching router by default.

### Inbound Example

```
<inbound>
    <stdio:inbound-endpoint system="IN" />
    <catch-all-strategy>
        <jms:outbound-endpoint queue="failure.queue"/>
    </catch-all-strategy>
    <selective-consumer-router>
        <mulexml:xpath-filter pattern="(msg/header/resultcode)='success'" />
    </selective-consumer-router>
</inbound>
```

This example uses a selective consumer router that will accept a message if a 'resultcode' element has a value of 'success'. If the message matches this filter's criteria, the message is passed to the component. If the message does not match, the catch-all strategy is invoked, which sends the message via its configured endpoint, in this case a JMS queue called 'failure.queue'.

The rest of this page describes the inbound routers you can use. For more detailed information on inbound router configuration elements and attributes, see the [Inbound Router Configuration Reference](#).

### No Router

If no router is defined on the inbound, all messages received via the endpoints will be processed by the service component.

```
<inbound>
    <jms:inbound-endpoint queue="foo.bar"/>
    <vm:inbound-endpoint path="foo.bar.local"/>
</inbound>
```

### Selective Consumer

A selective consumer is an inbound router that can apply one or more filters to the incoming message. If the filters match, the message is forwarded to the component. Otherwise, the message is forwarded to the catch-all strategy on the router. If no catch-all is configured, the message is ignored and a warning is logged.

Configuration for this router is as follows:

```
<inbound>
    <selective-consumer-router>
        <mulexml:xpath-filter expression="msg/header/resultcode = 'success'" />
    </selective-consumer-router>
    <forwarding-catch-all-strategy>
        <jms:endpoint topic="error.topic"/>
    </forwarding-catch-all-strategy>
</inbound>
```

For information on using filters with this router, see [Using Filters](#). Note that by default the filter is applied to the message after the inbound transformers are applied. If you need to execute filters on the message without applying any transformation, you can set the `transformFirst` property on this router to control whether transformations are applied.

```

<inbound>
  <forwarding-catch-all-strategy>
    <jms:endpoint topic="error.topic"/>
  </forwarding-catch-all-strategy>
  <selective-consumer-router transformFirst="false">
    <mulexml:jxpath-filter expression="msg/header/resultcode = 'success'"/>
  </selective-consumer-router>
</inbound>

```

### Idempotent Message Filter

An idempotent filter ensures that only unique messages are received by a service by checking the unique message ID of the incoming message. The ID can be generated from the message using an expression defined in the `idExpression` attribute. By default, the expression used is `#[message:id]`, which means the underlying endpoint must support unique message IDs for this to work. Otherwise, a `UniqueIdNotSupportedException` is thrown.

There is a simple idempotent filter implementation provided at [org.mule.routers.IdempotentMessageFilter](#). The default implementation uses a simple file-based mechanism for storing message IDs, but you can extend this class to store the IDs in a database instead by implementing the `ObjectStore` interface.

Configuration for this router is as follows:

```

<inbound>
  <idempotent-receiver-router idExpression="#[message:id]-#[header:foo]">
    <simple-text-file-store directory="../idempotent"/>
  </idempotent-receiver-router>
</inbound>

```

The optional `idExpression` attribute determines what should be used as the unique message ID. If this attribute is not used, `#[message:id]` is used by default.

The nested element shown above configures the location where the received message IDs are stored. In this example, they are stored to disk so that the router can remember state between restarts. If the `directory` attribute is not specified, the default value used is  `${mule.working.dir}/objectstore` where `mule.working.dir` is the working directory configured for the Mule instance.

If no store is configured, the `InMemoryObjectStore` is used by default.

### Idempotent Secure Hash Message Filter

This filter ensures that only unique messages are received by a service by calculating the hash of the message itself using a message digest algorithm. This approach provides a value with an infinitesimally small chance of a collision and can be used to filter message duplicates. Note that the hash is calculated over the entire byte array representing the message, so any leading or trailing spaces or extraneous bytes (like padding) can produce different hash values for the same semantic message content. Therefore, you should ensure that messages do not contain extraneous bytes. This router is useful when the message does not support unique identifiers.

Configuration for this filter is as follows:

```

<inbound>
  <idempotent-secure-hash-receiver-router messageDigestAlgorithm="SHA256">
    <simple-text-file-store directory="../idempotent"/>
  </idempotent-secure-hash-receiver-router>
</inbound>

```

Idempotent Secure Hash Message Filter also uses object stores, which are configured the same way as the Idempotent Message Filter. The optional `messageDigestAlgorithm` attribute determines the hashing algorithm that will be used. If this attribute is not specified, the default algorithm SHA-256 is used.

### Collection Aggregator

The Collection Aggregator groups incoming messages that have matching group IDs before forwarding them. The group ID can come from the

correlation ID or another property that links messages together.

You can specify the `timeout` attribute to determine how long the router waits in milliseconds for messages to complete the group. By default, if the expected messages are not received by the `timeout` time, an exception is thrown and the messages are not forwarded. As of Mule 2.2, you can set the `failOnTimeout` attribute to `false` to prevent the exception from being thrown and simply forward whatever messages have been received so far.

The aggregator is based on the [Selective Consumer](#), so you can also apply filters to the messages. Configuration for this router is as follows:

```
<inbound>
  <collection-aggregator-router timeout="6000" failOnTimeout="false">
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </collection-aggregator-router>
</inbound>
```

### Message Chunking Aggregator

After an outbound router such as the [List Message Splitter](#) splits a message into parts, the message chunking aggregator router reassembles those parts back into a single message. The aggregator uses the correlation ID, which is set by the outbound router, to identify which parts belong to the same message. This aggregator is based on the [Selective Consumer](#), so filters can also be applied to messages.

Configuration for this router is as follows:

```
<inbound>
  <message-chunking-aggregator-router>
    <expression-message-info-mapping correlationIdExpression="#[header:correlation]"/>
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </message-chunking-aggregator-router>
</inbound>
```

The optional `expression-message-info-mapping` element allows you to identify the correlation ID in the message using an expression. If this element is not specified, `MuleMessage.getCorrelationId()` is used.

The Message Chunking aggregator also accepts the `timeout` and (as of Mule 2.2) `failOnTimeout` attributes as described under [Collection Aggregator](#).

### Custom Correlation Aggregator

This router is used to configure a custom message aggregator. Mule provides an abstract implementation that has a template method that performs the message aggregation. A common use of the aggregator router is to combine the results of multiple requests such as "ask this set of vendors for the best price of X".

The aggregator is based on the [Selective Consumer](#), so you can also apply filters to messages. It also accepts the `timeout` and (as of Mule 2.2) `failOnTimeout` attributes as described under [Collection Aggregator](#).

Configuration for this router is as follows:

```
<inbound>
  <custom-correlation-aggregator-router class="org.mule.CustomAggregator">
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </custom-correlation-aggregator-router>
</inbound>
```

There is an [AbstractEventAggregator](#) that provides a thread-safe implementation for custom aggregators, which you can use to write a custom aggregator router. For example, the [Loan Broker](#) examples included in the Mule distribution use a custom `BankQuotesInboundAggregator` router to aggregate bank quotes.

### Correlation Resequencer

The Correlation Resequencer Router will hold back a group of messages and resequence them using the `messages.correlationSequence` property. A `java.util.Comparator` is used to sort the messages. This router is based on the [Selective Consumer](#), which means that filters

can be applied to the incoming messages. It also accepts the `timeout` and (as of Mule 2.2) `failOnTimeout` attributes as described under [Collection Aggregator](#).

```
<inbound>
  <correlation-resequencer-router>
    <mulexml:jxpath-filter expression="msg/header/resultcode = 'success'" />
  </correlation-resequencer-router>
</inbound>
```

### Forwarding Router

This router allows messages to be forwarded to an outbound router without first being processed by a component. It essentially acts as a bridge between an inbound and an outbound endpoint. This is useful in situations where the developer does not need to execute any logic on the inbound message but does need to forward it on to a component residing on another destination (such as a remote Mule node or application) over the network.

Configuration for this router is as follows:

```
<service name="FileReader">
  <inbound>
    <file:inbound-endpoint path="/temp/myfiles/in" />
    <forwarding-router/>
  </inbound>
  <echo-component/>
  <outbound>
    <tcp:outbound-endpoint host="192.168.0.6" port="12345">
      <object-to-byte-array-transformer/>
    </tcp:outbound-endpoint>
  </outbound>
</service>
```

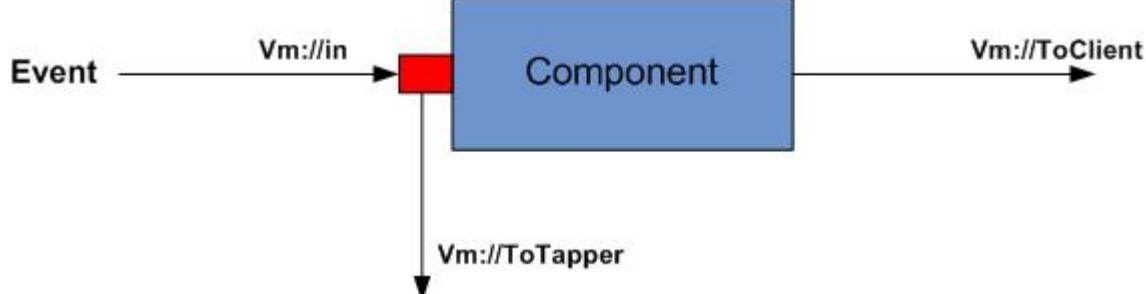
When a file becomes available on the local file system, an event is triggered that creates the message, which is then automatically forwarded via TCP to 192.168.0.6. Notice that there is an `outboundTransformer` configured. This will be used to transform the message's payload before it is dispatched over TCP. There is an echo component configured, but when the forwarding consumer is used, the component invocation is skipped, and the message is forwarded directly to the the outbound router(s).

Configuring the service as a bridge is recommended for most forwarding scenarios. However, if you need to selectively forward only some events while others are processed by the component, you will need to use this router.

The Forwarding router extends the [Selective Consumer](#), so you can configure filters on this router.

### Wiretap Router

The WireTap inbound router allows you to route certain messages to a different endpoint as well as to the component.



To copy all messages to a specific component, you configure an outbound endpoint on the WireTap router:

```

<inbound>
    <vm:inbound endpoint path="FromUser" />
    <wire-tap-router>
        <vm:outbound-endpoint path="tapped.channel" />
    </wire-tap-router>
</inbound>

```

In the following scenario, no component is specified, so all data from the inbound VM channel is copied to the outbound endpoint using implicit bridging. However, let's assume you want to forward some of the data to another component called WireTapReceiver based on a filter. For the sake of illustration, this component simply prepends the message with "INTERCEPTED:" before sending it to the FromTapper VM channel. The code for the WireTapReceiver component is as follows:

```

public class WireTapReceiver {

    public String handleInterceptedData (String aMessage) {
        //Just Prepend the message with a label
        return "\nINTERCEPTED: "+aMessage;
    }
}

```

Following is the configuration of the Mule services:

```

<model name="default">
    <service name="StdComp">
        <inbound>
            <vm:inbound-endpoint path="In" />
            <wire-tap-router>
                <vm:outbound-endpoint path="ToTapper" />
            </wire-tap-router>
        </inbound>
        <outbound>
            <pass-through-router>
                <vm:outbound-endpoint path="ToClient" />
            </pass-through-router>
        </outbound>
    </service>
    <service name="wiretapper">
        <inbound>
            <vm:inbound-endpoint path="ToTapper" />
        </inbound>
        <component class="org.myclass.WireTapReceiver"/>
        <outbound>
            <pass-through-router>
                <vm:outbound-endpoint path="FromTapper" />
            </pass-through-router>
        </outbound>
    </service>
</model>

```

Note: Mule uses a separate dispatcher thread for the wiretap endpoint.

#### **Using Filters with the WireTap Router**

The WireTap router is useful both with and without filtering. If filtered, it can be used to record or take note of particular messages or to copy messages that require additional processing to a different component. If filters aren't used, you can make a backup copy of all messages received by a component. The behavior here is similar to that of an interceptor, but interceptors can alter the message flow by preventing the message from reaching the component. WireTap routers cannot alter message flow but just copy on demand.

In the previous example, the StdComp service receives messages from the In endpoint, and the router passes the message to the component and copies it to the vm:/ToTapper endpoint. The WireTapper component listens on this channel and forwards the message, after processing, to the FromTapper endpoint.

The WireTap router is based on the [Selective Consumer](#) router, so it can take any [filters](#) supported by [SelectiveConsumer](#). In this example,

only messages that match the filter expression are copied to the `vm://ToTapper` endpoint.

```
<wire-tap-router>
    <wildcard-filter pattern="the quick brown*" />
    <vm:outbound-endpoint path="tapped.channel" />
</wire-tap-router>
```

### Using Multiple WireTap Routers

You can have multiple WireTap routers for the same service:

```
<inbound>
    <endpoint address="vm://In" />
    <wire-tap-router>
        <wildcard-filter pattern="the quick brown*" />
        <vm:outbound-endpoint path="ToTapper" />
    </wire-tap-router>
    <wire-tap-router>
        <wildcard-filter pattern="the slow green*" />
        <vm:outbound-endpoint path="ToOtherTapper" />
    </wire-tap-router>
</inbound>
```

In this example, input is passed to the component and also copied to one of two destinations depending on the filter.

### Method Invocation in the Wire-tapped Component

You can invoke your service with a specific method. For example, if your inbound endpoint is not `vm://In` but `axis\:\:\http://localhost\:8080/services`, or if your component `StdComp` is a customized component with a method `foo()`, you can invoke the web service and its method `foo()` via the following endpoint:

```
http\://localhost\:8080/services/StdComp?method=foo&param=bar
```

When this message is wire-tapped to the receiving component, Mule might fail with an exception if the receiving component does not have the method `foo()`. To avoid this problem and to ensure that the desired method is invoked, you overwrite the method of the message by specifying `?method=methodName`, or by specifying `?method=` so that the `onCall()` method will be called instead. For example:

```
<wire-tap-router>
    <outbound-endpoint address="vm://inboundEndpoint3?connector=vm2" />
</wire-tap-router>
...
<service name="serviceComponent3">
    <inbound>
        <inbound-endpoint address="vm://inboundEndpoint3?connector=vm2&method=" synchronous="false" />
        </inbound>
        <component class="org.mule.components.simple.LogComponent" />
    </service>
```

### Additional WireTap Router Features

The WireTap router supports the following additional features:

- Transactions are supported, so the forwarding of messages can either start or join a transaction provided that the endpoint supports transactions.
- Reply-To can be used to route replies from this endpoint.

### Custom Inbound Router

You can configure custom inbound routers by specifying the custom router class on the `<custom-inbound-router>` element and by using Spring properties. Optionally, you can also configure an outbound endpoint in case this is needed for implementing a custom wiretap router for example.

Configuration for this router is as follows:

```
<inbound>
    <custom-inbound-router class="org.my.CustomInboundRouter">
        <mulexml:jxpath-filter expression="msg/header/resultcode = 'success' />
        <spring:properties>
            <spring:property key="key1" value="value1"/>
            <spring:property key="key2" value="value2"/>
        </spring:properties>
        <vm:outbound-endpoint path="out"/>
    </custom-inbound-router>
</inbound>
```

Your Rating: 

Results:  3 rates

## Outbound Routers

### ***Outbound Routers***

[ Overview ] [ Pass-through Router ] [ Filtering Router ] [ Recipient List Routers ] [ Multicasting Router ] [ Chaining Router ] [ List Message Splitter ] [ Filtering XML Message Splitter ] [ Expression Splitter Router ] [ Round Robin Message Splitter ] [ Message Chunking Outbound Router ] [ Exception Based Routers ] [ Template Endpoint Router ] [ Custom Outbound Router ]

#### Overview

After a message has been processed by a component, you use outbound routers to determine where to send the message next. You configure the outbound endpoints on the outbound router, not on the `<outbound>` element. Outbound routers allow you to define multiple routing constraints for any given message. You can specify a [catch-all strategy](#) to invoke if none of the routers accept the current message.

#### ***Matching All Routers***

By default, a message is processed only by the first outbound router whose conditions it matches. If you want the message to be processed by all the outbound routers, you can set the `matchAll` attribute to true. For example, assume you always want to send a confirmation of a deposit back to the original depositor. Also assume that if the deposit was above \$100,000, you want to send a notification message to the 'high net worth client manager' for possible follow-up. In this case, you would set the `matchAll` attribute on the `<outbound>` definition as follows:

```
<outbound matchAll="true">
    <filtering-router>
        <endpoint address="jms://deposit.queue"/>
    </filtering-router>
    <filtering-router>
        <jms:outbound-endpoint queue="large.deposit.queue"/>
        <mulexml:jxpath-filter expression="deposit/amount >= 100000"/>
    </filtering-router>
</outbound>
```

In this example, the message will always match the first router because there is no filter on it. Additionally, the message will match the second router if the deposit amount is  $\geq \$100000$ , in which case both routers will have been invoked.

#### ***Outbound Example***

```

<outbound>
    <catch-all-strategy>
        <jms:outbound-endpoint queue="default.queue" />
    </catch-all-strategy>
    <filtering-router>
        <smtp:outbound-endpoint to="exceptions@muleumo.org" subject="Exception!" from="mule@mycompany.com!">
            <transformer ref="ExceptionToEmail"/>
        </smtp:outbound-endpoint>
        <payload-type-filter expectedType="java.lang.Exception" />
    </filtering-router>
    <filtering-router>
        <vm:endpoint path="my.component" />
        <and-filter>
            <payload-type-filter expectedType="java.lang.String" />
            <regex-filter pattern="the quick brown (.*)"/>
        </and-filter>
    </filtering-router>
</outbound>

```

The following sections describe each Mule outbound router and how to configure them. Outbound routers can be more complex to configure, as they allow different routing paths that can be selected depending on the logic defined in one or more filters. For more detailed information on outbound router configuration elements and attributes, see the [Outbound Router Configuration Reference](#).

#### **ReplyTo**

All outbound routers can have a `reply-to endpoint` that defines where the message should be routed after the recipient of the message has finished processing it.

In the following example, the first service sets up the routing, including passing the message through to an outbound endpoint and specifying the reply-to endpoint that will receive the results from the second service.

```

<service name="Requester">
    <inbound>
        <vm:inbound-endpoint path="IN" synchronous="true" />
    </inbound>
    <outbound>
        <pass-through-router>
            <jms:outbound-endpoint queue="RequestQueue" synchronous="true" />
            <reply-to address="jms://ReplyQueue" />
        </pass-through-router>
    </outbound>
</service>

<service name="Replier">
    <inbound>
        <jms:inbound-endpoint queue="RequestQueue" synchronous="true" />
    </inbound>
    <echo-component/>
</service>

```

The `<reply-to>` endpoint is used only by the very next service in the flow. For example, assume there are three services in the flow, and the first service has an outbound router with a reply-to endpoint. The second service in the flow will send the results of its invocation to that reply-to endpoint as well as passing the message along to the third service. The third service, however, processes the message as normal without sending its results to the reply-to endpoint.

The `<reply-to>` endpoint can be any valid [Mule endpoint URI](#). For information on which transports support reply-to, see [Transports Reference](#).

Note that adding a reply-to endpoint makes the outbound endpoint asynchronous, so a response is not sent back to the caller, just to the reply-to endpoint. If you want to send a response back to the caller, you can add an [asynchronous reply router](#) in addition to the reply-to.

#### **Pass-through Router**

This router always matches and simply sends or dispatches message via the one endpoint it has configured.

Configuration for this router is as follows:

## 2.0:

```
<outbound>
  <outbound-pass-through-router>
    <smtp:outbound-endpoint to="ross@muleumo.org" />
  </outbound-pass-through-router>
</outbound>
```

## 2.1 and later:

```
<outbound>
  <pass-through-router>
    <smtp:outbound-endpoint to="ross@muleumo.org" />
  </pass-through-router>
</outbound>
```

You can also use the pass-through router to perform protocol bridging to another outbound endpoint. For example:

```
<service name="HttpProxyService">
  <inbound>
    <!-- WSDL URL: http://localhost:8888/stockquote.asmx?wsdl -->
    <inbound-endpoint address="http://localhost:8888" synchronous="true" />
  </inbound>
  <outbound>
    <pass-through-router>
      <outbound-endpoint address="http://www.webservicex.net#[header:http.request]" synchronous="true" />
    </pass-through-router>
  </outbound>
</service>
```

By default, the pass-through router does not return a response. If you do need to return a response, set synchronous to true on the outbound endpoint, or use the chaining router, which returns a response by default.

## Filtering Router

This router uses filters to determine whether this router will be used (see [Filters](#) below). The filtering router specifies only one endpoint to which messages are routed if the filter conditions are met. If you need to specify more than one endpoint in a router, use a different outbound router and set filters on its individual endpoints.

Configuration for this router is as follows:

```

<outbound>
  <forwarding-catch-all-strategy>
    <jms:outbound-endpoint queue="error.queue" />
  </forwarding-catch-all-strategy>
  <filtering-router>
    <smtp:outbound-endpoint to="ross@muleumo.org" />
    <payload-type-filter expectedType="java.lang.Exception" />
  </filtering-router>
  <filtering-router>
    <jms:outbound-endpoint queue="string.queue" />
    <and-filter>
      <payload-type-filter expectedType="java.lang.String" />
      <regex-filter pattern="the quick brown (.*)" />
    </and-filter>
  </filtering-router>
</outbound>

```

The filter is applied to the message first, and then the transformers are applied. If you need to transform the message before the filter is applied, you can set a transformer on this router that will be applied to the message before the filter is applied.

```

<outbound>
  <filtering-router>
    <smtp:outbound-endpoint to="ross@muleumo.org" />
    <payload-type-filter expectedType="java.lang.Exception" />
    <transformer ref="aTransformer" />
  </filtering-router>
</outbound>

```

## Recipient List Routers

The recipient list routers can be used to send the same message to multiple endpoints over a single endpoint, or to implement routing-slip behavior where the next destination for the message is determined from message properties or the payload. Mule provides an abstract recipient list implementation [org.mule.routing.outbound.AbstractRecipientList](#), which provides a thread-safe base for specialized implementations.

Mule provides a static recipient list router that takes a configured list of endpoints from the current message or from a statically declared list on the endpoint.

Configuration for this router is as follows:

```

<outbound>
  <static-recipient-list-router>
    <payload-type-filter expectedType="javax.jms.Message" />
    <recipients>
      <spring:value>jms://orders.queue</spring:value>
      <spring:value>jms://tracking.queue</spring:value>
    </recipients>
  </static-recipient-list-router>
</outbound>

```

Mule also provides an expression recipient list router, which allows you to create a static list as well as use an expression to determine the list.

```

<outbound>
  <expression-recipient-list-router evaluator="headers-list" expression=
"recipient1,recipient2,recipient3" />
</outbound>

```

## Multicasting Router

The Multicasting router can be used to send the same message over multiple endpoints. When using this router, be sure to configure the correct transformers on the endpoints to handle the message source type.

Configuration for this router is as follows:

```
<outbound>
  <multicasting-router>
    <jms:endpoint queue="test.queue" transformer-refs="StringToJmsMessage"/>
    <http:endpoint host="10.192.111.11" transformer-refs="StringToHttpClientRequest"/>
    <tcp:endpoint host="10.192.111.12" transformer-refs="StringToByteArray"/>
    <payload-type-filter expectedType="java.lang.String"/>
  </multicasting-router>
</outbound>
```

If you set the endpoints to synchronous, each endpoint processes the original message sequentially, and the results are collected into a single reply and then sent to the reply-to address. For example:

```
<outbound>
  <multicasting-router>
    <jms:endpoint queue="test.queue" transformer-refs="StringToJmsMessage" synchronous="true"/>
    <http:endpoint host="10.192.111.11" transformer-refs="StringToHttpClientRequest" synchronous="true"/>
    <tcp:endpoint host="10.192.111.12" transformer-refs="StringToByteArray" synchronous="true"/>
    <payload-type-filter expectedType="java.lang.String"/>
    <reply-to address="jms:reply.queue"/>
  </multicasting-router>
</outbound>
```

If you want the endpoints to process the message simultaneously and send back individual replies, set the endpoints to asynchronous (`synchronous="false"`). An individual reply will be returned from each endpoint to the caller. If you want to collect the asynchronous responses before sending the reply, use asynchronous endpoints, set a `replyTo` for the multicasting router, and then use the [Collection Asynchronous Reply Router](#). For example:

```
<outbound>
  <multicasting-router>
    <jms:endpoint queue="test.queue" transformer-refs="StringToJmsMessage" synchronous="false"/>
    <http:endpoint host="10.192.111.11" transformer-refs="StringToHttpClientRequest" synchronous="false"/>
    <tcp:endpoint host="10.192.111.12" transformer-refs="StringToByteArray" synchronous="false"/>
    <payload-type-filter expectedType="java.lang.String"/>
    <reply-to address="jms:reply.queue"/>
  </multicasting-router>
</outbound>
<async-reply failOnTimeout="false" timeout="2000">
  <jms:inbound-endpoint queue="reply.queue"/>
  <collection-async-reply-router/>
</async-reply>
```

## Chaining Router

The chaining router can be used to send the message through multiple endpoints using the result of the first invocation as the input for the next. For example, this can be useful where you want to send the results of a synchronous request-response invocation such as a Web service call to a JMS queue. Endpoint transformers can be used to transform the message to the format the next endpoint requires.

Configuration for this router is as follows:

```

<outbound>
  <chaining-router>
    <axis:outbound-endpoint address="http://localhost:8081/services/xyz?method=getSomething" />
    <jms:outbound-endpoint queue="something.queue">
      <transformer ref="SomethingToJmsMessage" />
    </jms:outbound-endpoint>
  </chaining-router>
</outbound>

```

The endpoints specified in the chaining router are always synchronous and pass the message along in a single thread. The exception is the last endpoint, on which you can set `synchronous="false"` if you do not want to send a response to the caller.

Note that if any of the endpoints in the chain return null, the router exits.

### List Message Splitter

A message splitter can be used to break down an outgoing message into parts and dispatch those parts over different endpoints configured on the router. The List Message Splitter accepts a list of objects that will be routed to different endpoints. The actual endpoint used for each object in the list is determined by a filter configured on the endpoint itself. If the endpoint's filter accepts the object, the endpoint will be used to route the object.

By default the `AbstractMessageSplitter` sets a correlation ID and correlation sequence on the outbound messages so that inbound routers such as the `Collection Aggregator` or `Correlation Resequencer` are able to resequence or combine the split messages.

The router configuration below expects the message payload to be a `java.util.List` and will route objects in the list that are of type `com.foo.Order`, `com.foo.Item`, and `com.foo.Customer`. The router will allow any number and combination of these objects.

Configuration for this router is as follows:

```

<outbound>
  <list-message-splitter-router>
    <jms:outbound-endpoint queue="order.queue">
      <payload-type-filter expectedType="com.foo.Order" />
    </jms:outbound-endpoint>
    <jms:outbound-endpoint queue="item.queue">
      <payload-type-filter expectedType="com.foo.Item" />
    </jms:outbound-endpoint>
    <jms:outbound-endpoint queue="customer.queue">
      <payload-type-filter expectedType="com.foo.Customer" />
    </jms:outbound-endpoint>
    <payload-type-filter expectedType="java.util.List" />
  </list-message-splitter-router>
</outbound>

```

Note that there is also a filter on the router itself that ensures that the message payload received is of type `java.util.List`. If there are objects in the list that do not match any of the endpoint filters, a warning is written to the log and processing continues. To route any non-matching object types to another endpoint, add the endpoint at the end of the list without a filter.

### Filtering XML Message Splitter

This router is similar to the List Message Splitter but operates on XML documents. Supported payload types are:

- `org.dom4j.Document` objects
- `byte[]`
- `java.lang.String`

If no match is found, it is ignored and logged at the `WARN` level.

The router splits the payload into nodes based on the `splitExpression` property. The actual endpoint used for each object in the list is determined by a filter configured on the endpoint itself. If the endpoint's filter accepts the object, the endpoint will be used to route the object. Each part returned is actually returned as a new DOM4J document.

The router can optionally perform a validation against an external XML schema document. To perform the validation, set `externalSchemaLocation` to the XSD file in your classpath. Setting this property overrides whatever schema document you declare in the

XML header.

By default, the router fails if none of the endpoint filters match the payload. To prevent the router from failing in this case, you can set the `failIfNoMatch` attribute to `false`.

Configuration for this router is as follows:

```
<outbound>
    <mulexml:filter-based-splitter splitExpression="root/nodes" validateSchema="true"
externalSchemaLocation="/com/example/TheSchema.xsd">
        <vm:outbound-endpoint path="order">
            <payload-type-filter expectedType="com.foo.Order"/>
        </vm:outbound-endpoint>
        <vm:outbound-endpoint path="item">
            <payload-type-filter expectedType="com.foo.Item"/>
        </vm:outbound-endpoint>
        <vm:outbound-endpoint path="customer">
            <payload-type-filter expectedType="com.foo.Customer"/>
        </vm:outbound-endpoint>
        <payload-type-filter expectedType="org.dom4j.Document"/>
    </mulexml:filter-based-splitter>
</outbound>
```

### Expression Splitter Router

This router is similar to the list message splitter router, but it splits the message based on an [expression](#). The expression must return one or more message parts to be effective.

```
<outbound>
    <expression-splitter-router evaluator="xpath" expression="/mule:mule/mule:model/mule:service"
disableRoundRobin="true" failIfNoMatch="false">
        <outbound-endpoint ref="service1">
            <expression-filter evaluator="xpath" expression="/mule:service/@name = 'service splitter'"/>
        </outbound-endpoint>
        <outbound-endpoint ref="service2">
            <expression-filter evaluator="xpath" expression="/mule:service/@name = 'round robin
deterministic'"/>
        </outbound-endpoint>
    </expression-splitter-router>
</outbound>
```

### Round Robin Message Splitter

The round robin message splitter will split a DOM4J document into nodes based on the `splitExpression` property. It will then send these document fragments to the list of endpoints specified in a round-robin fashion. Optionally, you can specify a namespaces property map that contain prefix/namespace mappings.

For instance, the following fragment will route the `"/a:orders/a:order"` nodes inside the document to the `robin1` and `robin2` endpoints.

```
<outbound>
    <mxml:round-robin-splitter splitExpression="/a:orders/a:order" deterministic="false">
        <outbound-endpoint ref="robin1"/>
        <outbound-endpoint ref="robin2"/>
        <mxml:namespace prefix="a" uri="http://acme.com"/>
    </mxml:round-robin-splitter>
</outbound>
```

The router can optionally perform a validation against an external XML schema document. To perform the validation, set `externalSchemaLocation` to the XSD file in your classpath. Setting this property overrides whatever schema document you declare in the XML header.

```

<outbound>
    <mxml:round-robin-splitter splitExpression="/a:orders/a:order" deterministic="false"
externalSchemaLocation="mySchema.xsd" validateSchema="true">
        <outbound-endpoint ref="robin1"/>
        <outbound-endpoint ref="robin2"/>
        <mxml:namespace prefix="a" uri="http://acme.com"/>
    </mxml:round-robin-splitter>
</outbound>

```

### Message Chunking Outbound Router

This routing pattern allows you to split a single message into a number of fixed-length messages that will all be routed to the same endpoint. It will split the message up into a number of smaller chunks according to the `messageSize` attribute that you configure for the router. If you do not configure a `messageSize`, or if it has a value of zero, the message will not be split up and the entire message will be routed to the destination endpoint as is. The router splits up the message by first converting it to a byte array and then splitting this array into chunks. If the message cannot be converted into a byte array, a `RoutingException` is raised.

A message chunking router is useful if you have bandwidth problems (or size limitations) when using a particular transport. If you want to be able to route different segments of the original message to different endpoints, consider using the [List Message Splitter](#) or [Filtering XML Message Splitter](#) router instead.

To put the chunked items back together again, you can use the [Message Chunking Aggregator](#) as the inbound router on the next service.

### Sample Configuration

```

<service name="chunkingService">
    <inbound>
        <vm:inbound-endpoint path="fromClient"/>
    </inbound>
    <outbound>
        <message-chunking-router messageSize="4">
            <vm:outbound-endpoint path="toClient"/>
        </message-chunking-router>
    </outbound>
</service>

```

In the example above, any data received on the `vm fromClient` endpoint is chunked into messages four bytes long before being sent along the `vm toClient` endpoint. If we sent "The quick brown fox jumped over the lazy dog" to this service, anything listening on the `vm toClient` endpoint would receive the following messages (the spaces have been replaced with underscores for better legibility):

Message #	Contents
1	The_
2	quic
3	k_br
4	own_
5	fox_
6	jump
7	ed_o
8	ver_
9	the_
10	lazy
11	_dog

## Exception Based Routers

The Exception Based router can be used to send a message over an endpoint by selecting the first endpoint that can connect to the transport. This can be useful for setting up retries. When the first endpoint fails, the second will be invoked, and if that fails, it will try the next endpoint. Note that this router overrides the endpoint mode to synchronous while looking for a successful send and will resort to using the endpoint's mode for the last item in the list.

Configuration for this router is as follows:

```
<outbound>
  <exception-based-router>
    <tcp:endpoint host="10.192.111.10" port="10001" />
    <tcp:endpoint host="10.192.111.11" port="10001" />
    <tcp:endpoint host="10.192.111.12" port="10001" />
  </exception-based-router>
</outbound>
```

Another variation of this router is the `recipient-list-exception-based-router`, which uses a dynamic rather than static list of endpoints/recipients.

```
<outbound>
  <recipient-list-exception-based-router evaluator="xpath" expression="/Endpoint/Address" />
</outbound>
```

## Template Endpoint Router

The template endpoint router allows endpoints to be altered at runtime based on properties set on the current message or fallback values set on the endpoint properties. Templated values are expressed using square brackets around a property name, such as:

```
axis:http://localhost:8082/MyService?method=[SOAP_METHOD]
```

Configuration for this router is as follows:

```
<outbound>
  <template-endpoint-router>
    <outbound-endpoint address="foobar://server:1234/path/path?param1=[header1]&param2=[header2]" />
  </template-endpoint-router>
</outbound>
```

The header1 and header2 parameters are substituted with the actual values from the current message. The parameters can be used only in the query string, as the square brackets are not valid characters for the authority and path URI components.

## Custom Outbound Router

You can configure custom outbound routers by specifying the custom router class on the `<custom-outbound-router>` element and by using Spring properties.

Configuration for this router is as follows:

```

<outbound>
  <custom-outbound-router class="org.my.CustomOutboundRouter" transformers-ref="Transformer1">
    <tcp:endpoint host="10.192.111.10" port="10001" />
    <tcp:endpoint host="10.192.111.11" port="10001" />
    <mulexml:jxpath-filter expression="msg/header/resultcode = 'success'" />
    <spring:properties>
      <spring:property key="key1" value="value1"/>
      <spring:property key="key2" value="value2"/>
    </spring:properties>
  </custom-outbound-router>
</outbound>

```

Your Rating: 

Results:  4 rates

## Message Splitting and Aggregation

### *Message Splitting and Aggregation*

#### Message Splitters

##### *List Message Splitter*

A message splitter can be used to break down an outgoing message into parts and dispatch those parts over different endpoints configured on the router. The List Message Splitter accepts a list of objects that will be routed to different endpoints. The actual endpoint used for each object in the list is determined by a filter configured on the endpoint itself. If the endpoint's filter accepts the object, the endpoint will be used to route the object.

By default the AbstractMessageSplitter sets a correlation ID and correlation sequence on the outbound messages so that inbound routers such as the [Collection Aggregator](#) or [Correlation Resequencer](#) are able to resequence or combine the split messages.

The router configuration below expects the message payload to be a `java.util.List` and will route objects in the list that are of type `com.foo.Order`, `com.foo.Item`, and `com.foo.Customer`. The router will allow any number and combination of these objects.

Configuration for this router is as follows:

```

<outbound>
  <list-message-splitter-router>
    <jms:outbound-endpoint queue="order.queue">
      <payload-type-filter expectedType="com.foo.Order" />
    </jms:outbound-endpoint>
    <jms:outbound-endpoint queue="item.queue">
      <payload-type-filter expectedType="com.foo.Item" />
    </jms:outbound-endpoint>
    <jms:outbound-endpoint queue="customer.queue">
      <payload-type-filter expectedType="com.foo.Customer" />
    </jms:outbound-endpoint>
    <payload-type-filter expectedType="java.util.List" />
  </list-message-splitter-router>
</outbound>

```

Note that there is also a filter on the router itself that ensures that the message payload received is of type `java.util.List`. If there are objects in the list that do not match any of the endpoint filters, a warning is written to the log and processing continues. To route any non-matching object types to another endpoint, add the endpoint at the end of the list without a filter.

#### *Expression Splitter Router*

This router is similar to the list message splitter router, but it splits the message based on an [expression](#). The expression must return one or more message parts to be effective.

```

<outbound>
    <expression-splitter-router evaluator="xpath" expression="/mule:mule/mule:model/mule:service"
        disableRoundRobin="true" failIfNoMatch="false">
        <outbound-endpoint ref="service1">
            <expression-filter evaluator="xpath" expression="/mule:service/@name = 'service splitter'"/>
        </outbound-endpoint>
        <outbound-endpoint ref="service2">
            <expression-filter evaluator="xpath" expression="/mule:service/@name = 'round robin
deterministic'"/>
        </outbound-endpoint>
    </expression-splitter-router>
</outbound>

```

### Filtering XML Message Splitter



The XML Message Splitter is deprecated. Use the Expression Message Splitter with an xpath expression instead.

This router is similar to the List Message Splitter but operates on XML documents. Supported payload types are:

- org.dom4j.Document Objects
- byte[]
- java.lang.String

If no match is found, it is ignored and logged at the WARN level.

The router splits the payload into nodes based on the `splitExpression` property. The actual endpoint used for each object in the list is determined by a filter configured on the endpoint itself. If the endpoint's filter accepts the object, the endpoint will be used to route the object. Each part returned is actually returned as a new DOM4J document.

The router can optionally perform a validation against an external XML schema document. To perform the validation, set `externalSchemaLocation` to the XSD file in your classpath. Setting this property overrides whatever schema document you declare in the XML header.

By default, the router fails if none of the endpoint filters match the payload. To prevent the router from failing in this case, you can set the `failIfNoMatch` attribute to `false`.

Configuration for this router is as follows:

```

<outbound>
    <mulexml:filter-based-splitter splitExpression="root/nodes" validateSchema="true"
        externalSchemaLocation="/com/example/TheSchema.xsd">
        <vm:outbound-endpoint path="order">
            <payload-type-filter expectedType="com.foo.Order" />
        </vm:outbound-endpoint>
        <vm:outbound-endpoint path="item">
            <payload-type-filter expectedType="com.foo.Item" />
        </vm:outbound-endpoint>
        <vm:outbound-endpoint path="customer">
            <payload-type-filter expectedType="com.foo.Customer" />
        </vm:outbound-endpoint>
        <payload-type-filter expectedType="org.dom4j.Document" />
    </mulexml:filter-based-splitter>
</outbound>

```

The above splitters simply split an a message into multiple parts, each part being passed onto the next message processor or endpoints. There are also message routers that split the message before routing such as the Round-Robin Message Splitter. See [Message Routers](#)for more information about these.

### Message Aggregation

#### Collection Aggregator

The Collection Aggregator groups incoming messages that have matching group IDs before forwarding them. The group ID can come from the correlation ID or another property that links messages together.

You can specify the `timeout` attribute to determine how long the router waits in milliseconds for messages to complete the group. By default, if the expected messages are not received by the `timeout` time, an exception is thrown and the messages are not forwarded. As of Mule 2.2, you can set the `failOnTimeout` attribute to `false` to prevent the exception from being thrown and simply forward whatever messages have been received so far.

The aggregator is based on the [Selective Consumer](#), so you can also apply filters to the messages. Configuration for this router is as follows:

```
<inbound>
  <collection-aggregator-router timeout="6000" failOnTimeout="false">
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </collection-aggregator-router>
</inbound>
```

### **Message Chunking Aggregator**

After an outbound router such as the [List Message Splitter](#) splits a message into parts, the message chunking aggregator router reassembles those parts back into a single message. The aggregator uses the correlation ID, which is set by the outbound router, to identify which parts belong to the same message. This aggregator is based on the [Selective Consumer](#), so filters can also be applied to messages.

Configuration for this router is as follows:

```
<inbound>
  <message-chunking-aggregator-router>
    <expression-message-info-mapping correlationIdExpression="#[header:correlation]"/>
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </message-chunking-aggregator-router>
</inbound>
```

The optional `expression-message-info-mapping` element allows you to identify the correlation ID in the message using an expression. If this element is not specified, `MuleMessage.getCorrelationId()` is used.

The Message Chunking aggregator also accepts the `timeout` and (as of Mule 2.2) `failOnTimeout` attributes as described under [Collection Aggregator](#).

### **Custom Correlation Aggregator**

This router is used to configure a custom message aggregator. Mule provides an abstract implementation that has a template method that performs the message aggregation. A common use of the aggregator router is to combine the results of multiple requests such as "ask this set of vendors for the best price of X".

The aggregator is based on the [Selective Consumer](#), so you can also apply filters to messages. It also accepts the `timeout` and (as of Mule 2.2) `failOnTimeout` attributes as described under [Collection Aggregator](#).

Configuration for this router is as follows:

```
<inbound>
  <custom-correlation-aggregator-router class="org.mule.CustomAggregator">
    <payload-type-filter expectedType="org.foo.some.Object"/>
  </custom-correlation-aggregator-router>
</inbound>
```

There is an [AbstractEventAggregator](#) that provides a thread-safe implementation for custom aggregators, which you can use to write a custom aggregator router. For example, the [Loan Broker](#) examples included in the Mule distribution use a custom `BankQuotesInboundAggregator` router to aggregate bank quotes.

Your Rating: 

Results:  5 rates

## Asynchronous Reply Routers

### Asynchronous Reply Routers

[ Overview ] [ Single Asynchronous Reply ] [ Collection Asynchronous Reply ] [ Custom Asynchronous Reply ]

#### Overview

Asynchronous reply routers are used in request/response scenarios where message traffic is triggered by a request and the traffic needs to be consolidated before a response is given. The classic example of this is where a request is made and tasks are executed in parallel. Each task must finish executing and the results processed before a response can be sent back.

Asynchronous reply routers are only useful with services that use synchronous calls because there is no response when dispatching a message asynchronously. Mule provides aggregator routers that can be used in conjunction with a message splitter or recipient list router to aggregate messages before returning a response. For more information on these routers, see [Outbound Routers](#).

#### Example

Consider the inbound configuration and the asynchronous reply router in the `LoanBroker` configuration:

```
<service name="LoanBroker">
    <inbound>
        <vm:inbound-endpoint path="Loan.Requests"/>
    </inbound>
    <component class="org.mule.samples.loanbroker.SyncLoanBroker">
        <outbound>
            <static-recipient-list-router>
                <reply-to address="jms://Loan.Quotes"/>
                <message-property-filter expression="recipients!=null"/>
            </static-recipient-list-router>
        </outbound-router>
        <async-reply>
            <jms:inbound-endpoint queue="Loan.Quotes"/>
            <custom-async-reply-router class=
                "org.mule.samples.loanbroker.routers.BankQuotesResponseAggregator"/>
        </async-reply>
    </component>
</service>
```

This configuration specifies that the Loan Broker will receive requests from `vm://Loan.Requests` and will dispatch multiple requests to different banks via the outbound router. The bank endpoints are defined in a List called 'recipients', which is a property on the outbound message. The important setting on the outbound router is the `<reply-to>` endpoint, which tells Mule to route all responses to the `jms://Loan.Quotes` endpoint, which is the endpoint on which the async-reply router is listening. When all responses are received, the `BankQuotesResponseAggregator` selects the cheapest quotes and returns it. Mule then handles returning this to the requester. The `<reply-to>` endpoint is applied to the next service invoked. For example, if service1 dispatches to service2, and service1 has an outbound router with a reply-to endpoint, service2 will send the results of its invocation to the reply-to endpoint. For more information, see [ReplyTo](#).

#### Response Transformers

If you want to transform a response message without doing any other work on the response, you set the `transformers` attribute on the response router without any other routing configuration.

```
<response-router transformers="OrderConfirmationToXml XmlToWebPage"/>
```

#### Time-outs

The timeout setting determines how long Mule should wait for replies before returning the result. The default value is determined by the value of the `defaultSynchronousEventTimeout` attribute that has been configured for the Mule instance. (For more information, see [Global Settings Configuration Reference](#).) You can also specify an independent timeout value for asynchronous replies for a given service using the optional `timeout` attribute on the `async-reply` element.

The optional `failOnTimeout` attribute determines whether to throw an exception if the router times out before all expected messages have been received. If set to false (the default), the current messages are returned for processing.

For example:

```
<async-reply failOnTimeout="false" timeout="2000">
  <inbound-endpoint ref="replyEndpoint"/>
  <collection-async-reply-router/>
</async-reply>
```

The rest of this page describes the asynchronous reply routers you can use. For detailed information on the elements you configure for asynchronous reply routers, see [Asynchronous Reply Router Configuration Reference](#).

### Single Asynchronous Reply

The Single Asynchronous Reply router configures a single response router. This router returns the first message it receives on a reply endpoint and discards the rest.

```
<single-async-reply-router/>
```

### Collection Asynchronous Reply

The Collection Asynchronous Reply router configures a collection response router. This router returns a MuleMessageCollection message type that contains all messages received for the current correlation.

```
<collection-async-reply-router/>
```

### Custom Asynchronous Reply

The Custom Asynchronous Reply router allows you to configure a custom asynchronous reply router. To configure the custom router, set the class attribute to the custom router class.

```
<custom-async-reply-router class="org.mule.CustomAsyncReplyRouter" />
```

Your Rating: 

Results:  0 rates

## Catch-all Strategies

### ***Catch-all Strategies***

[ Forwarding ] [ Custom Forwarding ] [ Logging ] [ Custom ]

You can configure a catch-all strategy that will be invoked if no routing path can be found for the current message. An inbound or outbound endpoint can be associated with a catch-all strategy so that any orphaned messages can be caught and routed to a common location. For detailed information on the elements you configure for catch-all strategies, see [Catch-all Strategy Configuration Reference](#).

For example:

```

<service name="dataService">
    <inbound>
        <inbound-endpoint address="vm://in2" connector-ref="vmQueue">
            <string-to-byte-array-transformer/>
        </inbound-endpoint>

        <selective-consumer-router>
            <payload-type-filter expectedType="java.lang.Integer"/>
        </selective-consumer-router>

        <custom-forwarding-catch-all-strategy class=
"org.mule.test.usecases.routing.InboundTransformingForwardingCatchAllStrategy">
            <outbound-endpoint address="vm://catchall" connector-ref="vmQueue">
                <string-to-byte-array-transformer/>
            </outbound-endpoint>
        </custom-forwarding-catch-all-strategy>
    </inbound>
    ...
    <outbound>
        <filtering-router transformer-refs="TestCompressionTransformer">
            <outbound-endpoint address="test://appleQ2" name="TestApple-Out" />
            <payload-type-filter expectedType="java.lang.String" />
        </filtering-router>
        <custom-catch-all-strategy class="org.mule.tck.testmodels.mule.TestCatchAllStrategy" />
    </outbound>
    ...
</service>

```

Following are descriptions of the different catch-all strategies you can use.

### Forwarding

This catch-all strategy is used to forward the message to an endpoint that is configured if no outbound routers match.

```

<forwarding-catch-all-strategy>
    <jms:outbound-endpoint queue="error.queue" />
</forwarding-catch-all-strategy>

```

### Custom Forwarding

This catch-all strategy is the same as the default forwarding catch-all strategy, but it allows you to specify a custom implementation to use by configuring the `class` attribute. You can also configure additional optional properties.

```

<custom-forwarding-catch-all-strategy class="org.my.CustomForwardingCatchAllStrategy">
    <jms:outbound-endpoint queue="error.queue" />
    <spring:property key="myProperty" value="myValue" />
</forwarding-catch-all-strategy>

```

### Logging

This catch-all strategy does nothing with the message and simply logs a warning indicating that the message was not dispatched because there was no routing path defined.

```

<logging-catch-all-strategy/>

```

## Custom

This catch-all strategy allows you to use a custom class to perform whatever behavior you require. To implement a custom catch-all strategy that forwards the message to an endpoint, you should used the custom forwarding catch-all strategy instead.

```
<custom-catch-all-strategy/>
```

Your Rating: 

Results:  2 rates

# Using Mule Configuration Patterns

## Using Mule Configuration Patterns

### Introduction

Configuring Mule involves XML, and though using a decent XML editor can help a lot (thanks to the contextual help it provides from Mule's schemas), there are still a enough angle brackets to warrant a coffee break as projects get more complicated. As the number of services in a Mule project increases, so does the amount of noise in its configuration files, making it harder and harder to understand and maintain them. In Mule 3, we've decided to tackle this problem with the introduction of pattern-based configuration.

### XML namespace:

```
xmlns:file="http://www.mulesoft.org/schema/mule/pattern"
```

### XML Schema location:

```
http://www.mulesoft.org/schema/mule/pattern  
http://www.mulesoft.org/schema/mule/pattern/3.1/mule-pattern.xsd
```

### When to Use a Pattern



A configuration pattern provides a very specific integration feature.

The following table will help you pick up the configuration pattern that fits your needs.

Pattern Name	Usage
Simple Service	Exposes JAX-WS annotated components as SOAP web services. Exposes JAX-RS annotated beans as RESTful components. Can also handle JAXB, XML and raw content with simple POJO components.
Web Service Proxy	Proxies remote web services. Can perform transformations on the SOAP envelope. Can rewrite or redirect remote WSDLs to local ones.
Bridge	Establishes a direct conduit between an inbound endpoint and an outbound endpoint. Supports request-response and one-way bridging. Can perform transformations. Supports transactional bridging of inbound to outbound.
Validator	Validates inbound messages against a defined acceptance filter. Returns an ACK or NACK response synchronously and dispatches valid messages asynchronously.

Should none of the existing patterns satisfy your need, turn to [Using Flows for Service Orchestration](#) for advanced configuration mechanisms.

### Related Topics

- [Pattern-Based Configuration](#)
- [Providing feedback or suggesting new Patterns](#)

Your Rating: 

Results:  1 rates

## Pattern-Based Configuration

### Pattern-Based Configuration

[ Demonstrating a Commitment to Patterns ] [ Sharing Properties ] [ Future Directions ]

Configuring Mule involves XML, and though using a decent XML editor can help a lot (thanks to the contextual help it provides from Mule's schemas), there are still enough angle brackets to warrant a coffee break as projects get more complicated. As the number of services in a Mule project increases, so does the amount of noise in its configuration files, making it harder and harder to understand and maintain them. In Mule 3, we've decided to tackle this problem with the introduction of pattern-based configuration.

#### Demonstrating a Commitment to Patterns

Patterns-based engineering is a software discipline that encourages the identification of recurrent practices in order to create consumable artifacts that encourage subsequent and efficient re-use. We believe that these principles can be applied to configuring Mule. This is why we've started an effort to collect and implement configuration patterns.

Concretely, these patterns will present themselves as syntactic sugar for Mule configuration. They will be new XML elements that will allow you to perform common configuration tasks with the least possible amount of XML.

#### Sharing Properties

On top of that, configuration patterns will also offer some form of inheritance to allow sharing common properties across them. Finally, they will rely heavily on the re-architecture of Mule's core, which has paved the way for the creation of such specialized artifacts.

Any transport or module will have the possibility to contribute configuration patterns, so some of these patterns will be highly specialized. Of course, it will always be possible to use the more generic configuration elements offered by Mule for cases when a pattern doesn't fully cover your needs.

#### Future Directions

These documentation pages will be updated on an ongoing basis as new configuration patterns emerge from the Mule community.

Your Rating: 

Results:  0 rates

## Simple Service Pattern

### Simple Service Pattern

The goal of the Simple Service pattern is as simple as its name suggests: to provide a simple way to expose request-response services. In other terms, it is used to expose some business logic in a synchronous manner over the transport of your choice.



#### Core Features

Simple Service allows the deployment of SOA-style services on Mule in a short and elegant manner. Without further ado, let's delve into examples that will illustrate the extensive capacities of this new configuration element.

#### Exposing a POJO as a Simple Service

```
<pattern:simple-service name="maths-service"
    address="vm://maths.in"
    component-class="org.mule.tck.services.SimpleMathsComponent" />
```

That is all! An instance of SimpleMathsComponent is now accessible via a VM endpoint, which has been automatically configured to be synchronous (i.e. request-response). As usual, Mule will use a resolution strategy to find the best method for receiving the payload that has been sent to the service.

Of course, referencing global endpoints and components is also possible:

#### Using References

```
<pattern:service name="maths-service"
    endpoint-ref="maths-service-endpoint"
    component-ref="math-component" />
```

By the same token, transformers can be configured on the service:

#### Transformers

```
<pattern:service name="byte-array-massager"
    address="vm://bam.in"
    transformer-refs="byte-array-to-string append-bar"
    responseTransformer-refs="append-baz string-to-byte-array"
    component-class="org.mule.component.simple.EchoComponent" />
```

Simple Service can also work with component configuration elements from any supported schema. The following shows a very convenient way to configure server stubs during functional tests:

#### Functional Test Service

```
<pattern:service name="functional-test-component"
    address="vm://ftc.in">
    <test:component />
</pattern:service>
```

Similarly, inbound endpoint and exception strategy can both be configured as child elements:

#### Child Elements

```
<pattern:service name="maths-service"
    component-class="org.mule.tck.services.SimpleMathsComponent">
    <vm:inbound-endpoint path="maths.in"
        exchange-pattern="request-response" />
    <custom-exception-strategy class="com.acme.AcmeExceptionStrategy" />
</pattern:service>
```

Simple Service, like all configuration pattern elements, supports inheritance the same way standard Spring beans do. Inheritance can be used for sharing common settings across configuration elements. This is illustrated hereafter:

#### Inheritance

```
<pattern:service name="global-exception-strategy"
    abstract="true">
    <custom-exception-strategy class="com.acme.AcmeExceptionStrategy" />
</pattern:service>

<pattern:service name="inherited-exception-strategy"
    parent="global-exception-strategy"
    address="vm://maths.in"
    component-class="org.mule.tck.services.SimpleMathsComponent" />
```

**Note:** Inheritance of a nested 'component' element is not supported and as such cannot be defined on an abstract simple-service.

#### SOAP Services

Simple Service allows you to expose JAX-WS annotated components as SOAP services. All you need to do is add a type attribute and your

component can be invoked with SOAP calls.

Here is an example:

```
JAX-WS Service
```

```
<pattern:service name="weather-forecaster-ws"
    address="http://localhost:6099/weather-forecast"
    component-class="org.mule.test.integration.tck.WeatherForecaster"
    type="jax-ws" />
```

Note that if the transport used is HTTP, Simple Service will happily serve a WSDL document for requests whose paths end with "?WSDL".

## RESTful Services

Simple Service is also capable of exposing JAX-RS annotated components as RESTful services.

Again, using the relevant type is the only effort needed to start serving RESTful resources, as shown here:

```
JAX-RS Service
```

```
<pattern:service name="weather-report-rsc"
    address="http://localhost:6099/rest"
    component-class="org.mule.test.integration.tck.WeatherReportResource"
    type="jax-rs" />
```

If the component in the above configuration is annotated with @Path("/weather-report"), then its resources will be reachable under this URI: <http://localhost:6099/rest/weather-report>.

JAX-RS services work with the HTTP and Servlet transports.

## JAXB Support

Simple Service can handle JAXB serialized payloads. Nothing special is needed in the XML configuration:

```
JAXB Serialized Service
```

```
<pattern:service name="weather-jaxb-consumer"
    address="vm://weather-consumer.in"
    component-class="org.mule.test.integration.tck.WeatherReportConsumer" />
```

But, for this to work, it is required that @Payload, a Mule-specific annotation, is used on the component:

```
Annotated Service Component for JAXB Payloads
```

```
package org.mule.test.integration.tck;

import org.mule.api.annotations.param.Payload;

public class WeatherReportConsumer
{
    public String consume(@Payload WeatherReportType weatherReport)
    {
        return weatherReport.report;
    }
}
```

## XPath Support

Finally, Simple Service can also handle XML payload with a direct extraction of values via XPath expressions. Like with JAXB, nothing special is needed in XML:

## XPath Payload Service

```
<pattern:simple-service name="weather-xpath-consumer"
    address="vm://weather-xpath-consumer.in"
    component-class="org.mule.test.integration.tck.WeatherReportXpathConsumer" />
```

But again, a Mule annotation, @XPath in this case, is needed for this to work:

## Annotated Service Component for XPath Payloads

```
package org.mule.test.integration.tck;

import org.mule.api.annotations.expression.XPath;

public class WeatherReportXpathConsumer
{
    public String consume(@XPath(value = "/weatherReport/report") String report)
    {
        return report;
    }
}
```

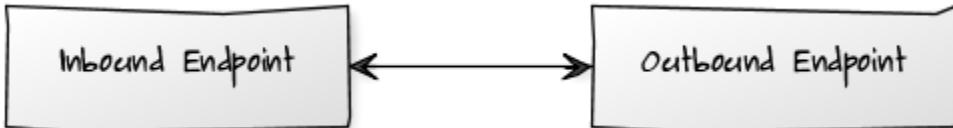
Your Rating: 

Results:  5 rates

## Bridge Pattern

### Bridge Pattern

Connecting systems together is one of the most essential task of integration. Mule ESB offers the necessary building blocks for achieving such connections. One of these building blocks allows establishing bridges between message sources and destinations. Known as the messaging bridge pattern, this building block is available in Mule 3 as the newly introduced Bridge configuration element.



### Core Features

A messaging bridge acts as a direct conduit between an inbound endpoint and an outbound endpoint. It is a neutral component as it doesn't apply any business logic on the messages that flow through it. This said, as we will see later on, a bridge can perform protocol adaptation and data transformation.

By default, the Bridge configuration element establishes a request-response conduit between its endpoints: any response coming from the outbound endpoint will be routed back to the inbound endpoint. The following illustrates such a synchronous bridge:

## Synchronous - aka Request-Response - Bridge

```
<pattern:bridge name="request-response-bridge"
    inboundAddress="vm://synchronous-bridge.in"
    outboundAddress="vm://maths-service.in" />
```

Using a synchronous bridge made sense in this case because the target of the outbound endpoint was a request-response math computation service, whose response was expected by the caller of the bridge. In other scenarios, no response needs to be routed back: the bridge becomes a one-way conduit. Consider the following example where messages are sent to the logging system in a fire and forget manner:

### One-Way Bridge

```
<pattern:bridge name="one-way-bridge"
    exchange-pattern="one-way"
    inboundAddress="vm://asynchronous-bridge.in"
    outboundAddress="vm://log-service.in" />
```

The Bridge element, like other pattern-based configuration elements, supports referencing global endpoints:

### Endpoint References

```
<pattern:bridge name="endpoint-ref-bridge"
    inboundEndpoint-ref="endpoint-ref-bridge-channel"
    outboundEndpoint-ref="maths-service-channel" />
```

It also supports child elements to enable advanced configuration of endpoints and definition of exception strategy (in case something nasty happens while a message was processed by the Bridge):

### Child Elements

```
<pattern:bridge name="child-endpoint-bridge">
    <vm:inbound-endpoint path="child-endpoint-bridge.in"
        exchange-pattern="request-response" />
    <vm:outbound-endpoint path="maths-service.in"
        exchange-pattern="request-response" />
    <custom-exception-strategy class="com.acme.AcmeExceptionStrategy" />
</bridge>
```

Finally, inheritance is also supported, making it possible to share properties across several Bridges:

### Inheritance

```
<pattern:bridge name="abstract-parent-bridge"
    abstract="true"
    outboundAddress="vm://maths-service.in" />

<pattern:bridge name="concrete-child-bridge"
    parent="abstract-parent-bridge"
    inboundAddress="vm://concrete-child-bridge.in" />
```

## Adaptation and Transformation

So far, our examples were showing bridges with homogeneous transports: the same transport (VM) was used for both the inbound and outbound endpoints. Because it relies on Mule's protocol adaptation and messaging abstraction, a bridge can very well handle scenarios where heterogeneous protocols are used. Consider the following example:

### Heterogeneous Transports Bridge

```
<pattern:bridge name="dlqDumper"
    exchange-pattern="one-way"
    inboundAddress="jms://myDlq"
    outboundAddress="file://./test?outputPattern=#[header:INBOUND:JMSMessageID].dl" />
```

As you can see, we have configured a bridge to handle the messages we're receiving on a dead letter queue (DLQ). This bridge consumes all the dead messages sent to the specified JMS queue and writes them to the file system. Of course, this is a one-way bridge: we're not interested into sending anything back to the DLQ!

Beyond handling heterogeneous protocols, a bridge can handle differences in data format. Indeed, as mentioned above, a bridge can perform transformations, which is oftentimes needed when integrating disparate systems. The following shows an example where data is transformed to

and from a canonical form:

### Transforming Bridge

```
<pattern:bridge name="transforming-bridge"
    inboundAddress="vm://transforming-bridge.in"
    transformer-refs="to-canonical-form"
    responseTransformer-refs="from-canonical-form"
    outboundAddress="vm://target-service.in" />
```

### Transaction Support

Sometimes, important messages transit through a bridge: in that case, a message must reliably be consumed from the inbound endpoint and delivered to the outbound endpoint. If this delivery fails, the message should be pushed back so the delivery can be attempted again later.

The way to achieve this is to declare Bridge as being transacted. In that case, it will consume its inbound messages and dispatch its outbound ones within a transaction. Look at the following example for an idea of how this works:

### Transacted Bridge

```
<pattern:bridge name="queue-to-topic"
    transacted="true"
    inboundAddress="jms://myQueue"
    outboundAddress="jms://topic:myTopic" />
```

In this example, we bridge a JMS queue to a topic. We've used the transacted attribute as we want to ensure that all inbound messages will be successfully consumed and re-published.

Obviously, for this to work, the transports used by the bridge should support transactions. If the inbound and outbound endpoints use heterogeneous protocols, the bridge will look for an XA transaction manager (which must be already [configured](#)).

## Validator Pattern

### Validator Pattern

When processing messages, a certain format is always assumed to have been respected so that the required data can be retrieved. It is possible and oftentimes desirable to be very liberal with the data representation used by incoming messages: as long as you can find the needed pieces of information, the rest doesn't matter.

But sometimes, a strict up-front validation of incoming messages is needed. This is that kind of scenarios that the Validator configuration pattern addresses.



### Core Features

Services that expose a SOAP API benefit from the validation inherent to their host web service framework, which enforces the compliance of incoming messages against the strict contract defined with WSDL. For all the other types of services, Mule's filtering and routing infrastructure provides all the necessary building blocks for putting a strict validation in place.

The Validator pattern provides you with a framework for performing this kind of up-front validation. It has been designed in such way that validation is performed synchronously while dispatching of valid requests is performed asynchronously. This provides a decoupling of the validation and processing phases, which is a common pattern when clients are producing messages faster they are actually processed.

The Validator pattern leverages Mule's extensive expression framework for building the acknowledgement and rejection messages. It uses Mule's filters to express the conditions for a message to be valid.

Let's take a look at a Validator that accepts only integers:

### Integer Only Validator

```
<pattern:validator name="integer-validator"
    inboundAddress="vm://validator.in"
    ackExpression="#[string:GOOD:#[message:payload]@#[context:serviceName]]"
    nackExpression="#[string:BAD:#[message:payload]@#[context:serviceName]]"
    outboundAddress="vm://test-service.in">
    <payload-type-filter expectedType="java.lang.Integer"/>
</pattern:validator>
```

Suppose we send 123 to this Validator: it will accept the message and acknowledge with "GOOD:123@integer-validator". To the opposite, if we send "abc" it will reject the message with "BAD:abc@integer-validator".

It is possible to use global endpoints and filters and refer to them from a Validator, as show hereafter:

### Using References

```
<pattern:validator name="validator-with-refs"
    inboundEndpoint-ref="validator-with-refs-channel"
    ackExpression="#[string:GOOD:#[message:payload]@#[context:serviceName]]"
    nackExpression="#[string:BAD:#[message:payload]@#[context:serviceName]]"
    validationFilter-ref="int-payload-filter"
    outboundEndpoint-ref="test-service-channel" />
```

It is also possible to define the inbound and outbound endpoints and a custom exception strategy as child elements, which is handy when you use endpoints that need complex configuration:

### Child Elements

```
<pattern:validator name="validator-with-child-elements">
    <vm:inbound-endpoint path="validator-with-child-endpoints.in"
        exchange-pattern="request-response" />
    <payload-type-filter expectedType="java.lang.Integer"/>
    <outbound-endpoint ref="test-service-channel" />
    <custom-exception-strategy
        class="com.acme.ValidationExceptionStrategy" />
</pattern:validator>
```

Finally, like all configuration patterns, the Validator element supports inheritance. This is useful if you want to share ack/nack expressions or validation rules across several validators:

### Inheritance

```
<pattern:validator name="abstract-parent-validator"
    abstract="true"
    ackExpression="#[string:GOOD:#[message:payload]@#[context:serviceName]]"
    nackExpression="#[string:BAD:#[message:payload]@#[context:serviceName]]" />

<pattern:validator name="concrete-validator"
    parent="abstract-parent-validator"
    inboundAddress="vm://concrete-validator.in"
    outboundAddress="vm://test-service.in">
    <payload-type-filter expectedType="java.lang.Integer"/>
</pattern:validator>
```

## Outbound Errors



Since Mule 3.0.1

By default, if something wrong happens during the outbound dispatch of a valid message, the caller will not know about it. In that case, it's possible to use Mule's exception handling mechanism to log the error, store the valid message to be dispatched and even attempt redeliveries.

That said, it is possible that in some scenarios the caller must be informed that the dispatch of its valid message has failed. For that matter, a third optional expression can be used:

```
Inheritance

<pattern:validator name="dispatch-error"
    inboundAddress="vm://dispatch-error.in"
    ackExpression="#[string:GOOD:#[message:payload]@#[context:serviceName]]"
    nackExpression="#[string:BAD:#[message:payload]@#[context:serviceName]]"
    errorExpression="#[string:ERROR:#[message:payload]@#[context:serviceName]]"
    outboundAddress="http://acme.com/services/fragile"
    validationFilter-ref="int-payload-filter" />
```

With this configuration, the errorExpression will be used to create the response to the caller if the outbound dispatch fails. Note that when this expression is used, the outbound endpoint will use the request-response exchange pattern.

Your Rating:  Results:  1 rates

## Web Service Proxy Pattern

### Web Service Proxy Pattern

Proxying web services is a very common practice used for different reasons like security or auditing. This pattern allows a short and easy configuration of such a proxy.



### Core Features

A web service proxy acts as an intermediate between a caller application and the target web service. This gives the proxy a chance to transparently introduce new behaviors in the calling sequence. For example, it can:

- add or remove HTTP headers,
- transform the SOAP envelope (body or header) to add or remove specific entries,
- rewrite remote WSDLs so they appear to bind to services inside a corporate firewall,
- introduce custom error handling.

Let's take a look at Web Service Proxy in action:

```
Web Service Proxy

<pattern:web-service-proxy name="weather-forecast-ws-proxy"
    inboundAddress="http://localhost:8090/weather-forecast"
    outboundAddress="http://ws.acme.com:6090/weather-forecast" />
```

With this configuration in place, all calls to the local weather forecaster proxy will be redirected to the remote one.

The Web Service Proxy is provided by the ws module, which must be present on the classpath to be usable. Its namespace is <http://www.mulesoft.org/schema/mule/ws> and its schema location is <http://www.mulesoft.org/schema/mule/ws/3.0/mule-ws.xsd>.

The true value add comes from the automatic address rewriting that will be performed by the proxy: calling <http://localhost:8090/weather-forecast?wsdl> will return the remote WSDL where the port addresses will have been automatically rewritten based on the URL of the request hitting the proxy. That way, if your Mule instance is accessed behind a load balancer or any kind of network indirection, the generated WSDL will point the caller to port addresses that respect your particular network topology.

As said above, the proxy can perform changes on the SOAP invocation by the use of transformers. This is demonstrated hereafter:

### Proxy with Transformers

```
<pattern:web-service-proxy name="weather-forecast-ws-proxy-transformers"
    inboundAddress="http://localhost:8090/weather-forecast"
    transformer-refs="zip-code-transformer add-credentials-transformer"
    responseTransformer-refs="weather-code-transformer"
    outboundEndpoint-ref="target-ws-endpoint" />
```

Notice how transformers are introduced by using references to globally declared ones. This technique is also applicable to global endpoints, as you can see with the above reference to target-ws-endpoint.

The Web Service Proxy element supports child elements. The following shows a configuration variant where endpoints are declared internally and an exception strategy has been added in:

### Child Elements

```
<pattern:web-service-proxy name="weather-forecast-ws-proxy">
    <http:inbound-endpoint address="http://localhost:8090/weather-forecast" />
    <http:outbound-endpoint address="http://ws.acme.com:6090/weather-forecast" />
    <custom-exception-strategy class="com.acme.AcmeExceptionStrategy" />
</pattern:web-service-proxy>
```

Finally, the Web Service Proxy also supports inheritance, which allows sharing common configuration attributes across several concrete instantiations of the proxy. Check out the following to see how inheritance works:

### Inheritance

```
<pattern:web-service-proxy name="abstract-ws-proxy-zipcode-changer"
    abstract="true"
    transformer-refs="zip-code-transformer add-credentials-transformer"
    responseTransformer-refs="weather-code-transformer" />

<pattern:web-service-proxy name="weather-forecast-ws-proxy"
    parent="abstract-ws-proxy-zipcode-changer">
    <http:inbound-endpoint address="http://localhost:8090/weather-forecast" />
    <http:outbound-endpoint address="http://ws.acme.com/weather-forecast" />
</pattern:web-service-proxy>
```

The proxy offers a few extra options as far as WSDL handling is concerned. Let's look at them.

### WSDL Redirection

In some cases, the remote web service doesn't follow the common practice of exposing its WSDL on the same address as the service with a "?wsdl" appended at the end. In that case, it is required to point the Web Service Proxy to the exact location of the remote WSDL, as illustrated there:

### Remote WSDL Redirection

```
<pattern:web-service-proxy name="weather-forecast-ws-proxy"
    inboundAddress="http://localhost:8090/weather-forecast"
    outboundAddress="http://ws.acme.com:6090/weather-forecast"
    wsdlLocation="http://ws.acme.com:6090/wsdl/weather-forecast" />
```

In this scenario, the remote WSDL will have its port addresses rewritten as explained above.

For the case when no remote WSDL is available or if the remote WSDL needs manual adjustment before being exposed by the Web Service Proxy, the solution consists in storing the correct WSDL as a local file and have the proxy serve it. This is done as shown here:

### File WSDL Redirection

```
<pattern:web-service-proxy name="weather-forecast-ws-proxy"
    inboundAddress="http://localhost:8090/weather-forecast"
    outboundAddress="http://ws.acme.com:6090/weather-forecast"
    wsdlFile="path/to/correct/weather-forecaster.wsdl" />
```

In this case, the WSDL will be served as is from the file: no rewriting will occur.

Your Rating:  Results:  6 rates

## Message Sources and Message Processors

### Elements of Mule Programming

[ [Message Sources](#) ] [ [Message Processors](#) ]

Mule 3 provides three different constructs that can be used to build applications:

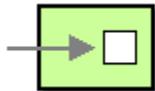
- [Using Mule Services](#) are the classic Mule way of organizing message flow. Each service consists of three sections:
  - Input, where messages are received
  - An optional component, where any sort of application logic can be applied to a message
  - An optional output, where the messages are sent to other services or transports.
- [Using Flows for Service Orchestration](#), which are new in Mule 3. A flow is a combination of message sources and message processors that doesn't have as fixed a format as a service does. It can be as simple or as complex as required, and can include, for instance, processing by multiple components before any output is performed.
- [Using Mule Configuration Patterns](#) are also new in Mule 3. A configuration pattern is like a pre-built flow: the logic is already built into it, so that only some simple tailoring is needed to make it functional.

A Mule application can contain any or all of these.

Within flows, the streamlined architecture of Mule 3 allows [Transformers](#), [Filters](#), [Components](#), [Routers](#) and other message processing artifacts to be used, and nested, freely as required. They all implement a common [MessageProcessor](#) interface and can be used interchangeably. Services also allow these building blocks to be used in some specific extension points.

Here you'll find information about the different building blocks provided by Mule presented in categories.

## Message Sources



A message source receives or generates new messages to be processed by Mule. Inbound Endpoints are currently the only supported Message Source type.

### Inbound Endpoints

Inbound Endpoints receive new messages from a channel or resource by using a server socket, polling a remote socket or resource, or by registering a listener.

For information about available transports see [Connecting Using Transports](#)

For information about configuring endpoints see [Configuring Endpoints](#)

### Poll (From Mule 3.1)

Rather than using an inbound endpoint you can poll any message processor and use the result as the source of your flow. A frequency in milliseconds can be configured otherwise the default of 1s is used. Examples of things you can poll are outbound-endpoints, other flows or processor chains or any message processor.

To configure polling use the `<poll>` element instead of an inbound endpoint.

```

<flow name="myFlow">
    <poll frequency="2000">
        <http:outbound-endpoint host=".." port=".." />
    </poll>
    <processor ref="" />
    <processor ref="" />
</flow>

```

Currently `<poll>` has no support for cron expression or for customizing its threading profile

## Message Processors



After a message has been received from a Message Source it is processed by Mule using one or more message processors. You can use message processors in certain extension points with [Services](#), or with complete flexibility when using [Flows](#).

Message Processors can be categorized by function:

### Perform Some Logic

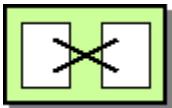


You'll often need to perform some business logic as part of your flow. Mule supports components implemented in Java and using scripting languages.

Components can do whatever you need them to do including mutating the message if required. Components support the concept of entry point resolution and lifecycle adaption designed to accommodate the use of component implementations with Mule without modification.

Look here for more about configuring Mule [Components](#).

### Transform the Message



When external systems use different message formats or you need to convert the payload type you'll need to one or more Message Transformers.

See [Using Transformers](#) for a list of available transformers as well as information on how to implement your own.

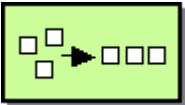
### Filter messages

	To only allow certain messages to continue to be processed you can use a <a href="#">Filter</a> . Filters may filter based on message type, message contents or some other criteria.
	To prevent messages from being processed if security credentials are not provided or do not match you can use <a href="#">Security Filter</a>
	To filter out duplicate messages you should use an <a href="#">Idempotent Filter</a>

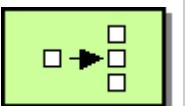
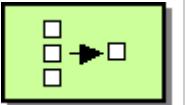
## Control Message Flow

There are a number of ways in which you can control message flow, which are described below. These are specified somewhat different in flows and services. In services, since the input and output sections are quite distinct, there are separate groups of **Inbound Routers** and **Outbound Routers**. In flows, **routers** are a subset of message processors.

### Resequence Messages

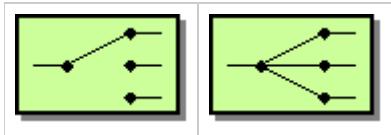
	In order to re-sequencer incoming messages use a Message Re-sequencer
---	---

### Split or Aggregate Messages

	Message splitters allow a single incoming message to be split into $n$ pieces each of the parts being passed onto the next message processor as a new message.
	Aggregators do the opposite and aggregate multiple inbound messages into a single message.

For information on provided splitter and aggregator implementations and details on how to implement your own see [Message Splitting and Aggregation](#)

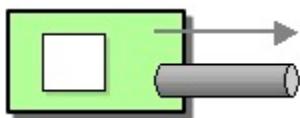
### Route Messages



In order to determine message flow in runtime Message Routers are used. Message routing can be configured statically or is determined in runtime using message type, payload or properties or some other criteria. Some message routers route to a single route whereas other routers route to multiple routes.

[Routing Message Processors](#)

### Send Messages over a transport



Once you have completed message processing you may wish to send the resulting message to an external service or location. You may also need to invoke a remote service elsewhere in the flow.

Outbound endpoints are used to send messages over a channel using a transport.

For information about available transports see [Connecting Using Transports](#)

For information about configuring endpoints see [Configuring Endpoints](#)

### Other

#### Message Processor Chain

A Message Processor Chain allows to define a reusable set of message processors that are chained together and invoked in sequence. When configuring Mule using XML a processor chain is defined using the `processor-chain` element.

```
<processor-chain name="myReusableChain">
  <byterarray-to-object-transformer />
  <expression-filter expression=" " />
  <custom-processor class=" " />
</processor-chain>
```

## Response Adaptor

A response adaptor is configured using the `response` element. It is used when you want to use a Message Processor on a response message. In the following case the append-string-transformer is invoked after response was received from the outbound endpoint invocation. This can be useful when you have a message process that performs response processing (e.g. CXF) and you need to add a message processor after this.

```
<http:outbound-endpoint address="http://foo.bar/formAction" exchange-pattern="request-response"
method="POST">
  <response>
    <append-string-transformer message=" - RECEIVED BY MULE" />
  </response>
</http:outbound-endpoint>
```

In the following example response block is invoked after the flow finished processing and before the response message is returned to the caller of the inbound endpoint.

```
<flow ...>
  <http:inbound-endpoint address="http://localhost:8080/hello" exchange-pattern="request-response">

    <response>
      <message-properties-transformer>
        <add-message-property key="Content-Type" value="text/html" />
      </message-properties-transformer>
    </response>
  </http:inbound-endpoint>
  <component class="com.foo.Bar" />
</flow>
```

## Custom Message Processors

Custom Message Processors can be implemented by simply extending the `MessageProcessor` or `InterceptingMessageProcessor` interface and using the `<custom-processor>` element. If you prefer to used a referenced spring bean as a message processor then you can use the standard `<processor ref=""/>` element and reference it directly.

### Configuring a custom message processor with a class name

```
<custom-processor name="customMsgProc" class=" " />
```

### Configuring a custom message processor by referencing a spring bean

```
<processor ref="myBean" />
```

For information on implementing your own Filters or Transformers see the respective pages. There is also more detailed information on implementing your own [Custom Message Processors](#).

Your Rating: 

Results:  2 rates

## Routing Message Processors

## Using Message Processors to Control Message Flow

[ Quick Reference ] [ All ] [ Async ] [ Choice ] [ Collection Aggregator ] [ Collection Splitter ] [ Custom Aggregator ] [ Custom Processor ] [ First Successful ] [ Idempotent Message Filter ] [ Idempotent Secure Hash Message Filter ] [ Message Chunk Aggregator ] [ Message Chunk Splitter ] [ Message Filter ] [ Processor Chain ] [ Recipient List ] [ Redelivery Policy ] [ Request Reply ] [ Resequencer ] [ Round Robin ] [ Splitter ] [ Until Successful ] [ WireTap ]

Message Processors are used within flows to control how messages are sent and received within that flow. This is further described in [Using Flows for Service Orchestration](#). (Within services, the same job is performed by message routers – see [Using Mule Services](#) and [Using Message Routers](#) for further details.)

Click a link in the Quick Reference table below for details on a specific message processor.

### Quick Reference

Message Processor	Description
All	Broadcast a message to multiple targets
Async	Run a chain of message processors in a separate thread
Choice	Send a message to the first matching message processor
Collection Aggregator	Aggregate messages into a message collection
Collection Splitter	Split a message that is a collection
Custom Aggregator	A custom-written class that aggregates messages
Custom Processor	A custom-written message processor
First Successful	Iterate through message processors until one succeeds (added in 3.0.1)
Idempotent Message Filter	Filter out duplicate message by message ID
Idempotent Secure Hash Message Filter	Filter out duplicate message by message content
Message Chunk Aggregator	Aggregate messages into a single message
Message Chunk Splitter	Split a message into fixed-size chunks
Message Filter	Filter messages using a filter
Processor Chain	Create a message chain from multiple targets
Redelivery Policy	Specify processing for a message which has been redelivered sufficiently often
<b>[Mule 3.2] Request Reply</b>	Receive a message for asynchronous processing and accept the asynchronous response on a different channel
Resequencer	Reorder a list of messages
Round Robin	Round-robin among a list of message processors (added in 3.0.1)
<b>[Mule 3.2] Until Successful</b>	Repeatedly attempt to process a message until successful
Splitter	Split a message using an expression
WireTap	Send a message to an extra message processor as well as to the next message processor in the chain

### All

The All message processor can be used to send the same message to multiple targets.

Configuration for this router is as follows:

```

<all>
  <jms:endpoint queue="test.queue" transformer-refs="StringToJmsMessage" />
  <http:endpoint host="10.192.111.11" transformer-refs="StringToHttpClientRequest" />
  <tcp:endpoint host="10.192.111.12" transformer-refs="StringToByteArray" />
</all>

```

If any of the targets specified is an endpoint that has a filter configured on it, only messages accepted by that filter are sent to that endpoint.

All messages (if any) returned by the targets are aggregated together and form the response from this processor.

## Async

The Async message processor runs a chain of message processors in another thread, optionally specifying a threading profile for the thread to be used. The message processor is configured as follows:

```

<async>
  <append-string-transformer message="-async" />
  <vm:outbound-endpoint path="async-async-out"
    exchange-pattern="one-way" />
  <threading-profile doThreading="true" maxThreadsActive="16" />
</async>

```

This transforms the current message and sends it to the specified endpoint, using a threadpool that contains up to 16 concurrent threads.

## Choice

The Choice message processor sends a message to the first message processor that matches. If none match and a message processor has been configured as "otherwise", the message is sent there. If none match and no otherwise message processor has been configured, an exception is thrown.

Choice is configured as follows:

```

<choice>
  <when expression="payload=='foo'" evaluator="groovy">
    <append-string-transformer message=" Hello foo" />
  </when>
  <when expression="payload=='bar'" evaluator="groovy">
    <append-string-transformer message=" Hello bar" />
  </when>
  <otherwise>
    <append-string-transformer message=" Hello ?" />
  </otherwise>
</choice>

```

If the message payload is "foo" or "bar", the corresponding transformer is run. If not, the transformer specified under "otherwise" is run.

## Collection Aggregator

The Collection Aggregator groups incoming messages that have matching group IDs before forwarding them. The group ID can come from the correlation ID or another property that links messages together.

You can specify the `timeout` attribute to determine how long the router waits in milliseconds for messages to complete the group. By default, if the expected messages are not received by the `timeout` time, an exception is thrown and the messages are not forwarded. You can also set the `failOnTimeout` attribute to `false` to prevent the exception from being thrown and simply forward whatever messages have been received so far.

Configuration for the Collection Aggregator is as follows:

```
<collection-aggregator timeout="6000" failOnTimeout="false" />
```

## Collection Splitter

The Collection Splitter acts on messages whose payload is a Collection type. It sends each member of the collection to the next message processor as separate messages. You can specify the attribute `enableCorrelation` to determine whether a correlation ID is set on each individual message.

```
<collection-splitter enableCorrelation="IF_NOT_SET" />
```

Configuration for the Collection Splitter is as follows:

## Custom Aggregator

A Custom Aggregator is an instance of a user-written class that aggregates messages. This class must implement the interface `MessageProcessor`. Often, it will be useful for it to subclass `AbstractAggregator`, which provides the skeleton of a thread-safe aggregator implementation, requiring only specific correlation logic. As with most custom objects in Mule, it can be configured either with a fully specified class name or as a reference to a Spring bean. It can also be configured with the same `timeout` and `failOnTimeout` attributes described under [Collection Aggregator](#).

Configuration for a Custom Aggregator is as follows:

```
<custom-aggregator timeout="6000" failOnTimeout="false" ref="POaggregator" />
```

or

```
<custom-aggregator failOnTimeout="true" class="com.mycompany.utils.PurchaseOrderAggregator" />
```

## Custom Processor

A Custom Processor is an instance of a user-written class that acts as a message processor. This class must implement the interface `MessageProcessor`. As with most custom objects in Mule, it can be configured either with a fully specified class name or as a reference to a Spring bean.

Configuration for a Custom Processor is as follows:

```
<processor ref="HighSpeedRouter" />
```

or

```
<custom-processor class="com.mycompany.utils.HighSpeedRouter" />
```

## First Successful

The First Successful message processor iterates through its list of child message processors, routing a received message to each of them in order until one processes the message successfully. If none succeed, an exception is thrown.

Success is defined as:

- If the child message processor throws an exception, this is a failure.
- Otherwise:
  - If the child message processor returns a message that contains an exception payload, this is a failure.
  - If the child message processor returns a message that does not contain an exception payload, this is a success.
  - If the child message processor does not return a message (e.g. is a one-way endpoint), this is a success.

This message processor was added in Mule 3.0.1.

```
<first-successful>
  <http:outbound-endpoint address="http://localhost:6090/weather-forecast" />
  <http:outbound-endpoint address="http://localhost:6091/weather-forecast" />
  <http:outbound-endpoint address="http://localhost:6092/weather-forecast" />
  <vm:outbound-endpoint path="dead-letter-queue" />
</first-successful>
```

From 3.1.0 you can further customize the behavior of this router by specifying a '*failureExpression*' that allows you to use Mule Expressions to define a failure. The *failureExpression* attribute is configured as follows:

```
<first-successful failureExpression="exception-type:java.net.SocketTimeoutException">
  <http:outbound-endpoint address="http://localhost:6090/weather-forecast" />
  <http:outbound-endpoint address="http://localhost:6091/weather-forecast" />
  <vm:outbound-endpoint path="dead-letter-queue" />
</first-successful>
```

In the above example a failure expression is being used to more exactly define the exception type that will be considered a failure, alternatively you can use any other Mule expression that can be used with expression filters, just remember that the expression denotes failure rather than success.

## Idempotent Message Filter

An idempotent filter ensures that only unique messages are received by a service by checking the unique message ID of the incoming message. The ID can be generated from the message using an expression defined in the *idExpression* attribute. By default, the expression used is `#[message:id]`, which means the underlying endpoint must support unique message IDs for this to work. Otherwise, a `UniqueIdNotSupportedException` is thrown.

There is a simple idempotent filter implementation provided at [org.mule.routers.IdempotentMessageFilter](#). The default implementation uses a simple file-based mechanism for storing message IDs, but you can extend this class to store the IDs in a database instead by implementing the `ObjectStore` interface.

Configuration for this router is as follows:

```
<idempotent-message-filter idExpression="#[message:id]-#[header:foo]">
  <simple-text-file-store directory=".//idempotent" />
</idempotent-message-filter>
```

The optional *idExpression* attribute determines what should be used as the unique message ID. If this attribute is not used, `#[message:id]` is used by default.

The nested element shown above configures the location where the received message IDs are stored. In this example, they are stored to disk so that the router can remember state between restarts. If the *directory* attribute is not specified, the default value used is  `${mule.working.dir}/objectstore` where `mule.working.dir` is the working directory configured for the Mule instance.

If no store is configured, the `InMemoryObjectStore` is used by default.

## Idempotent Secure Hash Message Filter

This filter ensures that only unique messages are received by a service by calculating the hash of the message itself using a message digest algorithm. This approach provides a value with an infinitesimally small chance of a collision and can be used to filter message duplicates. Note that the hash is calculated over the entire byte array representing the message, so any leading or trailing spaces or extraneous bytes (like padding) can produce different hash values for the same semantic message content. Therefore, you should ensure that messages do not contain extraneous bytes. This router is useful when the message does not support unique identifiers.

Configuration for this filter is as follows:

```
<idempotent-secure-hash-filter messageDigestAlgorithm="SHA256">
  <simple-text-file-store directory=".//idempotent" />
</idempotent-secure-hash-filter>
```

Idempotent Secure Hash Message Filter also uses object stores, which are configured the same way as the Idempotent Message Filter. The optional `messageDigestAlgorithm` attribute determines the hashing algorithm that will be used. If this attribute is not specified, the default algorithm SHA-256 is used.

## Message Chunk Aggregator

After a splitter such as the [Message Chunk Splitter](#) splits a message into parts, the message chunk aggregator router reassembles those parts back into a single message. The aggregator uses the message's correlation ID to identify which parts belong to the same message.

Configuration for the Message Chunk Aggregator is as follows:

```
<message-chunk-aggregator>
  <expression-message-info-mapping correlationIdExpression="#[header:correlation]" />
</message-chunk-aggregator>
```

The optional `expression-message-info-mapping` element allows you to identify the correlation ID in the message using an expression. If this element is not specified, `MuleMessage.getCorrelationId()` is used.

The Message Chunk Aggregator also accepts the `timeout` and `failOnTimeout` attributes as described under [Collection Aggregator](#).

## Message Chunk Splitter

The Message Chunk Splitter allows you to split a single message into a number of fixed-length messages that will all be sent to the same message processor. It will split the message up into a number of smaller chunks according to the `messageSize` attribute that you configure for the router. The message is split by first converting it to a byte array and then splitting this array into chunks. If the message cannot be converted into a byte array, a `RoutingException` is raised.

A message chunk splitter is useful if you have bandwidth problems (or size limitations) when using a particular transport.

To put the chunked items back together again, you can use the [Message Chunk Aggregator](#).

Configuration for the Message Chunk Splitter is as follows:

```
<message-chunk-splitter message-size="512" />
```

## Message Filter

The Message Filter is used to control whether a message is processed by using a [filter](#). In addition to the filter, you can configure whether to throw an exception if the filter does not accept the message and an optional message processor to send unaccepted messages to.

Configuration for the Message Filter is as follows:

```
<message-filter throwOnUnaccepted="false" onUnaccepted="rejectedMessageLogger">
  <message-property-filter pattern="Content-Type=text/xml" caseSensitive="false" />
</message-filter>
```

## Processor Chain

A Processor Chain is a linear chain of message processors which process a message in order. A Processor Chain can be configured wherever a message processor appears in a Mule Schema. For example, to allow a [Wire Tap](#) to transform the current message before sending it off, you can configure the following:

```
<wire-tap>
  <processor-chain>
    <append-string-transformer message="tap" />
    <vm:outbound-endpoint path="wiretap-tap" exchange-pattern="one-way" />
  </processor-chain>
</wire-tap>
```

## Recipient List

The Recipient List message processor allows you to send a message to multiple endpoints by specifying an expression that, when evaluated, provides the list of endpoints. These messages can optionally be given a correlation ID, as in the [Collection Splitter](#). An example is

```
<recipient-list enableCorrelation="ALWAYS" evaluator="header" expression="myRecipients" />
```

which finds the list of endpoints in the message header named `myRecipients`.

## Redelivery Policy

The Redelivery Policy determines what happens when the processing of a message causes an exception, resulting in the message being redeliver over and over. The policy consists of

- A message processor that will be called if the message is redelivered too many times without being processed successfully
- How many redeliveries will be allowed before that occurs. (Default: 5)
- How to identify messages as being "the same"; By default, the message contents are used, as in the [Idempotent Secure Hash Message Filter](#) but, optionally, an expression can be specified, as in the [Idempotent Message Filter](#).

Here is an example of its use:

```
<flow name="processOrders">
    <vm:inbound-endpoint path="orders">
        <vm:transaction action="ALWAYS_BEGIN" />
    </vm:inbound-endpoint>
    <redelivery-policy maxRedeliveryCount="5" >
        <process-failed-message>
            <outbound-endpoint address="vm://dead-letter-queue" />
        </process-failed-message>
    </redelivery-policy>
    <component class="com.mycompany.OrderProcessor" />
</flow>
```

This flow reads messages transactionally from a vm endpoint, and passes them to the order-processing component. Supposes that this component intermittently throws exceptions when resources are unavailable. That will cause the transaction to be aborted and the message reprocessed, which is desirable. But if the same message is reprocessed and rejected five times in a row, on its next redelivery it will be sent to the dead letter queue instead.

## Request Reply

### [Mule 3.2]

The Request Reply message processor receives a message on one channel, allows the back-end process to be forked to invoke other services asynchronously, and accepts the asynchronous result on another channel.

Here is an example that uses the Request Reply message processor:

```
<flow name="main">
    <vm:inbound-endpoint path="input" />
    <request-reply store Prefix="mainFlow" >
        <vm:outbound-endpoint path="request" />
        <vm:inbound-endpoint path="reply" />
    </request-reply>
    <component class="com.mycompany.OrderProcessor" />

    <flow name="request-reply">
        <vm:inbound-endpoint path="request" />
        <component class="come.mycompany.AsyncOrderGenerator" />
    </flow>
</flow>
```

The request is received in the main flow. It then configures an asynchronous request-reply that sends the message to the request-reply flow and implicitly sets the MULE\_REPLYTO message property to `vm://reply`. This tells the request-reply flow where to send its response. The main flow then waits for the response. The request-reply flow receives the message asynchronously. The request is then processed by the `AsyncOrderGenerator` component. When the process is complete, the reply is sent to `vm://reply`. The asynchronous response is received and give to the `OrderProcessor` component to complete the order processing.

## Resequencer

The Resequencer sorts a set of received messages by their correlation sequence property and issues them in the correct order. It uses the `timeout` and `fileOnTimeout` attributes described in [Collection Aggregator](#) to determine when all the messages in the set have been received.

The Resequencer is configured as follows:

```
<resequencer timeout="6000" failOnTimeout="false" />
```

## Round Robin

The Round Robin message processor iterates through a list of child message processors in round-robin fashion: the first message received is routed to the first child, the second message to the second child, and so on. After a message has been routed to each child, the next is routed to the first child again, restarting the iteration.

This message processor was added in Mule 3.0.1.

```
<round-robin>
  <http:outbound-endpoint address="http://localhost:6090/weather-forecast" />
  <http:outbound-endpoint address="http://localhost:6091/weather-forecast" />
  <http:outbound-endpoint address="http://localhost:6092/weather-forecast" />
</round-robin>
```

## Splitter

A Splitter uses an expression to split a message into pieces, all of which are then sent to the next message processor. Like other splitters, it can optionally specify non-`Object` locations within the message for the message ID and correlation ID.

The Splitter is configured as shown below:

```
<splitter evaluator="xpath" expression="//acme:Trade" />
```

This uses the specified XPath expression to find a list of nodes in the current message and sends each of them as a separate message.

## Until Successful

### [Mule 3.2]

The Until Successful message processor processes a message with its child message processor until the processing succeeds. This processing occurs asynchronously, therefore execution is returned to the parent flow immediately.

The Until Successful message processor is able to retry:

- Dispatching to outbound endpoints, for example, when you're reaching out to a remote web service that may have availability issues.
- Execution of a component method, for example, to retry an action on a Spring Bean that may depend on unreliable resources.
- A sub-flow execution, to keep reexecuting several actions until they all succeed.
- Any other message processor execution, to allow more complex scenarios.

```
<until-successful objectStore-ref="objectStore"
                  maxRetries="5"
                  secondsBetweenRetries="60">
  <outbound-endpoint ref="retryableEndpoint" />
</until-successful>
```

This message processor needs an `ListableObjectStore` instance in order to persist messages pending (re)processing. There are several implementations available in Mule, including the following:

- `DefaultInMemoryObjectStore`. The default in-memory store.
- `DefaultPersistentObjectStore`. The default persistent store
- `FileObjectStore`. A file-based store.

- QueuePersistenceObjectStore. The global queue store.
- SimpleMemoryObjectStore. An in-memory store

See [Mule Object Stores](#) for further information about object stores in Mule.

Here is how you would create an in-memory store:

```
<spring:bean id="objectStore" class="org.mule.util.store.SimpleMemoryObjectStore" />
```

Success or failure are defined as:

- If the child message processor throws an exception, this is a failure.
- If the child message processor does not return a message (e.g. is a one-way endpoint), this is a success.
- If a 'failure expression' (see below) has been configured, the return message is evaluated against this expression to determine failure or not.
- Otherwise:
  - If the child message processor returns a message that contains an exception payload, this is a failure.
  - If the child message processor returns a message that does not contain an exception payload, this is a success.

Here is an example showing how to configure the failure expression:

```
<until-successful objectStore-ref="objectStore"
    failureExpression="#[header:INBOUND:http.status = 202]"
    maxRetries="6"
    secondsBetweenRetries="600">
    <http:outbound-endpoint address="http://acme.com/api/flakey"
    exchange-pattern="request-response"
        method="POST" />
</until-successful>
```

The Until Successful message processor is also able to synchronously acknowledge that it has accepted a message and will try to process it repeatedly. The following is an example where the message correlation ID is used as an acknowledgement message:

```
<until-successful objectStore-ref="objectStore"
    ackExpression="#[message:correlationId]"
    maxRetries="3"
    secondsBetweenRetries="10">
    <flow-ref name="signup-flow" />
</until-successful>
```

It is also possible to define a DLQ (dead letter queue) endpoint to which messages will be sent if they have failed processing too many times:

```
<until-successful objectStore-ref="objectStore"
    dlqEndpoint-ref="dlqChannel"
    maxRetries="3"
    secondsBetweenRetries="10">
...
</until-successful>
```

## WireTap

The WireTap message processor allows you to route certain messages to a different message processor as well as to the next one in the chain. For instance, To copy all messages to a specific endpoint, you configure it as an outbound endpoint on the WireTap routing processor:

```
<wire-tap>
    <vm:outbound-endpoint path="tapped.channel"/>
</wire-tap>
```

## Using Filters with the WireTap

The WireTap routing processor is useful both with and without filtering. If filtered, it can be used to record or take note of particular messages or to copy only messages that require additional processing. If filters aren't used, you can make a backup copy of all messages received. The behavior here is similar to that of an interceptor, but interceptors can alter the message flow by preventing the message from reaching the component. WireTap routers cannot alter message flow but just copy on demand. In this example, only messages that match the filter expression are copied to the vm endpoint.

```
<wire-tap>
    <vm:outbound-endpoint path="tapped.channel"/>
    <wildcard-filter pattern="the quick brown*"/>
</wire-tap>
```

Your Rating: 

Results:  3 rates

## Custom Message Processors

### Custom Message Processors

The interface you implement or abstract class you extend will depend what you want your custom message processor to do:

#### **Observe the Message**

Message observers do not mutate the message neither do they determine which Message Processor should be invoked next. The easiest way to implement a message observer is to extend AbstractMessageObserver and implementing the abstract observe method.

Example usage: logging, notifications and statistics etc.

#### **Mutate the Message**

Any MessageProcessor can mutate the message as required. If the Message Processor only mutates the message and is not concerned with determining how the message flow continues, then it makes most sense to implement MessageProcessor rather than the InterceptingMessageProcoessor or MessageRouter interfaces.

Example usage: transformers

#### **Intercept the Message Flow**

Sometime you may need to intercept message flow in order to add some level of indirection to the flow by wrapping the next Message Processor or invoking the next message processor in a particular way. I introduced the InterceptingMessageProcessor in the last blog post and this is what we'll use, but the easiest way to implement a message processor that intercepts flow is by extending AbstractInterceptingMessageProcessor. You still need to implement process but all the rest of the plumbing is done for you and you just need to call processNext when you want to invoke the next MessageProcessor .

Example Usage: exception handling, transactions and asynchronous invocation.

#### **Filter the Message Flow**

In the same way that we may want to intercept the message flow for any of the reason above often you may want to intercept the message flow with the single purpose of filtering, that is to determine if a particular message should continue onto the next message processor or not. You can also implement extend AbstractInterceptingMessageProcessor for this but. If the acceptance criteria can be expressed through the use of a Filter then you can simple use the MessageFilter otherwise you can extend AbstractFilteringMessageProcessor and implement the abstract accept method.

Example Usage: all message payload filters, idempotent filter, security filter

#### **Route the Message**

A MessageRouter has multiple possible routes and is responsible for choosing which route should be used to continue message processing of a particular message. The MessageRouter interface allows multiple routes to be added (or removed).

## Building Message Processors

There is a generic MessageProcessorBuilder interface for building MessageProcessors in the Mule 3 API, this is used internally only for now, but in the future in order to ensure MessageProcessors are created at the correct time users will almost certainly need to implement this interface when configuring some Message Processors.

## Chaining Message Processors

The chaining of Message Processors happens behind the scenes when you configure a <service>, <flow>, endpoint or any other element that allows an ordered list of message processors to be supplied. Rather than force all Message Processors to implement InterceptorMessageProcessor to enable chaining we allow MessageProcessor implements that don't care about the message flow to simply implement MessageProcessor and then when the chain is created these are adapted so as to use them in a chain.

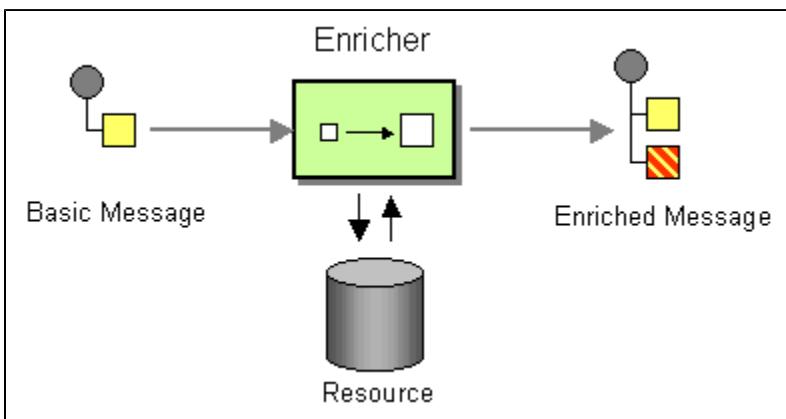
Your Rating:  5 stars

Results:  2.8 7 rates

## Message Enricher

### Message Enricher

One common scenario involves the need to enrich an incoming message with information that isn't provided by the source system. You can use a content enricher if the target system needs more information than the source system can provide.



Consider a message from a source system contains a ZIP code but the target system needs the two letter state. A message enricher can be used to lookup the state using the ZIP code from an enrichment resource. The enricher calls out to the enrichment resource with the current message (containing the zip code) then enriches the current message with the result.

#### Simple Example

```
<flow name="orderProcessingFlow">
    <inbound-endpoint ref="orderEndpoint" />
    <enricher target="#{variable:state}">
        <outbound-endpoint ref="stateLookup" />
    </enricher>
    <outbound-endpoint ref="orderStep2" />
</flow>
```

This is a very simple flow with one-way inbound and outbound endpoints, and which acts as part of an order processing pipeline. This flow uses an enricher to add a state *flow variable* to the current message with the state that the addressLookup endpoint returns. The 'target' attribute defines how the current message is enriched using a MessageEnricher which uses the same syntax as expression evaluators.

NOTE: Mule 3.1 currently supports enrichment of *flow variables* and message headers only.

### More Complex Enrichment

In this particular example the 'addressLookup' endpoint receives the full message, in some cases this might be a generic service that doesn't know how to parse our order message but rather just a ZIP string. It is very easy to improve the configuration to support this, consider the following snippet:

## Example 2

```
<flow name="orderProcessingFlow">
    <inbound-endpoint ref="orderEndpoint" />
    <enricher target="#{variable:address}">
        <outbound-endpoint ref="addressLookup">
            <expression-transformer evaluator="xpath" expression="/order/address/zip" />
        </outbound-endpoint>
    </enricher>
    <outbound-endpoint ref="orderStep2" />
</flow>
```

The “enrichment resource” can be any message processor, outbound endpoint, processor-chain or flow-ref. If using an outbound-endpoint then of course it should have a request-response exchange pattern.

## More Advanced Examples

The <enricher> element also supports more advanced use cases where the message returned by the enrichment resource isn't just a simple string which is exactly what we need to enrich the current message with, often you may want to enrich your message with just part of the information from the result of the invocation of an external service.

For example, if you have a requirement to validate the credit card used for the order as well as add the full address using the ZIP code. The credit card validation process should populate a header call “paymentValidated” with either true or false, this will be used later on. We can easily perform credit card validation by calling out to an authorization service like Authorize.Net to perform this validation, but we have a problem this cloud connector returns more than just a boolean.

The solution is to use the enrichers ‘source’ attribute which will select a value from the result message before using it to enrich the target.

## Example 3

```
<flow name="orderProcessingFlow">
    <inbound-endpoint ref="orderEndpoint" />
    <enricher target="#{variable:paymentValidated}" source="/authorizenet/authorization/@valid">
        <authorizenet:authorize cardNumber="/order/cc/number" />
    </enricher>
    <outbound-endpoint ref="orderStep2" />
</flow>
```

This approach allows you to map content in the response message from the enrichment resource to the current message. If you have a more advanced use case you can also map n values in the enrichment resource response to m values enriched in the current message, this is done using child <enrich> elements each of which has source and target attributes. Note: if you use child <enrich> elements the source/target attributes on <enricher> are not allowed.

## Example 4

```
<flow name="orderProcessingFlow">
    <inbound-endpoint ref="orderEndpoint" />
    <enricher>
        <authorizenet:authorize cardNumber="/order/cc/number" />
        <enrich target="#{variable:paymentValidated}" source="/authorizenet/authorization/@valid" />
        <enrich target="#{variable:paymentAuthCode}" source="/authorizenet/authorization/code" />
    </enricher>
    <outbound-endpoint ref="orderStep2" />
</flow>
```

## Reference Information on Enricher

### Variables and Enrichers

You can find information on using a variable with any enricher listed on these pages:

Default enrichers are loaded at runtime.

Schema documentation for enricher element.

API documents: [org.mule.core.enricher.MessageEnricher](#)

Your Rating: 

Results:  0 rates

## Logger Element for Flows

### Logger Element for Flows

Mule has always supported flows with <log-component>, but as of Mule 3.1, a new implementation called <logger> has been baked into the message processor for use anywhere within a flow.

#### Where Can <logger> Be Used?

- Anywhere in a <flow>
- Within inbound and outbound endpoint elements (or within their optional <response> elements)
- Within the <inbound> section of a <service> after the endpoints
- Within the <asynch-reply> section of a <service> after the endpoints

#### How Do I Use <logger> at a Certain Point in a Flow or a Service?

Add <logger> at the point where you want to log the message:

```
<flow>
    <inbound-endpoint ref="myInboundEndpoint" />
    <logger message="Hello from my Flow!" />
</flow>
```

This will log the message specified using the default DEBUG level, so you will see this in your log only if you have your log level set at DEBUG or TRACE level.

#### How Does <logger> Work Using WARN/ERROR?

Add the 'level' attribute. An XML editor should show you the permissible values: ERROR, WARN, INFO, DEBUG and TRACE.

This example uses the a wire tap configured with an expression filter to log our message using the ERROR category only when the incoming payload does not contain a certain string.

```
<flow>
    <inbound-endpoint ref="myInboundEndpoint" />
    <wiretap>
        <logger message="Oops we have an invalid message!" level="ERROR" />
        <expression-filter evaluator="groovy" expression="!payload.contains('valid message')" />
    </wiretap>
</flow>
```

#### How Can I Log the Entire Message Payload?

In order to log your message payload, headers, the whole message or a single value from within your payload, <logger> leverages Mule Expressions. Rather than specifying a single expression for logging you can embed as many expressions as you required in your log 'message' as required. This allows you to give some context to what is being logged, and enables you to log multiple things at once.

```
<flow>
    <inbound-endpoint ref="myInboundEndpoint" />
    <wiretap>
        <logger message="Oops! #[payload] is an invalid message!" level="ERROR" />
        <expression-filter evaluator="groovy" expression="!payload.contains('valid message')" />
    </wiretap>
</flow>
```

## Modifying and Expanding <logger>

Because you can use [Mule Expressions](#), you can endlessly vary the output of your log messages. You can also configure <loggers> to use your own category instead of using the givens within [LoggerMessageProcessor](#).

Your Rating:  Results:  0 rates

# Configuring Components

## Configuring Components

[ [Simple Components](#) ] [ [Java Components](#) ] [ [Other Components](#) ] [ [Customizing Behavior with Interceptors](#) ] [ [Lifecycle](#) ]

Service components contain the business logic for working with the messages passed through Mule ESB. A service component can be any type of object, including a [Spring bean](#), [POJO](#), [script](#), [web service](#), or [REST call](#).

Because they are highly specific to your implementation, you will typically [create your own custom components](#), or simply use an existing POJO. Mule also ships with some standard components you can use or extend as needed. This page describes how to configure the different types of components.

For detailed information on the elements you configure for components, see [Component Configuration Reference](#).

## Simple Components

There are several simple components included with Mule that are useful for testing or bypassing component execution.

Configuration Element	Description
<log-component/>	Logs component invocations, outputting the received message as a string. This component does not return a response.
<echo-component/>	Extends the log component to log and echo the incoming message. The message is transformed before being returned, so transformations on inbound endpoints will be applied.
<null-component/>	Throws an exception when invoked. This is useful for testing use cases that use a forwarding consumer inbound router.
<passthrough-component>	Similar to the echo component but does not log. This component is useful when defining services that consist of inbound and outbound endpoints/routers but don't have a component implementation. Note that explicitly configuring this component has exactly the same result as configuring a service with no component.
<bridge-component/>	Identical to the pass-through component but preserves the Mule 1.4 terminology.
<test:component/>	Configures the Mule <code>FunctionalTestComponent</code> , which allows more complex testing scenarios to be created. For more information, see <a href="#">Functional Testing</a> .

## Java Components

Java components specify a Java class to be used as the service component or configure a reference to an implementation in a container such as Spring. They also configure the way in which Mule should manage the Java service component's life-cycle, invoke it, and (if pooled) manage the pool of instances.

Java components can be configured quickly and easily by simply specifying the service component implementation class name on the `<component>` or `<pooled-component>` element. The `<pooled-component>` element allows you to establish a pooling profile for the service (see [Tuning Performance](#)). In both cases, the `PrototypeObjectFactory` will be used by default and a new object instance will be created for each request or (for pooled components) for each new object in the pool.

```
<component class="org.my.ServiceComponentImpl" />
...
<pooled-component class="org.my.ServiceComponentImpl" />
```

Alternatively, you can explicitly specify object factories, such as the `SingletonObjectFactory` that creates a single instance of the object:

```

<component>
    <singleton-object class="org.my.ServiceComponentImpl" />
</component>

```

The explicit syntax is required instead of the shortcut `<component class>` syntax if you add interceptors to the component. For further configuration options and information on the default settings that are applied, see [Configuring Java Components](#).

## Other Components

These are several other components available that allow you to use different technologies such as [web services](#) for your service components. These components are often included as part of [transports](#) or [modules](#).

Configuration	Description
<code>&lt;http:rest-service-component/&gt;</code>	Proxies a remote call to a REST-style web service.
<code>&lt;cxf:wrapper-component/&gt;</code>	Proxies a remote call to a web service using CXF.
<code>&lt;script:component/&gt;</code>	Configures a JSR-223 <a href="#">script</a> for the service component.

## Customizing Behavior with Interceptors

Mule interceptors are useful for attaching behaviors to multiple service components. The interceptor pattern is often referred to as practical AOP (Aspect Oriented Programming), as it allows the developer to intercept processing on an object and potentially alter the processing and outcome. For complete information, see [Using Interceptors](#).

## Lifecycle

Components have a lifecycle like any other object in the Mule registry. Lifecycle can be configured by adding one or more lifecycle interfaces to your component. Since Mule 3.0 [JSR-250](#) annotations can be used to configure initialise and destroy methods. The following list describes in order the lifecycle phases for component objects.

Lifecycle	Description	Interface	Annotation
initialise	The first lifecycle method called once any injectors on the component have been called. This means any properties on the component will be set before the initialise lifecycle is called.	<code>org.mule.api.lifecycle.Initialisable</code>	<code>javax.annotation.PostConstruct</code>
start	This is called when the MuleContext is started.	<code>org.mule.api.lifecycle.Startable</code>	
stop	This is called when the MuleContext is stopped, or the service that owns this component is stopped.	<code>org.mule.api.lifecycle.Stoppable</code>	
dispose	Called as the object is being disposed off. Typically this happens because either the MuleContext is shutting down or the service that wraps this component was unregistered.	<code>org.mule.api.lifecycle.Disposable</code>	<code>javax.annotation.PreDestroy</code>

## Lifecycle examples

### Full lifecycle:

```

org.my.ServiceComponentImpl implements org.my.ServiceComponent, org.mule.api.lifecycle.Lifecycle
{
    public void initialise() throws InitialisableException { }

    public void start() throws MuleException { }

    public void stop() throws MuleException { }

    public void dispose() { }
}

```

#### **Initialise-only lifecycle:**

```

org.my.ServiceComponentImpl implements org.my.ServiceComponent, org.mule.api.lifecycle.Initialisable
{
    public void initialise() throws InitialisableException { }
}

```

#### **Initialise/Dispose lifecycle using JSR-250 annotations:**

```

org.my.ServiceComponentImpl implements org.my.ServiceComponent
{
    @PostConstruct
    public void init() { }

    @PreDestroy
    public void destroy() { }
}

```

Your Rating: 

Results:  1 rates

## Configuring Java Components

### Configuring Java Components

[ Object Factories ] [ Entry Point Resolvers ] [ Lifecycle Adapter Factory ] [ Bindings ] [ Configuring a Pooled Java Component ]

Java is the default component type in Mule. Mule provides two **JavaComponent** implementations: **DefaultJavaComponent**, which you configure with the `component` element, and **PooledJavaComponent**, which adds pooling functionality and which you configure with the `pooled-component` element. These two implementations provide the following functionality and configuration options:

- An **ObjectFactory** is used to obtain the Java service component implementation.
- **EntryPointResolvers** can be configured to define how Mule services should invoke the component methods when processing a message.
- A custom **LifecycleAdaptor** can be configured to customize the way in which the component implementation is initialized and disposed.
- **Bindings** can be configured to bind component interface methods to endpoints. These endpoints are then invoked synchronously when the method is called.

When you specify the class directly on the `component` or `pooled-component` element, the `PrototypeObjectFactory` is used by default, and a new instance is created for each invocation, or a new pooled component is created in the case of the `PooledJavaComponent`.

Example:

```

<component class="org.my.CustomComponent" />
...
<pooled-component class="org.my.CustomComponent" />

```

Alternatively, you can specify the implementation using an object factory.

Example:

```
<component>
    <singleton-object class="org.my.CustomComponent" />
</component>
...
<component>
    <spring-object bean="myCustomComponentBean" />
</component>
```

All other component configuration elements are configured as children of the `component` or `pooled-component` element.

Note: In Mule 2.0, Java component pooling is used only if the `<pooled-component>` element is used. In previous versions of Mule, pooling was the default.

## Object Factories

Object factories manage both object creation in the case of a Mule instantiated instance or object look-up from another container such as Spring via a single API. The following object factories are included with Mule and can be configured using Mule's core schema.

<code>&lt;prototype-object class=.../&gt;</code>	PrototypeObjectFactory
<code>&lt;singleton-object class=.../&gt;</code>	SingletonObjectFactory
<code>&lt;spring-object bean=.../&gt;</code>	SpringBeanLookup

Object factories also allow you to set properties, which are injected when a new object instance is created.

Example:

```
<component>
    <singleton-object class="org.my.SingletonObject">
        <property key="myKey" value="theValue"/>
        <property key="myKey2" value="theValue2"/>
    </singleton-object>
</component>
```

For a real-world example of using `<spring-object/>`, see [Using Spring Beans as Service Components](#).

You can easily implement additional object factories to integrate with other containers or simply to create object instances in a different way.

Note: Object factories replace ContainerContexts in previous versions of Mule.

## Entry Point Resolvers

You can configure entry point resolvers that determine how your component is invoked when a message is received by the service. See [Developing Components](#) for a more detailed description of their functionality.

To configure entry point resolvers, you can either configure an entry point resolver `set` or configure a single entry point resolver independently. When using an entry point resolver `set`, the order in which the resolvers are configured is the order of precedence they are given in run-time.

Example:

```
<component class="org.my.PrototypeObjectWithMyLifecycle">
    <entry-point-resolver-set>
        <array-entry-point-resolver/>
        <callable-entry-point-resolver/>
    </entry-point-resolver-set>
</component>
```

```
<component class="org.my.PrototypeObjectWithMyLifecycle">
    <reflection-entry-point-resolver/>
</component>
```

You can also configure entry point resolvers (single or sets) on models to apply them to all services defined in that model. You use the same configuration syntax as above but on the `<model>` element instead of `<component>`.

## Lifecycle Adapter Factory

You can configure your Java component to use a custom lifecycle adaptor. If you do not configure a custom implementation, the default implementation will be used, which allows the optional propagation of Mule's lifecycle to your component depending on the Mule lifecycle interfaces that are implemented.

Example:

```
<component class="org.my.PrototypeObjectWithMyLifecycle">
    <custom-lifecycle-adapter-factory class="org.my.MyLifecycleMuleAdapterFactory"/>
</component>
```

See [Developing Components](#) for more information about lifecycles.

## Bindings

Components can use bindings to call an external service during execution. The bindings used with a Java component bind a Java interface, or single interface method, to an outbound endpoint. The external service to be called should implement the same interface, and the component should encapsulate a reference to that interface, which is initialized during the bootstrap stage by the Mule configuration builder. The reference will be initialized using a reflective proxy class.

Binding can be used on Java components and script components. For more information see [Component Bindings](#).

## Configuring a Pooled Java Component

A pooled Java component will maintain a pool of object instances that will be reused, with a single instance being used by one thread at any one time. The configuration of component pooling is independent of the object factory, allowing you to use whichever object factory you need.

You configure the pool using the nested `pooling-profile` element as shown below:

```
<pooled-component class="org.my.PrototypeObject">
    <pooling-profile exhaustedAction="WHEN_EXHAUSTED_FAIL" initialisationPolicy="INITIALISE_ALL"
maxActive="1" maxIdle="2" maxWait="3" />
</pooled-component>
```

For more information about pooling and reference documentation for pooling configuration options, see [Tuning Performance](#).

Your Rating: 

Results:  0 rates

## Developing Components

### Developing Components

[ [Entry Point](#) ] [ [Default Message Flow Behavior](#) ] [ [Customizing the Message Flow Behavior](#) ] [ [Component Lifecycle](#) ]

When developing service components, you focus on developing code to handle the business logic, not to integrate the service component with Mule ESB. For example, if you are developing a service component that adds customer data to an invoice, you focus on writing code that queries the customer database and updates the invoice as needed. You do not have to write any special code to handle the message that's passed to the service component or to integrate with Mule, as all integration is handled through configuration. You can develop the service component as a POJO, or as a web service using popular containers such as Spring, EJB, or Plexus.

Mule does allow you to enable service components to obtain information about or even control the current message instead of just working with the message payload. To enable the service component to work directly with the message, you must implement the `Callable` interface in the

service component (see Entry Points below).

To get started with developing Mule service components, you can use the Mule IDE. The Mule IDE is an Eclipse plug-in that provides an integrated development environment for developing Mule applications. You can also use the [standard components](#) provided with Mule, or use them as a starting point for building your own.

## Entry Point

The *entry point* is the method in your component that is invoked by Mule when a message is received. To specify the method explicitly on your endpoint, you can use the `method` argument on the endpoint, such as:

```
<outbound-endpoint address="ejb://localhost:1099/SomeService?method=remoteMethod" />
```

or

```
<ejb:endpoint host="localhost" port="1099" object="SomeService" method="remoteMethod" />
```

If you do not specify this argument, Mule uses an *entry point resolver* to dynamically choose the method to invoke based on the payload type of the message. When the match is found, the method is cached with the parameter types so that introspection is only done once per method for the life of the service. If multiple methods in the component match the payload type, or no method matches, an error is thrown. You can also call a method on the component that has no arguments.

Alternatively, your component can implement the [org.mule.api.lifecycle.Callable](#) interface. If your component implements this interface, it will override any dynamic resolution and call the interface method implementation instead.

For details on configuring entry point resolvers, see [Entry Point Resolver Configuration Reference](#).

## Legacy Entry Point Resolver Set

The LegacyEntryPointResolverSet is used if no other resolver is configured. The LegacyEntryPointResolverSet provides generic entry point resolution as follows:

1. Check the methods of the service component's class for annotations (see [org.mule.impl.model.resolvers.AnnotatedEntryPointResolver](#) ).
2. Use the "method" attribute as described above, if one is specified (see [org.mule.model.resolvers.MethodHeaderPropertyEntryPointResolver](#) ).
3. If the component implements the [org.mule.api.lifecycle.Callable](#) lifecycle interface, use the `onCall(MuleEventContext)` method to receive the message.
4. The message type will be matched against methods on the component to see if there is a method that accepts the transformer return type. If so, this method will be used. Note if there is more than one match, an exception will be thrown.
5. If none of the above finds a match, an exception will be thrown and the component registration will fail.

There are many scenarios where the LegacyEntryPointResolverSet is unsuitable. More control is available by extending its set of implementations, or by configuring a completely new set. There are several EntryPointResolver implementations, such as [org.mule.model.resolvers.CallableEntryPointResolver](#) , [org.mule.model.resolvers.MethodHeaderPropertyEntryPointResolver](#) , and [org.mule.model.resolvers.ReflectionEntryPointResolver](#) . While these are used in the LegacyEntryPointResolverSet, they can be more comprehensively configured when specified separately.

## Calling No-arguments Methods

If you want to call a method with no arguments, you can use the [org.mule.model.resolvers.NoArgumentsEntryPointResolver](#) . Regardless of the payload of the current message, this resolver looks only for no-argument methods in the component. Additionally, [ReflectionEntryPointResolver](#) supports the resolution of no-argument service methods if the payload received is of type NullPayload.

## Custom Entry Point Resolver

If you want to create your own entry point resolver, you create a class that implements the [EntryPointResolver](#) interface and specify it with the `<custom-entry-point-resolver>` element in your Mule configuration.

## Default Message Flow Behavior

Mule has some default behavior rules about managing message flow to and from your component.

1. When a message is received, the entry point method is invoked as described above.
2. The response or outbound message is obtained as follows:
  - If the method invoked is not void, (that is, `Callable.onEvent()` returns an Object), the method return value is used. If null is returned, no further processing is done for the current request.

- If the method is void, the parameters used to invoke the method are used. This assumes that the parameters themselves were altered or there was no change to the message.
3. If the inbound endpoint has an exchange pattern configured that returns a response then the result of the component invocation is returned to caller.
  4. The outbound message is then routed according the services <outbound> configuration:

See [Service Messaging Styles](#) for more detail about the different configuration options that affect message flow.

## Customizing the Message Flow Behavior

To customize the message flow behavior, you must get a reference to `org.mule.api.MuleEventContext`. You can get the reference by implementing the Callable interface, which passes the event context as a parameter on this interface:

```
public interface Callable
{
    public Object onCall(MuleEventContext eventContext) throws Exception;
}
```

From the MuleEventContext, you can send and receive events synchronously and asynchronously, manage transactions, and override the default event flow behavior. For example:

```
MuleEventContext context = RequestContext.getEventContext();
OutboundEndpoint endpoint = ...;

//to send asynchronously
context.dispatchEvent(new MuleMessage("IBM:0.01", null), endpoint);

//or to request
InboundEndpoint endpoint = ...
MuleMessage quote = context.requestEvent(endpoint, 5000);
```

Even when you use the event context to manually control event flow, when your method returns, Mule will route the outbound event as normal. You can stop Mule processing events further as follows:

- If your service method is not void, you can return null. This approach tells Mule there is no further event information to process.
- If your service method is void, Mule will use the inbound message payload as the outbound message payload. You can override this behavior using the `setStopFurtherProcessing` method as described below.

## Halting Message Flow

To halt the message flow, you can either call `setStopFurtherProcessing()` from the `MuleEventContext` or else throw an exception. This will cause the `ExceptionStrategy` on the component to be invoked.

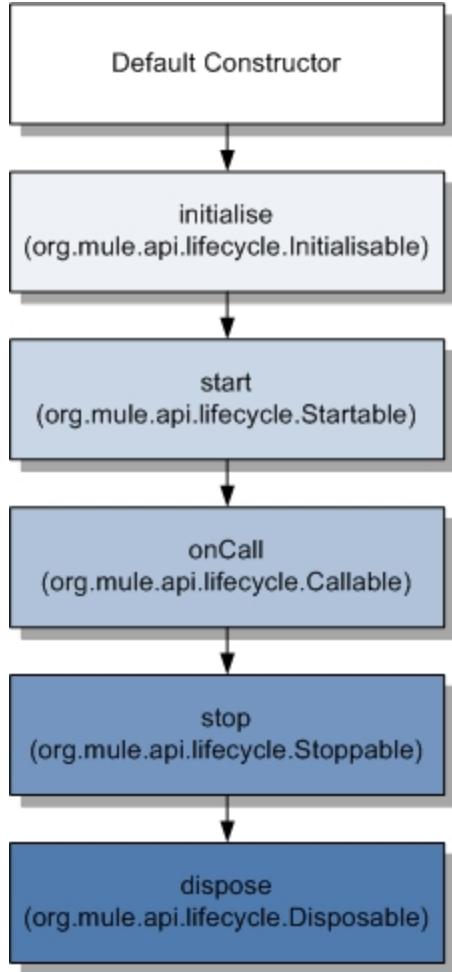
### Note:

The use of additional services or the use of component bindings is much preferred to the above techniques to control message flow from within your component implementation. This is because it allows for a much more decoupled implementation that can be modified via your configuration file and avoids the need to use Mule API in your component implementations. To take this approach, do one of the following:

- Ensure your service components are implemented in such a way that they do a single unit of work that do not need to do any message sending/receiving. This additional sending/receiving/routing is then done using Mule services.
- Design your component in such a way that interface methods can be mapped to outbound endpoints and then use bindings to map these in configuration. For information on how to configure bindings, see [Configuring Java Components](#).

## Component Lifecycle

Your component can implement several lifecycle interfaces. The lifecycle flow typically looks like this, with `onCall()` often being replaced by an entry point resolver as described above:



Following are the most commonly used interfaces:

- `org.mule.api.lifecycle.Initialisable` is called only once for the lifecycle of the component. It is called when the component is created when the component pool initializes.
- `org.mule.api.lifecycle.Startable` is called when the component is started. This happens once when the server starts and whenever the component is stopped and started either through the API or JMX.
- `org.mule.api.lifecycle.Stoppable` is called when the component is stopped. This happens when the server stops or whenever the component is stopped either through the API or JMX.
- `org.mule.api.lifecycle.Disposable` is called when the component is disposed. This is called once when the server shuts down.

For more information, see the [org.mule.api.lifecycle Javadocs](#).

Your Rating:

Results: 2 rates

## Entry Point Resolver Configuration Reference

### Entry Point Resolver Configuration Reference

This page provides details on the elements you configure for entry point resolvers and entry point resolver sets. This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page.

#### **Entry Point Resolver Sets**

cache: Unexpected program error: java.lang.NullPointerException

#### **Entry point resolver set**

An extensible set of entry point resolvers. These determine how a message is passed to a component in Java. Each entry point resolver is tried in turn until one succeeds in delivering the message to the component. This element can be set on the model or component; the model value provides a default that individual component values can override.

#### Attributes of <entry-point-resolver-set...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of <entry-point-resolver-set...>

Name	Cardinality	Description
abstract-entry-point-resolver	0..*	A placeholder for an entry point resolver element. Entry point resolvers define how payloads are delivered to Java code by choosing the method to call.

cache: Unexpected program error: java.lang.NullPointerException

#### **Legacy entry point resolver set**

An extensible set of entry point resolvers (which determine how a message is passed to a component in Java) that already contains resolvers to implement the standard logic. This is already provided by default and is only needed explicitly if it will be extended with other entry point resolvers. This element can be set on the model or component; the model value provides a default that individual component values can override.

#### Attributes of <legacy-entry-point-resolver-set...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of <legacy-entry-point-resolver-set...>

Name	Cardinality	Description
abstract-entry-point-resolver	0..*	A placeholder for an entry point resolver element. Entry point resolvers define how payloads are delivered to Java code by choosing the method to call.

cache: Unexpected program error: java.lang.NullPointerException

#### **Custom entry point resolver set**

A custom entry point resolver set. This allows user-supplied code to determine how a message is passed to a component in Java. This element can be set on the model or component; the model value provides a default that individual component values can override.

#### Attributes of <custom-entry-point-resolver-set...>

Name	Type	Required	Default	Description
class	class name	yes		An implementation of the EntryPointResolverSet interface.

#### Child Elements of <custom-entry-point-resolver-set...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

#### **Entry Point Resolvers**

cache: Unexpected program error: java.lang.NullPointerException

#### **Callable entry point resolver**

An entry point resolver for components that implement the Callable interface. This passes a MuleEventContext to the component. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

#### Attributes of <callable-entry-point-resolver...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of <callable-entry-point-resolver...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Custom entry point resolver**

A custom entry point resolver. This allows user-supplied code to determine how a message is passed to a component in Java. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

#### **Attributes of <custom-entry-point-resolver...>**

Name	Type	Required	Default	Description
class	class name	yes		An implementation of the EntryPointResolver interface.

#### **Child Elements of <custom-entry-point-resolver...>**

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### **Property entry point resolver**

Uses a message property to select the component method to be called. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

#### **Attributes of <property-entry-point-resolver...>**

Name	Type	Required	Default	Description
acceptVoidMethods	boolean	no		Whether the resolver should call void methods. By default, void methods are not considered as possible candidates for message delivery.
property	name (no spaces)	no		The name of the message property used to select a method on the component.

#### **Child Elements of <property-entry-point-resolver...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Method entry point resolver**

Delivers the message to a named method. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

#### **Attributes of <method-entry-point-resolver...>**

Name	Type	Required	Default	Description
acceptVoidMethods	boolean	no		Whether the resolver should call void methods. By default, void methods are not considered as possible candidates for message delivery.

#### **Child Elements of <method-entry-point-resolver...>**

Name	Cardinality	Description
include-entry-point	1..*	A possible method for delivery.

cache: Unexpected program error: java.lang.NullPointerException

## **Reflection entry point resolver**

Generates a list of candidate methods from the component via reflections. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

### **Attributes of <reflection-entry-point-resolver...>**

Name	Type	Required	Default	Description
acceptVoidMethods	boolean	no		Whether the resolver should call void methods. By default, void methods are not considered as possible candidates for message delivery.

### **Child Elements of <reflection-entry-point-resolver...>**

Name	Cardinality	Description
exclude-object-methods	0..1	If specified, methods in the Java Object interface are not included in the list of possible methods that can receive the message.
exclude-entry-point	0..*	Explicitly excludes a named method from receiving the message.

cache: Unexpected program error: java.lang.NullPointerException

## **Array entry point resolver**

Delivers the message to a method that takes a single array as argument. This element can be set on the model or component; the model value provides a default that individual component values can override. This element can also be used directly or as part of a set of resolvers; the resolvers in a set are used in turn until one is successful.

### **Attributes of <array-entry-point-resolver...>**

Name	Type	Required	Default	Description
acceptVoidMethods	boolean	no		Whether the resolver should call void methods. By default, void methods are not considered as possible candidates for message delivery.
enableDiscovery	boolean	no	true	If no method names are configured, attempts to discover the method to invoke based on the inbound message type.

### **Child Elements of <array-entry-point-resolver...>**

Name	Cardinality	Description
exclude-object-methods	0..1	If specified, methods in the Java Object interface are not included in the list of possible methods that can receive the message.
exclude-entry-point	0..*	Explicitly excludes a named method from receiving the message.
include-entry-point	0..*	A possible method for delivery.

cache: Unexpected program error: java.lang.NullPointerException

## **No arguments entry point resolver**

Calls a method without arguments (the message is not passed to the component).

### **Attributes of <no-arguments-entry-point-resolver...>**

Name	Type	Required	Default	Description
acceptVoidMethods	boolean	no		Whether the resolver should call void methods. By default, void methods are not considered as possible candidates for message delivery.
enableDiscovery	boolean	no	true	If no method names are configured, attempts to discover the method to invoke based on the inbound message type.

### **Child Elements of <no-arguments-entry-point-resolver...>**

Name	Cardinality	Description
exclude-object-methods	0..1	If specified, methods in the Java Object interface are not included in the list of possible methods that can receive the message.
exclude-entry-point	0..*	Explicitly excludes a named method from receiving the message.
include-entry-point	0..*	A possible method for delivery.

cache: Unexpected program error: java.lang.NullPointerException

### ***Include entry point***

A possible method for delivery.

#### **Attributes of <include-entry-point...>**

Name	Type	Required	Default	Description
method	name	no		The name of the method.

#### **Child Elements of <include-entry-point...>**

Name	Cardinality	Description

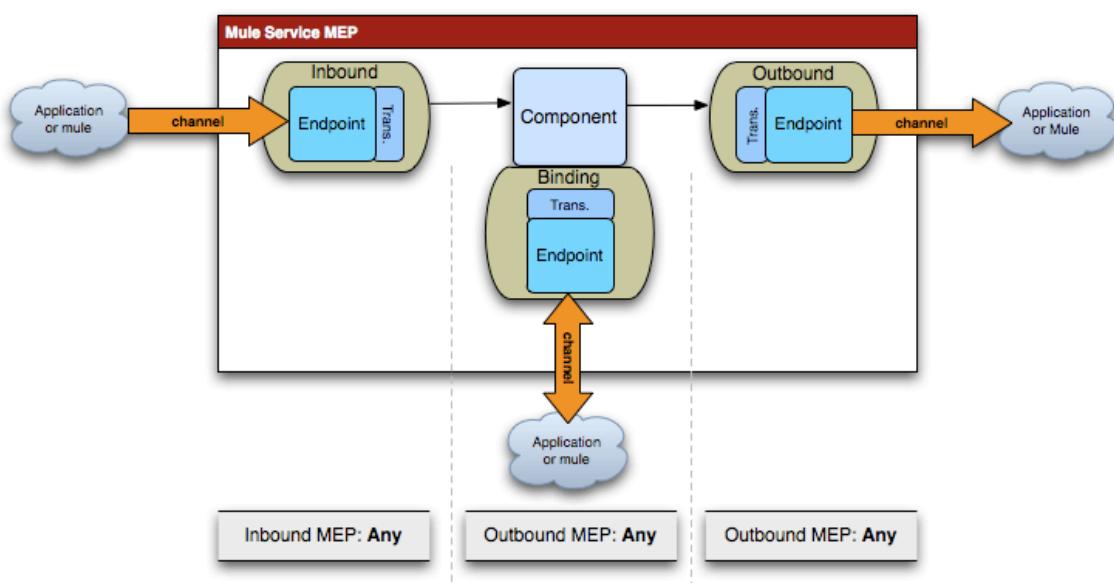
Your Rating: ★★★★☆ Results: ★★★★★ 0 rates

## **Component Bindings**

### **Component Bindings**

[ Java Component Binding Configuration ] [ Script Component Bindings ]

Components can use bindings to call an external service during execution. The bindings used with a Java component bind a Java interface, or single interface method, to an outbound endpoint. The external service to be called should implement the same interface, and the component should encapsulate a reference to that interface, which is initialized during the bootstrap stage by the Mule configuration builder. The reference will be initialized using a reflective proxy class.



With component bindings, you can configure multiple interfaces or a single interface with an endpoint bound to one or more Mule endpoints.

Mule currently supports component bindings for Java components (the default components in Mule) and script components, such as Groovy or JRuby. This page describes how to configure each.

### **Java Component Binding Configuration**

Bindings can be used by components to call out to an external service. The bound interface is added to the component as a field with the usual bean getter and setter methods. In the binding configuration for the component, you can bind this interface along with a method in the interface to a Mule endpoint. When that method is called, the call parameters are sent over the Mule endpoint to another service, which may be local or remote. A result may be returned from the service and passed back to the component using the return argument of the method. This model is very similar to traditional RPC calls. Here is an example:

```
public class InvokerComponent
{
    private HelloInterface hello;

    public String invoke(String s)
    {
        return "Received: " + hello.sayHello(s, "English");
    }
    public void setHello(HelloInterface hello)
    {
        this.hello = hello;
    }
    public HelloInterface getHello()
    {
        return hello;
    }
}
```

In this example, the component `InvokerComponent` has a field `hello`, which is of type `HelloInterface`, with getter and setter methods for the field. The `invoke` method will be called on the service and calls the `hello.sayHello()` method. This call will result in another service call.

The `HelloInterface` is very simple with a single method `sayHello`.

```
public interface HelloInterface
{
    public String sayHello(String to, String language);
}
```



As of Mule 3.x, you are able to set the interface return type to `MuleMessage` to give your component access to the full message, not just the payload.

Now, you simply configure your component to bind the `sayHello` method to the endpoint that will invoke another service.

```
<service name="InvokerComponent">
    <inbound>
        <jms:inbound-endpoint queue="Invoker.in"/>
    </inbound>

    <component class="org.mule.examples.bindings.InvokerComponent">
        <binding interface="org.mule.examples.bindings.HelloInterface" method="sayHello">
            <axis:outbound-endpoint use="LITERAL" style="WRAPPED"
                address="http://myhost.com:81/services/HelloWeb?method=helloMethod" synchronous=
            "true"/>
        </binding>
    </component>

    <outbound>
        <pass-through-router>
            <jms:outbound-endpoint queue="Invoker.out"/>
        </pass-through-router>
    </outbound>
</service>
```

The call to the external web service is synchronous, because you want a result returned and need to block until the call is finished.

Note that component bindings will not work with Java Proxy objects, unless the proxy explicitly handles the binding interface method. If using Spring components, ensure that you use CGLib proxies. For more information on the Spring-AOP proxying behavior, see <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/aop.html#aop-introduction-proxies>. If you're using the annotation-processors, such as for transactions, you would specify the following:

```
<tx:annotation-driven proxy-target-class="true" />
```

## Handling Data Types

You can handle data conversion when making a call and receiving a result using the normal transformer configuration on the endpoint. In the above example, assume the web service was expecting an `org.mule.examples.bindings.WebHelloRequest` object and returned an `org.mule.examples.bindings.WebHelloResponse` object. You don't want your component to know about these external data types, so you can configure transformers to do the conversion when the call is made:

```
<component class="org.mule.examples.bindings.InvokerComponent">
    <binding interface="org.mule.examples.bindings.HelloInterface" method="sayHello">
        <axis:outbound-endpoint use="LITERAL" style="WRAPPED"
            address="http://myhost.com:81/services/HelloWeb?method=helloMethod" synchronous=
            "true">
            <transformers>
                <custom-transformer class="org.mule.examples.bindings.StringsToWebRequest"/>
            </transformers>
            <response-transformers>
                <custom-transformer class="org.mule.examples.bindings.WebResponseToString"/>
            </response-transformers>
        </axis:outbound-endpoint>
    </binding>
</component>
```

## Exceptions

If the remote service call triggers an exception or fault, this exception will get serialized back to the local service call and thrown. If your service wants to handle this exception, you must add the exception (or `java.lang.Exception`) to the bound method signature and use a try catch block as usual.



Currently, the `invoke()` method of `org.mule.routing.binding.BindingInvocationHandler` always throws an `UndeclaredThrowableException` when there is an exception, even when the bound method/interface throws one of its declared exceptions. As of Mule 3.x, the declared exception will be returned if the cause is known and the type matches one of the exceptions declared in the given method's "throws" clause.

## Script Component Bindings

Similar to the Java component bindings, script bindings enable the same behavior for your scripting components. When using a scripting component, the binding is bound to the scripting context and is accessible by using the binding interface class name.

```

<service name="ScriptService">
  <inbound>
    <vm:inbound-endpoint ref="inboundEndpoint" />
  </inbound>
  <script:component>
    <script:script engine="groovy">
      return "Total: " + AdditionService.add(1,2)
    </script:script>
    <script:java-interface-binding interface="org.mule.tck.services.AdditionService" method="add">
      <vm:outbound-endpoint path="addition.service" synchronous="true" />
    </script:java-interface-binding>
  </script:component>
  <outbound>
    <pass-through-router>
      <vm:outbound-endpoint ref="receivedEndpoint" />
    </pass-through-router>
  </outbound>
</service>

```

The implementation for the component is contained within the `<script:script>` element:

```
return "Total: " + AdditionService.add(1,2)
```

We refer to the binding interface using the short class name `AdditionService` and invoke the `add` method, which will call a local addition service.

Your Rating:  Results:  1 rates

## Using Interceptors

### Using Interceptors

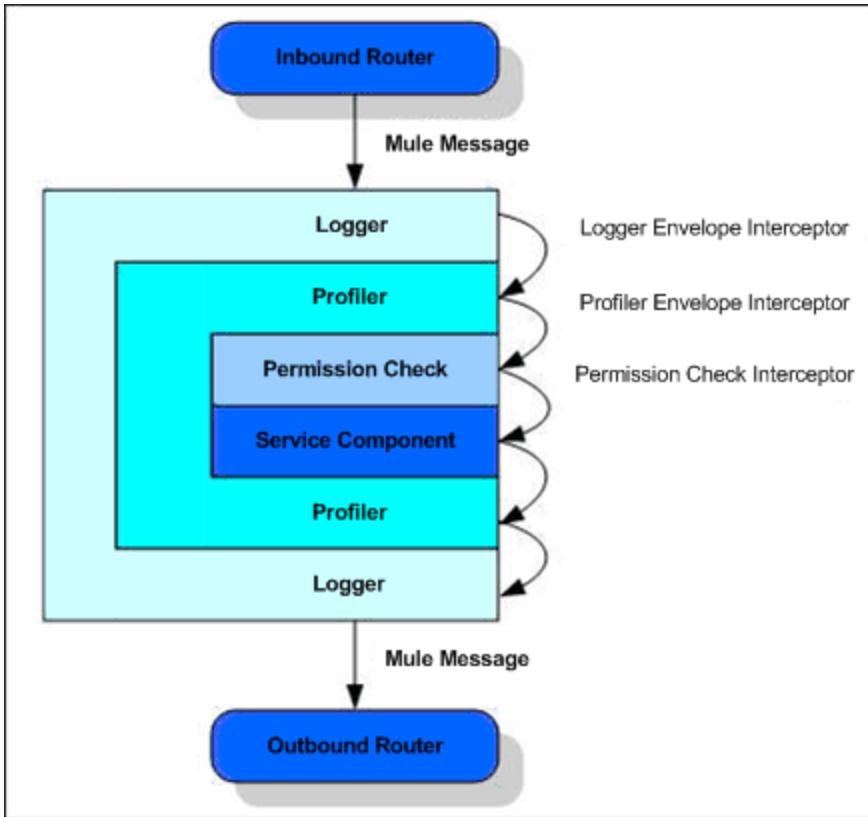
[ [Interceptor Event Flow](#) ] [ [Writing Interceptors](#) ] [ [Configuring Interceptors](#) ] [ [Interceptor Configuration Reference](#) ]

Mule ESB interceptors are useful for attaching behaviors to multiple service components. The interceptor pattern is often referred to as practical AOP (Aspect Oriented Programming), as it allows the developer to intercept processing on an object and potentially alter the processing and outcome. (See also [Spring AOP](#)). Interceptors are very useful for attaching behavior such as profiling and permission and security checks to your service components.

`AbstractEnvelopeInterceptor` is an envelope filter that will execute before and after the component is invoked. Good for logging and profiling.

### Interceptor Event Flow

The following shows an example interceptor stack and the event flow.



## Writing Interceptors

If you want to intercept a message flow to a component on the inbound message flow, you should implement the [Interceptor](#) interface. It has a single method:

```
MuleMessage intercept(Invocation invocation) throws MuleException;
```

The `invocation` parameter contains the current message and the `Service` object of the target component. Developers can extract the current `MuleMessage` from the message and manipulate it as needed. The `intercept` method must return a `MuleMessage` that will be passed on to the component (or to the next interceptor in the chain).

The `EnvelopeInterceptor` works in the same way, except that it exposes two methods that get invoked before and after the event processing:

```
MuleMessage before(Invocation invocation) throws MuleException;
MuleMessage after(Invocation invocation) throws MuleException;
```

## Configuring Interceptors

Interceptors can be configured on your components as follows:

```
<service name="MyService">
  <component>
    <custom-interceptor class="org.my.CustomInterceptor"/>
    <logging-interceptor/>
    <interceptor-stack ref="testInterceptorStack"/>
    <timer-interceptor/>
    <prototype-object class="org.my.ComponentImpl"/>
  </component>
</service>
```



When you configure interceptors, you must specify the object factory explicitly (in this example, <prototype-object>) instead of using the <component class> shortcut.

You can also define *interceptor stacks*, which are one or more interceptors that can be referenced using a logical name. To use an interceptor stack, you must first configure it in the global section of the Mule XML configuration file (above the <model> element):

```
<interceptor-stack name="default">
    <custom-interceptor class="org.my.CustomInterceptor"/>
    <logging-interceptor/>
</interceptor-stack>
```

You can configure multiple <interceptor> elements on your components, and you can mix using built-in interceptors, custom interceptors, and references to interceptor stacks.

## Interceptor Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### <logging-interceptor ...>

The logging interceptor (ported from 1.x).

#### Attributes

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### <custom-interceptor ...>

A user-implemented interceptor.

#### Attributes

Name	Type	Required	Default	Description
class	class name	no		An implementation of the Interceptor interface.

#### Child Elements

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### <interceptor-stack ...>

#### Attributes

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

name	name	no		The name used to identify this interceptor stack.
------	------	----	--	---

#### Child Elements

Name	Cardinality	Description
abstract-interceptor	1..1	A placeholder for an interceptor element.

#### <interceptor-stack ...>

A reference to a stack of interceptors defined globally.

#### Attributes

Name	Type	Required	Default	Description
ref	string	no		The name of the interceptor stack to use.

#### Child Elements

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### <timer-interceptor ...>

The timer interceptor (ported from 1.x).

#### Attributes

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements

Name	Cardinality	Description
------	-------------	-------------

Your Rating:  Results:  0 rates

## Connecting Using Transports

### Connecting Using Transports

#### Available Transports

A transport is responsible for carrying messages from application to application in the Mule ESB framework. In EIP terms, transports are used to implement message channels and provide connectivity to an underlying data source or message channel in a consistent way. For example, the [HTTP transport](#) handles messages sent via the HTTP protocol, whereas the [File transport](#) reads and writes files placed on the server's file system.

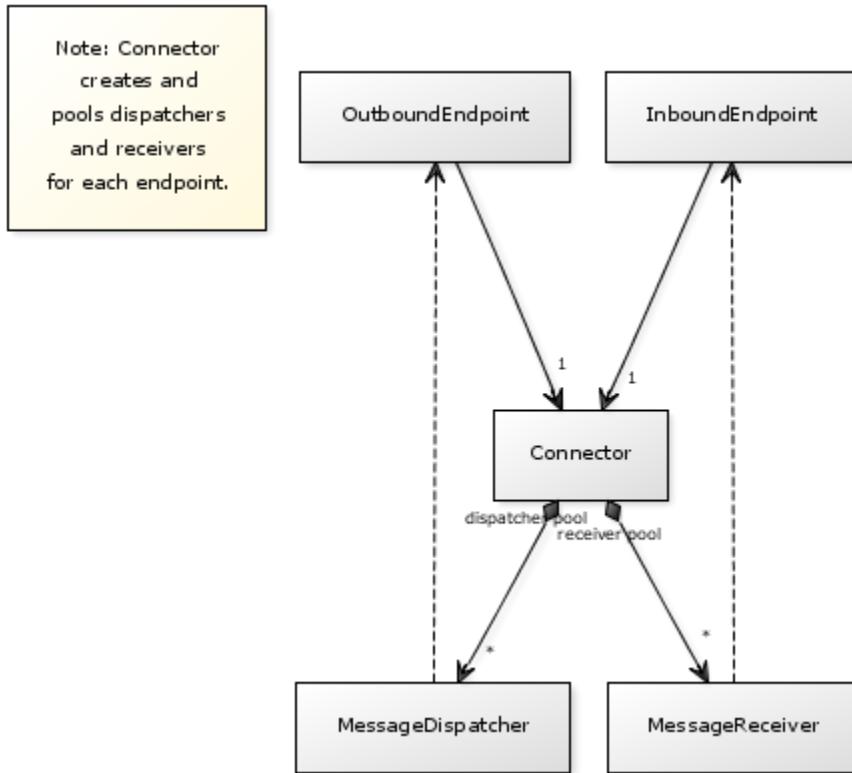
A transport also provides a set of functionality that handles the messages on a specific channel. This includes specific transformers like the `<body-to-parameter-map-transformer>` in the HTTP transport and other specific elements like the various filename parsers for the File transport.

The heart of a transport is the *connector*. The connector maintains the configuration and state for the transport. When receiving messages, a connector uses a *message receiver*, which reads the data, packages it as a message, and passes it to the first message processor in a flow or

the service component's inbound router. When sending messages, the connector uses a *message dispatcher*, which receives the messages and routing instructions from the previous Message Processor in a flow or a service component's outbound router and sends the message to the next endpoint or external service.

Transports are configured via endpoints in Mule. See [Configuring Endpoints](#) for an explanation on how this is done.

A list of available transports and the reference topic including example configuration for each specific transport bundled with Mule can be found in the [Transport Configuration Reference](#).



Your Rating:

Results: 1 rates

## Configuring a Transport

### Configuring a Transport

You can configure a [transport](#) in the following ways:

- Define a connector configuration using the `<connector>` element in the Mule XML configuration file.
- Set transport properties on endpoints to customize the transport behavior for a single endpoint instance.
- Use an endpoint URI that defines the scheme and connection information for the transport, such as `tcp://localhost:12345`. See [Mule Endpoint URLs](#) for more information. The URI consists of the protocol followed by transport-specific information, and then zero or more parameters to set as properties on the connector.

This page describes the common properties for all transports. The actual configuration parameters for each transport type are described separately for each transport. To see the details of a specific transport, see [Transports Reference](#).

### Common Connector Properties

All connectors require that you set the `name` attribute to a unique name for that connector. Additionally, they all include the following common properties.

Property	Description	Default	Required
<code>default-connector-exception-strategy</code>	The exception strategy to use when errors occur in the connector.		No

receiver-threading-profile	The threading properties and <a href="#">WorkManager</a> to use when receiving events from the connector.	The default receiver threading profile set on the <a href="#">Mule Configuration</a>	Yes
dispatcher-threading-profile	The threading properties and <a href="#">WorkManager</a> to use when dispatching events from the connector.	The default dispatcher threading profile set on the <a href="#">Mule Configuration</a> .	Yes
connection-strategy	Desupported as of Mule 2.0. Use <a href="#">retry policies</a> instead.		No
service-overrides	A map of <a href="#">service configuration values</a> that can be used to override the default configuration for this transport.		No
createMultipleTransactedReceivers	Whether to create multiple concurrent receivers for this connector. This property is used by transports that support transactions, specifically receivers that extend the <code>TransactedPollingMessageReceiver</code> , and provides better throughput.	false	No
numberOfConcurrentTransactedReceivers	If <code>createMultipleTransactedReceivers</code> is set to true, the number of concurrent receivers that will be launched.		No
dynamicNotification	Whether to enable dynamic notification.	false	No
validateConnections	Causes Mule to validate connections before use. Note that this is only a configuration hint; transport implementations may or may not make an extra effort to validate the connection.	true	No

You can also set a Spring property on a connector. This is useful if you are using a custom connector.

## Retry Policies

Retry policies are used to configure how a connector behaves when its connection fails. For complete information, see [Configuring Reconnection Strategies](#).

## Creating Your Own Transport

For information on creating a custom transport for Mule ESB, see [Creating Transports](#).

## Detailed Configuration Information

- Threading profiles
  - [Receiver threading profile](#)
  - [Dispatcher threading profile](#)
- Service overrides

cache: Unexpected program error: java.lang.NullPointerException

## Receiver Threading Profile

The threading profile to use when a connector receives messages.

### Attributes of `<receiver-threading-profile...>`

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.

poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

cache: Unexpected program error: java.lang.NullPointerException

### Dispatcher Threading Profile

The threading profile to use when a connector dispatches messages.

#### Attributes of <*dispatcher-threading-profile*>

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).

maxBufferSize	integer		no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.
---------------	---------	--	----	--	--

cache: Unexpected program error: java.lang.NullPointerException

## Service Overrides

Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.

### Attributes of <service-overrides...>

Name	Type	Required	Default	Description
messageReceiver	string	no		
transactedMessageReceiver	string	no		
xaTransactedMessageReceiver	string	no		
dispatcherFactory	string	no		
inboundTransformer	string	no		
outboundTransformer	string	no		
responseTransformer	string	no		
endpointBuilder	string	no		
messageFactory	string	no		
serviceFinder	string	no		
sessionHandler	string	no		
inboundExchangePatterns	string	no		
outboundExchangePatterns	string	no		
defaultExchangePattern	string	no		

Your Rating: 

Results:  0 rates

## Configuring Endpoints

### Configuring Endpoints

[ [Basic Configuration](#) ] [ [Dynamic Endpoints](#) ] [ [Message Processors](#) ] [ [Endpoint Usage](#) ] [ [Global Endpoints](#) ]

Endpoints are used to connect services. An endpoint is a specific channel on which a service can send messages and from which another service can receive messages. For example, a purchasing component may receive an order request over HTTP. Once the order has been processed by the component, a JMS message may be sent over a topic to notify an auditing system, and a response can be sent back over HTTP.

This page describes how to configure an endpoint. For details on the various attributes and elements you can configure on an endpoint, see [Endpoint Configuration Reference](#).

### Basic Configuration

In its most basic form, an endpoint consists of a [transport](#) and a transport-specific channel/destination/resource used to identify the channel and location where two services can exchange information. For example:

### URI-style Endpoints

```
<inbound-endpoint address="udp://localhost:65432"/>

<jetty:inbound-endpoint address="http://localhost:60211/mycomponent1" exchange-pattern="request-response" />

<outbound-endpoint address="smtp://user:secret@smtp.host"/>

<inbound-endpoint address="jms://test.queue"/>
```

Traditionally, endpoints in Mule ESB have been specified as a [URI](#) such as the examples above. This form is still supported, and indeed may prove to be more practical depending on your application. However, as of Mule 2.0, the recommended way to specify endpoints is via transport-specific namespaces, as shown in the following examples.

### Transport-specific Endpoints

```
<file:inbound-endpoint path="./.mule/in"
comparator="org.mule.transport.file.comparator.OlderFirstComparator" reverseOrder="true"/>

<ssl:endpoint name="clientEndpoint" host="localhost" port="60198"/>

<jetty:endpoint name="serverEndpoint" host="localhost" port="60203" path="services/Foo" />

<imaps:endpoint name="global1s" host="localhost" password="secret" port="123" user="bob"/>

<rmi:endpoint name="BadType" host="localhost" port="1099" object="MatchingUMO" method="reverseString"
/>

<quartz:endpoint name="qEP6" repeatCount="10" repeatInterval="1000" jobName="job"/>

<jms:inbound-endpoint queue="test.queue"/>
```

## Dynamic Endpoints

Starting in Mule 3, an outbound endpoint can also be dynamic. This means that the endpoint's URI is the value of an [expression](#), which is evaluated just before a message is sent to it. This allows the target of a message to be determined by its contents or the value of any message property. Dynamic endpoints can use either of the endpoint formats shown above.

### Dynamic Endpoints

```
<outbound-endpoint address="smtp://user:secret#[header:OUTBOUND:host]"/>

<http:outbound-endpoint host="localhost" port="#[header:INBOUND:portNumber]" path="orderService"/>

<jms:outbound-endpoint host="localhost" queue="#[mule:registry.defaultJmsQueue]"/>
```

The only part of an endpoint URI that cannot be dynamic is the scheme. Don't do this:

### Illegal Dynamic Endpoint

```
<outbound-endpoint address="#[header:INBOUND:endpointType]//localhost:8080/service"/>
```

## Connector

In many cases, the connector associated with an endpoint can simply be assumed based on the transport and created implicitly. However, if more than one connector of the same transport exists, or if non-default settings are used for the connector, you must refer to the connector from the endpoint using the `connector-ref` attribute.

#### Specifying a connector

```
<inbound-endpoint address="tcp://localhost:65432" connector-ref="tcpConnector1"/>
<tcp:inbound-endpoint host="localhost" port="65433" connector-ref="tcpConnector2"/>
```

## Properties

Properties on endpoints can be used to customize behavior. Any properties set on the endpoint can be used to override default properties on the associated transport's connector. For example, an SMTP outbound endpoint might set the `fromAddress` property to `workflow1` to override a default connector value of `sysadmin`. Any standard properties for an endpoint will be available as attributes in the XML schema if transport-specific endpoints are used. It is also possible to specify a non-standard property. For example:

#### Setting properties

```
<!-- Standard properties -->
<quartz:endpoint name="qEP6" repeatCount="10" repeatInterval="1000" jobName="job"/>

<!-- Non-standard properties -->
<quartz:endpoint name="qEP7" jobName="job2">
    <property key="actionOnTimeout" value="self-destruct"/>
    <property key="precision" value="2.5"/>
</quartz:endpoint>
```

## Exchange Pattern

By default, endpoints are one-way; that is, they accept (or send) messages, but do not return (or receive) responses to those messages. To set an endpoint to wait for a response, you set `exchange-pattern="request-response"`. This setting is not required by HTTP/S, SSL, TCP, and Servlet endpoints, which are request-response by default.

For more information on configuring messaging styles on an endpoint, see [Service Messaging Styles](#).

## Transaction

A transaction can begin or commit when an event is received or sent via an endpoint. The endpoint must be synchronous, and transaction support depends largely on the particular transport being used. For more information see [Transaction Management](#).

#### Transaction example

```
<jms:inbound-endpoint queue="in">
    <jms:transaction action="BEGIN_OR_JOIN"/>
</jms:inbound-endpoint>
```

## Encoding

This is the encoding an endpoint used to convert message content. For inbound endpoints, it is used to convert the bytes received to characters. For outbound endpoints, it is used to convert characters about to be sent to bytes. If no encoding is set on the endpoint, the default encoding for the Mule configuration is used. This is turn defaults to UTF-8.

#### Encoding example

```
<inbound-endpoint address="tcp://localhost:65432" encoding="iso-8859-1"/>
```

## MimeType

This is the mime type associated with an endpoint's messages. When set on an inbound endpoint, it indicates the type of message expected for incoming messages. Receiving a message with a different mime type will result in an exception. When set on an outbound endpoint, the result is

to set that mime type on all outgoing messages.

#### MimeType example

```
<inbound-endpoint address="tcp://localhost:65432" mimeType="text/xml"/>
```

### [Mule 3.2] Redelivery Policy

A redelivery policy can be defined on an inbound endpoint. It is similar to the maximum redelivery counts that can be set on JMS brokers, and solves a similar problem: if an exception causes the read of a message to be rolled back over and over, how to avoid an infinite loop? Here's an example:

#### MimeType example

```
<flow name =:syncFlow" processing-strategy="synchronous"?>
    <file:inbound-endpoint path="/tmp/file2ftp/ftp-home/dirk">
        <idempotent-redelivery-policy maxRedeliveryCount="3">
            <dead-letter-queue>
                <vm:endpoint path="error-queue" />
            </dead-letter-queue>
        </idempotent-redelivery-policy>
    </file:inbound-endpoint>
    ...

```

If something later in the flow throws an exception, the file won't be consumed, and will be reprocessed. The idempotent-redelivery-policy ensures that it won't be reprocessed more than 3 times; after that, it will be sent to `vm:error-queue`, where it can be handled as an error case.

## Message Processors

What is a message processor? It's a very simple interface for anything which takes a Mule message and does something with it (transforms it, filters it, splits it, etc.). One of the big advantages to everything implementing this simple interface is that message processors can be chained together in any order, there can be any number of them, and they can easily be swapped around. This sort of thing was not at all possible prior to Mule 3.

In the case of endpoints, the following message processors are allowed:

- Transformers
- Filters
- Security Filters
- Aggregators
- Splitters
- Custom Message Processors

You can put any number of these message processors as child elements on an endpoint (inbound or outbound), and they will get applied in the order in which they are listed to any message passing through that endpoint.

In the case of a synchronous outbound endpoint, there is a response message involved, and so any number of message processors can also be put inside a response wrapper and will get applied to the response message in the order in which they are listed.

Note that any of these elements could be declared locally (i.e., in-line in the endpoint) or globally (and referenced via a `ref="foo"` attribute).

### Transformers

Transformers can be configured on an endpoint encapsulating transformation logic in an endpoint that can then be reused as required.

Transformers are configured on endpoints using child message processors elements. When configured on an inbound endpoint they are used to transform the message received by the endpoint, and when configured on an outbound endpoint they are used to transform the message before it is sent.

Response transformers can be configured inside the nested `<response>` element. When configured on an inbound endpoint these transformer will be applied to the message just before it is sent back over the transport, and when configured on an outbound endpoint they are applied on the message received from the invocation of the outbound endpoint if there is one.

As with all message processors configured on endpoints, the order in which they are configured is the order in which they are executed.

```

<inbound-endpoint address="file:///test-data/in">
    <xml-to-object-transformer/>
    <expression-filter expression=" "/>
    <transformer ref="ExceptionBeanToErrorMessage" />
    <response>
        <custom-transformer class="" />
    </response>
</inbound-endpoint>

```

In the above example you can see two request transformers configured, one of which will be executed before the expression filter and the other one after. The custom transformer configured in the `<response>` element will be applied to the response message.

## Global Endpoints

Although globally defined transformers can be referenced from endpoints using the `<{transformer ref="" />` element as seen in the above example endpoints also support a shortcut notification.

The `transformer-refs` and `responseTransformer-refs` attributes can be used to quickly and easily reference global endpoints.

```

<inbound-endpoint address="file:///test-data/in" transformer-refs="globalTransformer1
globalTransformer2" responseTransformer-refs="globalTransformer2"/>

```

Any transformers referenced in this way will be added to the end of the list of message processors configured a child elements and will therefore be executed last. If you need them to be executed before something else like a filter or need to use global endpoints in conjunction with locally defined endpoints in a specific order then you'll need to use `<transformer>` elements instead.

## Filter

An endpoint can contain a filter to selectively ignore certain messages. The filter can be transport-specific such as a JMS selector or file filter or can be a general-purpose filter such as JXPath. Filtering is not supported by all transports, and setting a filter on an endpoint using some transports will result in an `UnsupportedOperationException`. For more information, see [Using Filters](#).

### Filter example

```

<jms:endpoint queue="in.queue">
    <jms:selector expression="JMSPriority > 5" />
</jms:endpoint>

<vm:endpoint name="fruitBowlEndpoint" path="fruitBowlPublishQ">
    <message-property-filter pattern="foo=bar" />
</vm:endpoint>

```

## Other Message Processors

Although filters and transformer are the message processor most used within endpoints, you can just as easily configure other message processors. See more information about the available messages processor on in the [Message Sources and Message Processors](#) page.

## Endpoint Usage

Endpoints can be used in the following places:

- Inbound Routers
- Outbound Routers
- Services
- Catch-all Strategies
- Exception Strategies

## Inbound Routers

See [Inbound Routers](#).

### Inbound router

```
<service name="Receiver">
    <inbound>
        <vm:inbound-endpoint path="inbound.channel" />
        <wire-tap-router>
            <vm:outbound-endpoint path="tapped.channel" />
        </wire-tap-router>
    </inbound>
    <component class="com.acme.SomeService" />
</service>
```

## Outbound Routers

See [Outbound Routers](#).

### Outbound routers

```
<service name="MessageChunker">
    <inbound>
        <jms:inbound-endpoint queue="big.messages" />
    </inbound>
    <outbound>
        <message-chunking-router messageSize="10">
            <jms:outbound-endpoint queue="small.chunks" />
        </message-chunking-router>
    </outbound>
</service>

<service name="LenderGatewayService">
    <inbound>
        <inbound-endpoint ref="LenderGateway" />
    </inbound>
    <outbound>
        <chaining-router>
            <outbound-endpoint ref="LenderService" />
            <outbound-endpoint ref="BankingGateway" transformer-refs="SetLendersAsRecipients
ObjectToJMSMessage" />
        </chaining-router>
    </outbound>
</service>
```

## Services

As a shortcut, endpoints can be configured directly on the service without a router in some cases.

### Implicit router

```
<service name="Echo">
    <inbound>
        <!-- Inbound router is implicit -->
        <stdio:inbound-endpoint system="IN" />
    </inbound>
    <echo-component/>
    <outbound>
        <!-- Outbound router is explicit -->
        <pass-through-router>
            <stdio:outbound-endpoint system="OUT" />
        </pass-through-router>
    </outbound>
</service>
```

## Catch-all Strategies

A single "catch-all" endpoint can be configured for certain types of routers. See [Catch-all Strategies](#).

### Catch-all strategy

```
<service name="dataService">
  <inbound>
    <inbound-endpoint ref="dataIn">
      <payload-type-filter expectedType="java.lang.String"/>
    </inbound-endpoint>
    <forwarding-catch-all-strategy>
      <jms:outbound-endpoint queue="error.queue"/>
    </forwarding-catch-all-strategy>
  </inbound>
  ...cut...
</service>
```

## Exception Strategies

A single error endpoint can be configured on an exception strategy. See [Error Handling](#).

### Exception strategy

```
<service name="dataService">
  <inbound>
    ...cut...
  </inbound>
  <component class="com.acme.DataProcessor"/>
  <outbound>
    ...cut...
  </outbound>
  <default-service-exception-strategy>
    <jms:outbound-endpoint queue="error.queue"/>
  </default-service-exception-strategy>
</service>
```

## Global Endpoints

Global endpoints, while not required, are a recommended best practice for having a nicely organized configuration file. A global endpoint can be thought of as a template for shared endpoint configuration. Global endpoints can be used as they are defined globally, or they can be extended by adding more configuration attributes or elements.

To reference a global endpoint, use the usual `<inbound-endpoint>` and `<outbound-endpoint>` elements, and specify the global endpoint name using the `ref` attribute.

### Global endpoint example

```
<file:endpoint name="fileReader" reverseOrder="true" comparator=
"org.mule.transport.file.comparator.OlderFirstComparator"/>
...cut...

<model>
  <service name="Priority1">
    <file:inbound-endpoint ref="fileReader" path="/var/priol"/>
    ...cut...
  </service>

  <service name="Priority2">
    <file:inbound-endpoint ref="fileReader" path="/var/prio2"/>
    ...cut...
  </service>
</model>
```

In the above example, the "fileReader" endpoint is used as a template for the inbound endpoints. The properties `reverseOrder` and `comparator` only need to be declared once, and the property `path` changes for each inbound endpoint.

Your Rating:  Results:  2 rates

## Mule Endpoint URIs

### Mule Endpoint URIs

Mule Endpoint URIs are any valid [URI](#) and describe how to connect to the underlying transport. Most connectors in Mule can be created from an endpoint URI except where not enough connection information can be expressed clearly in a URI, such as JMS connection properties. Endpoint URIs are set on [Mule Endpoints](#), which manage other connection instance information such as filters and transactions.

Mule Endpoint URIs usually appear in one of the following forms, although other provider implementations can introduce their own schemes.

#### `scheme://host[:port]//[address][?params]`

The scheme must always be set. The host and port are set for endpoints that use unwrapped socket based communications such as the TCP, UDP, HTTP, or multicast.

```
udp://localhost:65432
```

#### `scheme://[username][:password]@host[:port][?params]`

The user name and password are used to log in to the remote server specified by the host and port parameters. The POP3 and SMTP connectors use this format or URI.

```
pop3://ross:secret@pop3.mycompany.com
```

```
smtp://ross:secret@smtp.mycompany.com
```

#### `scheme://address[?params]`

Here we only define a protocol and an address. This tells Mule to get a connector that handles the specified scheme, or create one if needed, and to create a new endpoint using the specified address.

```
vm://my.queue
```

### URI Parameters

There are two types of parameters you can set on the URI:

1. Known Mule parameters that control the way the endpoint is configured, such as transformers for the endpoint.
2. Properties to be set on the connector or to be associated with the transport. This allows you to set properties on a connector used by this endpoint. Additionally, all properties will be associated with the transport, so you can mix connector and transport properties. For more information, see [Configuring Endpoints](#).

#### Known Parameters

Property	Description
connector	The name of an existing connector to use for this endpoint URI
transformers	Defines a comma-separated list of transformers to configure on the endpoint
address	Explicitly sets the endpoint address to the specified value and ignores all other info in the URI.

For example:

```
file:///C:/temp?transformers=FileToString,XmlToDom
jms://jmsEndpoint/topic:my.topic?connector=WMQConnector
```

## Other Parameters

Any other parameters set on the URI will be set on the connector if a connector is created and also set on the endpoint itself as properties.

## Endpoint Encoding

When using XML configuration, certain character entities defined in the [W3C SGML specification](#) need to be escaped to their SGML code. The most relevant are listed here. Don't forget to remove the space before the ';'.

For characters such as > < % #, the notation will be resolved and cause the constructor for the URI to throw an exception. To use one of these characters, you can specify %HEXNUMBER

Text code	Numerical code	What it looks like	Description, extras
%22	#34	"	quotation mark = APL quote, U+0022 ISONEW
&# ;	#38	&	ampersand, U+0026 ISOnum
%3C	#60	<	less-than sign, U+003C ISOnum
%3E	#62	>	greater-than sign, U+003E ISOnum
%25	#37	%	percentage sign, U+0023 ISOnum
%23	#35	#	hash sign, U+0025 ISOnum

Additionally, for connectors such as Axis, FTP, and the Email connectors, if your login credentials include @, you must escape it using %40. For example, instead of these URIs:

```
axis:http://wsuser@username:password@localhost/services/services/Version?method=getVersion
ftp://username:password@ftpserver
smtp://'sender@mydomain.com':'123456'@mailserver?address=QA
```

You must use these:

```
axis:http://wsuser%40username:password%40localhost/services/services/Version?method=getVersion
ftp://username:password%40ftpserver
smtp://'sender%40mydomain.com':'123456'%40mailserver?address=QA
```

Your Rating: 

Results:  1 rates

## Using Filters

### Using Filters

[ [Using in Flows](#) ] [ [Standard Filters](#) ] [ [Transport and Module Filters](#) ] [ [Creating Custom Filters](#) ]

Filters specify conditions that must be met for a message to be routed to a service or continue progressing through a flow. There are several standard filters that come with Mule ESB that you can use, or you can create your own filters.

You can create a global filter and then reference it from your services and flows. Global filters require the "name" attribute, whereas filters configured on endpoints or routers do not.

```

<!-- Globally defined filter with name attribute -->
<payload-type-filter name="payloadFilter" expectedType="java.lang.String" />

<model>
  <service>
    <inbound>
      <tcp:inbound-endpoint host="localhost" port="1234">
        <!-- Here we reference the filter defined globally using it for this endpoint -->
        <filter ref="payloadFilter"/>
      </tcp:inbound-endpoint>
    </inbound>
    <echo-component/>
  </service>
</model>

<flow>
  <tcp:inbound-endpoint host="localhost" port="1234">
    <!-- Here we reference the filter defined globally using it for this endpoint -->
    <filter ref="payloadFilter"/>
  </tcp:inbound-endpoint>
  <echo-component/>
</flow>

```

For reference to the configuration of each filter, see [Filters Configuration Reference](#).

## Using in Flows

Filters return null if the condition is evaluated to false. Otherwise, they return the message. In the case of flows, a condition of false will halt the processing of the flow for that message. If the inbound endpoint defined on a Flow has a request-response exchange-pattern and there are no <response> blocks in your flow then the response used is simply the result from the last Message Processor in the flow, which will be null.

## Standard Filters

Mule includes the following standard filters that you can apply to your routers:

- Payload Type Filter
- Expression Filter
  - Using XPath Expressions
  - Using JXPath Expressions
  - Using OGNL Expressions
- RegEx Filter
- Wildcard Filter
- Exception Type Filter
- Message Property Filter
- Logic Filters
  - And Filter
  - Or Filter
  - Not Filter

### Payload Type Filter

Checks the class type of the payload object inside a message.

```
<payload-type-filter expectedType="java.lang.String">
```

### Expression Filter

Evaluates a range of expressions. Use the `evaluator` attribute to specify the `expression evaluator` to use, one of the following: header, payload-type, exception-type, wildcard, regex, ognl, xpath, jxpath, bean, groovy, or custom. Use the `expression` attribute to set the actual expression. If the expression type is `xpath`, `bean`, or `ognl`, the expression should be a boolean. If the expression type is `custom`, set the `customEvaluator` attribute to the name of the custom evaluator, which must be registered with Mule (see [Creating Expression Evaluators](#)).

Optionally, set the `nullReturnsTrue` attribute to `true` if you want to return `true` whenever the expression is null.

## Using XPath Expressions

XPath expressions are supported using the standard [XPath](#) query language. It is based on JAXP, the Java API for XML processing. You can learn more about writing XPath queries from the [XPath tutorial](#).

```
<expression-filter evaluator="xpath" expression="(msg/header/resultcode)='success'">
```

You can also use the XPath Filter from the [XML Module Reference](#), which supports some additional properties. Mule also provides a Jaxen XPath Filter based on the [Jaxen](#) library that may provide faster performance in some scenarios.

## Using JXPath Expressions

JXPath is an XPath interpreter that can apply XPath expressions to graphs of objects of all kinds: JavaBeans, Maps, Servlet contexts, DOM etc, including mixtures thereof. For more information about JXPath, see the [JXPath user guide](#), [JXPath tutorial](#). For querying XML, it is recommended to use XPath expressions instead.

```
<expression-filter evaluator="jxpath" expression="(msg/header/resultcode)='success'">
```

You can also use the JXPath filter from the [XML Module Reference](#), which supports some additional properties.

## Using OGNL Expressions

OGNL is a simple yet very powerful expression language for plain Java objects. Similar to JXPath, it works on object graphs, and thus the corresponding filter enables simple and efficient content routing for payloads. For example:

```
<expression-filter evaluator="ognl" expression="[MULE:0].equals(42)"/>
```

or more simply:

```
<ognl-filter expression="[MULE:0].equals(42)"/>
```

This filter would block any messages whose payloads are not arrays or lists and do not contain the value 42 as the first element.

## RegEx Filter

Applies a regular expression pattern to the message payload. The filter applies `toString()` to the payload, so you might also want to apply a [PayloadTypeFilter](#) to the message using an [AndFilter](#) to make sure the payload is a String.

```
<regex-filter pattern="the quick brown (.*)"/>
```

## Wildcard Filter

Applies a wildcard pattern to the message payload. The filter applies `toString()` to the payload, so you might also want to apply a [PayloadTypeFilter](#) to the message using an [AndFilter](#) to make sure the payload is a String.

For the string "the quick brown fox jumped over the lazy dog", the following patterns would match:

- \*x jumped over the lazy dog
- the quick\*
- \*fox\*

```
<wildcard-filter pattern="the quick brown *"/>
```

## Exception Type Filter

A filter that matches an exception type.

```
<exception-type-filter expectedType="java.lang.RuntimeException"/>
```

## Message Property Filter

This filter allows you add logic to your routers based on the value of one or more properties of a message. This filter can be very powerful because the message properties are exposed, allowing you to reference any transport-specific or user-defined property. For example, you can match one or more HTTP headers for an HTTP event, match properties in JMS and email messages, and more.

By default, the comparison is case sensitive. You can set the `caseSensitive` attribute to override this behavior.

```
<message-property-filter pattern="Content-Type=text/xml" caseSensitive="false"/>
```

The expression is always a key value pair. If you want to use more complex expressions, you can use the logic filters. The following example shows two filters :

```
<and-filter>
  <message-property-filter pattern="JMSCorrelationID=1234567890"/>
  <message-property-filter pattern="JMSReplyTo=null"/>
</and-filter>
```

## Logic Filters

There are three logic filters that can be used with other filters: And, Or, and Not. Logic filters can be nested so that more complex logic can be expressed.

### And Filter

An And filter combines two filters and only accepts the message if it matches the criteria of **both** filters.

```
<and-filter>
  <payload-type-filter expectedType="java.lang.String"/>
  <regex-filter pattern="the quick brown (.*)"/>
</and-filter>
```

### Or Filter

The Or filter considers two filters and accepts the message if it matches the criteria of **either one** of the filters.

```
<or-filter>
  <payload-type-filter expectedType="java.lang.String"/>
  <payload-type-filter expectedType="java.lang.StringBuffer"/>
</or-filter>
```

### Not Filter

A Not filter accepts the message if it does **not** match the criteria in the filter.

```
<not-filter>
  <payload-type-filter expectedType="java.lang.String"/>
</not-filter>
```

## Transport and Module Filters

Several Mule transports and modules provide their own filters. For example, the [XML Module Reference](#) includes a filter to determine if a message is XML. For more information, see [Transports Reference](#) and [Modules Reference](#). Also, there are filters on [MuleForge](#) that have been contributed by the community.

## Creating Custom Filters

The standard filters handle most filtering requirements, but you can also create your own filter. To create a filter, implement the [Filter interface](#), which has a single method:

```
public boolean accept(MuleMessage message);
```

This method returns true if the message matches the criteria that the filter imposes. Otherwise, it returns false.

You can then use this filter with the `<custom-filter...>` element, using the `class` attribute to specify the custom filter class you created and specifying any necessary properties using the `<spring:property>` child element. For example:

```
<outbound>
  <filtering-router>
    <http:outbound-endpoint address="http://localhost:65071/services/EnterOrder?method=create"
exchange-pattern="request-response"/>
    <custom-filter class="org.mule.transport.http.filters.HttpRequestWildcardFilter">
      <spring:property name="pattern" value="/services/EnterOrder?wsdl"/>
    </custom-filter>
  </filtering-router>
</outbound>
```

Your Rating: 

Results:  3 rates

## Using Transformers

### Using Transformers

[ [Configuring Transformers](#) ] [ [Chaining Transformers](#) ] [ [Transformation Best Practices](#) ] [ [Available Transformers](#) ] [ [Common Attributes](#) ]

Transformers convert message payloads to formats expected by their destinations. Mule ESB provides many standard transformers, which you configure using predefined elements and attributes in your Mule XML configuration file. You can also configure custom transformers using the `<custom-transformer>` element, in which you specify the fully qualified class name of the custom transformer class. For more information on creating and configuring a custom transformer, see [Creating Custom Transformers](#).

Standard transformers are easier to use than custom transformers. You don't need to know the Java name of the transformer, and all properties are explicitly declared in a Mule configuration schema. Following is an example of declaring the standard Append String transformer, which appends string text to the original message payload:

```
<append-string-transformer name="myAppender" message=" ... that's good to know! "/>
```

If the original message payload was the string "foo", the transformer above would convert the string to "foo ... that's good to know!".

The [Available Transformers](#) section of this page describes all the standard transformers provided with Mule. Additionally, many [transports](#) and [modules](#) have their own transformers, such as the `ObjectToJMSMessage` transformer for the JMS transport.

## Configuring Transformers

You can configure a transformer *locally* or *globally*. You configure a local transformer right on the endpoint or in a Flow or where you want to apply it, whereas you configure a global transformer before any `<model>` or `<flow>` elements in your Mule configuration file and then then reference it.

For example, the following code defines two global transformers, which are referenced from two different places:

```
<xm:xml-to-object-transformer name="XMLToExceptionBean" returnClass=
"org.mule.example.errorhandler.ExceptionBean"/>
<custom-transformer name="ExceptionBeanToErrorMessage" class=
"org.mule.example.errorhandler.ExceptionBeanToErrorMessage"
returnClass="org.mule.example.errorhandler.ErrorMessage"/>

...
<flow name="Error Manager">
    <inbound-endpoint address="file://./test-data/in" transformer-refs="XMLToExceptionBean
ExceptionBeanToErrorMessage">
        <file:filename-wildcard-filter pattern="*.xml" />
    </inbound-endpoint>
    ...
</flow>
...
<flow name="Business Error Manager">
    <inbound-endpoint address="jms://exception.queue"/>
    <transformer ref="XMLToExceptionBean"/>
    <transformer ref="ExceptionBeanToErrorMessage"/>
    ...
</flow>
```

## Chaining Transformers

You can chain transformers together so that the output from one transformer becomes the input for the next. To chain transformers, you create a space-separated list of transformers in the `transformer-refs` or `responseTransformer-refs` attributes or by creating multiple `<transformer>` elements as shown above.

For example, this chain ultimately converts from a `ByteArray` to `InputStream`:

```
transformer-refs="ByteArrayToString StringToObject ObjectToInputStream"
```

You could also configure this as follows:

```
<transformer ref="ByteArrayToString"/>
<transformer ref="StringToObject"/>
<transformer ref="ObjectToInputStream"/>
```

Note that if you specify transformer chains, any default transformers or discoverable transformers are not applied. If you want to use those transformers, you must specify them explicitly with the other chained transformers.

## Transformation Best Practices

Mule has an efficient transformation mechanism. Transformers are applied to inbound or outbound endpoints, and the data is transformed just before it is sent from a service or just before it is received by another service. Transformers can be concatenated, so it is simple to perform multiple transformations on data in transit.

There is no one standard approach for how and where transformations should occur. Some maintain that because transformation should always be applied on inbound/outbound data, transformations should be available as part of the enterprise service bus instead of inside the service components. This approach matches the concepts of Aspect Oriented Programming (AOP). Others conclude that it is far more efficient to encode the transformation logic into the service components themselves. In the second case, however, there is no distinction between code that is related to a business process and code that is generic enough to be reused, which contradicts the philosophy of an ESB.

While there is no industry best practice, MuleSoft recommends that developers examine their transformation logic to see if it will always be used (AOP) or if it is specific to a business process. In general, if it will always be used, you should use a transformer, and if it is specific to a single

business process, it should be part of the service component.

Note the following cases where you should **not** configure a transformer:

- Default transformers: some transports have default transformers that are called by default, but only if you don't configure explicit transformations.
- Discoverable transformers: some transformers can be discovered and used automatically based on the type of message. You do not configure these transformers explicitly. These include custom transformers that have been defined as discoverable. For more information, see [Creating Custom Transformers](#).

**References:** <http://msdn2.microsoft.com/en-us/library/aa480061.aspx>  
<http://blogs.ittoolbox.com/eai/archives/transformation-in-a-soa-12186>

## Available Transformers

Following are the transformers available with Mule. Some transformers are specific to a transport or module. For more information, see [Transports Reference](#) and [Modules Reference](#). For a complete reference to the elements and attributes for the standard Mule transformers, see [Transformers Configuration Reference](#).

### Basic

The basic transformers are in the `org.mule.transformer.simple` package. They do not require any special configuration. For details on these transformers, see [Transformers Configuration Reference](#).

Transformer	Description
BeanBuilderTransformer	(As of Mule 2.2) Constructs simple bean objects by defining the object and then setting a number of expressions used to populate the bean properties. For example:  <pre>&lt;bean-builder-transformer name="testTransformer3" beanClass="org.mule.test.fruit.Orange"&gt;     &lt;bean-property property-name="brand" evaluator="mule" expression="message.payload" optional="true"/&gt;     &lt;bean-property property-name="segments" evaluator="mule" expression="message.header(SEGMENTS)"/&gt; &lt;/bean-builder-transformer&gt;</pre>
ByteArrayToHexString <-> HexStringToByteArray	A pair of transformers that convert between byte arrays and hex strings.
ByteArrayToMuleMessage <-> MuleMessageToByteArray	A pair of transformers that convert between byte arrays and Mule messages.
ByteArrayToObject <-> ObjectToByteArray	A pair of transformers that convert between byte arrays and objects. If the byte array is not serialized, ByteArrayToObject returns a String created from the bytes as the returnType on the transformer.
ByteArraySerializable <-> SerializableToByteArray	A pair of transformers that serialize and deserialize objects.
CombineCollectionsTransformer	Takes a payload which is a Collection of Collections and turns into a single List. For example, if the payload is a Collection which contains a Collection with elements A and B and another Collection with elements C and D, this will turn them into a single Collection with elements A, B, C and D.
ExpressionTransformer	Evaluates one or more expressions on the current message and return the results as an Array. For details, see <a href="#">Using Expressions</a> .
MessagePropertiesTransformer	A configurable message transformer that allows users to add, overwrite, and delete properties on the current message.
ObjectArrayToString <-> StringToObjectArray	A pair of transformers that convert between object arrays and strings. Use the configuration elements <code>&lt;byte-array-to-string-transformer&gt;</code> and <code>&lt;string-to-byte-array-transformer&gt;</code> .
ObjectToInputStream	Converts serializable objects to an input stream but treats <code>java.lang.String</code> differently by converting to bytes using the <code>String.getBytes()</code> method.
ObjectToOutputHandler	Converts a byte array into a String.
ObjectToString	Returns human-readable output for various kinds of objects. Useful for debugging.

StringAppendTransformer	Appends a string to an existing string.
StringToObjectArray	Converts a string to an object array. Use the configuration element <string-to-byte-array-transformer>.

## XML

The XML transformers are in the [org.mule.module.xml.transformer](#) package. They provide the ability to transform between different XML formats, use XSLT, and convert to POJOs from XML. For information, see [XML Module Reference](#).

Transformer	Description
XmloToObject <-> ObjectToXml	Converts XML to a Java object and back again using <a href="#">XStream</a> .
JAXB XmloToObject <-> JAXB ObjectToXml	Converts XML to a Java object and back again using the <a href="#">JAXB</a> binding framework (ships with JDK6)
XSLT	Transforms XML payloads using <a href="#">XSLT</a> .
XQuery	Transforms XML payloads using <a href="#">XQuery</a> .
DomToXml <-> XmlToDom	Converts DOM objects to XML and back again.
XmloToXMLStreamReader	Converts XML from a message payload to a StAX XMLStreamReader.
XPath Extractor	Queries and extracts object graphs using XPath expressions using JAXP.
JXPath Extractor	Queries and extracts object graphs using XPath expressions using JXPath.
XmlPrettyPrinter	Allows you to output the XML with controlled formatting, including trimming white space and specifying the indent.

## JSON

The JSON transformers are in the [org.mule.module.json.transformers](#) package. They provide the ability to work with JSON documents and bind them automatically to Java objects. For information, see [Native Support for JSON](#).

## Scripting

The Scripting transformer transforms objects using scripting, such as JavaScript or [Groovy](#) scripts. This transformer is in the [org.mule.module.scripting.transformer](#) package.

## Encryption

The encryption transformers are in the [org.mule.transformer.encryption](#) package.

Transformer	Description
Encryption <-> Decryption	A pair of transformers that use a configured EncryptionStrategy implementation to encrypt and decrypt data.

## Compression

The compression transformers are in the [org.mule.transformer.compression](#) package. They do not require any special configuration.

Transformer	Description
GZipCompressTransformer <-> GZipUncompressTransformer	A pair of transformers that compress and uncompress data.

## Encoding

The encoding transformers are in the [org.mule.transformer.codec](#) package. They do not require any special configuration.

Transformer	Description
Base64Encoder <-> Base64Decoder	A pair of transformers that convert to and from Base 64 encoding.
XMLEntityEncoder <-> XMLEntityDecoder	A pair of transformers that convert to and from XML entity encoding.

## Email

The Email transport provides several transformers for converting from email to string, object to MIME, and more. For details, see [Email Transport Reference](#).

## File

The File transport provides transformers for converting from a file to a byte array (byte[]) or a string. For details, see [File Transport Reference](#).

## HTTP

The HTTP transport provides several transformers for converting an HTTP response to a Mule message or string, and for converting a message to an HTTP request or response. For details, see [HTTP Transport Reference](#). Additionally, the Servlet transport provides transformers that convert from HTTP requests to parameter maps, input streams, and byte arrays. For details, see [Servlet Transport Reference](#).

## JDBC

The Mule Enterprise version of the JDBC transport provides transformers for moving CSV and XML data from files to databases and back. For details, see [JDBC Transport Reference](#).

## JMS

The [JMS Transport Reference](#) and [Mule WMQ Transport Reference](#) (enterprise only) both provide transformers for converting between JMS messages and several different data types.

### Strings and Byte Arrays

The [Multicast Transport Reference](#) and [TCP Transport Reference](#) both provide transformers that convert between byte arrays and strings.

## XMPP

The XMPP transport provides transformers for converting between XMPP packets and strings. For details, see [XMPP Transport Reference](#).

## Common Attributes

Following are the attributes that are common to all transformers.

### returnClass

This specifies the name of the Java class that the transformer returns.

### ignoreBadInput

If set to true, the transformer will ignore any data that it does not know how to transform, but any transformers following it in the current chain will be called. If set to false, the transformer will also ignore any data that it does not know how to transform, but no further transformations will take place.

### MimeType

This mime type will be set on all messages that this transformer produces.

### encoding

This encoding will be set on all messages that this transformer produces.

Your Rating:  Results:  0 rates

## Transformers Configuration Reference

### Transformers Configuration Reference

[ Transformer ] [ Auto transformer ] [ Combine collections transformer ] [ Custom transformer ] [ Message properties transformer ] [ Base64 encoder transformer ] [ Base64 decoder transformer ] [ Xml entity encoder transformer ] [ Xml entity decoder transformer ] [ Gzip compress

transformer ][ Gzip uncompress transformer ][ Byte array to hex string transformer ][ Hex string to byte array transformer ][ Byte array to object transformer ][ Object to byte array transformer ][ Object to string transformer ][ Object to xml transformer ][ Byte array to serializable transformer ][ Serializable to byte array transformer ][ Byte array to string transformer ][ String to byte array transformer ][ Append string transformer ][ Encrypt transformer ][ Decrypt transformer ][ Expression transformer ][ Xpath extractor transformer ]

This page provides details on configuring the standard transformers. Note that many [transports](#) and [modules](#) provide their own transformers as well.

cache: Unexpected program error: java.lang.NullPointerException

## Transformer

A reference to a transformer defined elsewhere.

### Attributes of <transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json
ref	string	yes		The name of the transformer to use.

### Child Elements of <transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Auto transformer

A transformer that uses the transform discovery mechanism to convert the message payload. This transformer works much better when transforming custom object types rather than Java types, because there is less chance for ambiguity.

### Attributes of <auto-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.

mimeType	string	no		The mime type, e.g. text/plain or application/json
----------	--------	----	--	--

#### Child Elements of <auto-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Combine collections transformer

Takes a payload which is a Collection of Collections and turns into a single List. For example, if the payload is a Collection which contains a Collection with elements A and B and another Collection with elements C and D, this will turn them into a single Collection with elements A, B, C and D.

This transformer will also work on MuleMessageCollections. In this case, it will take the individual Collection payloads of each MuleMessage and merge them into a single Collection on a new MuleMessage.

Usage:

```
<combine-collections-transformer/>
```

cache: Unexpected program error: java.lang.NullPointerException

### Custom transformer

A user-implemented transformer.

#### Attributes of <custom-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[']. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
class	class name	yes		An implementation of the Transformer interface.

#### Child Elements of <custom-transformer...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### Message properties transformer

A transformer that can add, delete or rename message properties.

#### Attributes of <message-properties-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json
overwrite	boolean	no	true	If false, a property is not added if the message already contains a property with that name.
scope	invocation/outbound/session/application	no		Property scope to/from which properties are added/removed. The scope determines the lifespan of the properties.

#### Child Elements of <message-properties-transformer...>

Name	Cardinality	Description
delete-message-property	0..*	Delete message properties matching a regular expression or wildcard.
add-message-property	0..*	Add a message property.
rename-message-property	0..*	Rename a message property.
add-message-properties	0..1	Add a set of message properties.

cache: Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

#### Base64 encoder transformer

A transformer that base64 encodes a string or byte array message.

#### Attributes of <base64-encoder-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.

mimeType	string	no		The mime type, e.g. text/plain or application/json
----------	--------	----	--	--

#### Child Elements of <base64-encoder-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Base64 decoder transformer

A transformer that base64 decodes a message to give an array of bytes.

#### Attributes of <base64-decoder-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### Child Elements of <base64-decoder-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Xml entity encoder transformer

A transformer that encodes a string using XML entities.

#### Attributes of <xml-entity-encoder-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

### **Child Elements of <xml-entity-encoder-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Xml entity decoder transformer**

A transformer that decodes a string containing XML entities.

### **Attributes of <xml-entity-decoder-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
MimeType	string	no		The mime type, e.g. text/plain or application/json

### **Child Elements of <xml-entity-decoder-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Gzip compress transformer**

A transformer that compresses a byte array using gzip.

### **Attributes of <gzip-compress-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
MimeType	string	no		The mime type, e.g. text/plain or application/json

### **Child Elements of <gzip-compress-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Gzip uncompress transformer

A transformer that uncompresses a byte array using gzip.

#### Attributes of <gzip-uncompress-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[']. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json

#### Child Elements of <gzip-uncompress-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Byte array to hex string transformer

A transformer that converts a byte array to a string of hexadecimal digits.

#### Attributes of <byte-array-to-hex-string-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[']. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json

#### Child Elements of <byte-array-to-hex-string-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Hex string to byte array transformer

A transformer that converts a string of hexadecimal digits to a byte array.

#### Attributes of <hex-string-to-byte-array-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### Child Elements of <hex-string-to-byte-array-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Byte array to object transformer

A transformer that converts a byte array to an object (either deserializing or converting to a string).

#### Attributes of <byte-array-to-object-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### Child Elements of <byte-array-to-object-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Object to byte array transformer

A transformer that serializes all objects except strings (which are converted using getBytes()).

### Attributes of <object-to-byte-array-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[']. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json

### Child Elements of <object-to-byte-array-transformer...>

Name	Cardinality	Description
cache	Optional	Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

## Object to string transformer

A transformer that gives a human-readable description of various types (useful for debugging).

### Attributes of <object-to-string-transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[']. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json

### Child Elements of <object-to-string-transformer...>

Name	Cardinality	Description
cache	Optional	Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

## Object to xml transformer

Converts a Java object to an XML representation using XStream.

#### **Attributes of <object-to-xml-transformer...>**

Name	Type	Required	Default	Description
driverClass	class name	no		Which XStream driver class to use. Unless you know what you are doing the default will be fine for most cases.
acceptMuleMessage	boolean	no	false	Whether the transformer will serialize the payload or the entire MuleMessage including not only its payload, but also its properties, correlation ID, etc.

#### **Child Elements of <object-to-xml-transformer...>**

Name	Cardinality	Description
alias	0..*	
converter	0..*	The core of XStream consists of a registry of Converters. The responsibility of a Converter is to provide a strategy for converting particular types of objects found in the object graph, to and from XML. XStream is provided with Converters for common types such as primitives, String, File, Collections, arrays, and Dates. For a list of default converters see: <a href="http://xstream.codehaus.org/converters.html">http://xstream.codehaus.org/converters.html</a>

cache: Unexpected program error: java.lang.NullPointerException

#### **Byte array to serializable transformer**

A transformer that converts a byte array to an object (deserializing the object).

#### **Attributes of <byte-array-to-serializable-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json

#### **Child Elements of <byte-array-to-serializable-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Serializable to byte array transformer**

A transformer that converts an object to a byte array (serializing the object).

#### **Attributes of <serializable-to-byte-array-transformer...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### **Child Elements of <serializable-to-byte-array-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Byte array to string transformer**

A transformer that converts a byte array to a string.

#### **Attributes of <byte-array-to-string-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### **Child Elements of <byte-array-to-string-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **String to byte array transformer**

A transformer that converts a string to a byte array.

#### **Attributes of <string-to-byte-array-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.

returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json

#### **Child Elements of <string-to-byte-array-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Append string transformer**

A transformer that appends a string to a string payload.

cache: Unexpected program error: java.lang.NullPointerException

#### **Encrypt transformer**

A transformer that encrypts a message.

#### **Attributes of <encrypt-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
strategy-ref	string	no		The name of the encryption strategy to use. This should be configured using the password-encryption-strategy element, inside a security-manager element at the top level.

#### **Child Elements of <encrypt-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Decrypt transformer**

A transformer that decrypts a message.

#### **Attributes of <decrypt-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
strategy-ref	string	no		The name of the encryption strategy to use. This should be configured using the password-encryption-strategy element, inside a security-manager element at the top level.

#### Child Elements of <decrypt-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Expression transformer

A transformer that evaluates one or more expressions on the current message. Each expression equates to a parameter in the return message. The return message for two or more expressions will be an Object[].

cache: Unexpected program error: java.lang.NullPointerException

#### Xpath extractor transformer

The XPathExtractor is a simple transformer that evaluates an XPath expression using the JAXP libraries against the given bean and returns the result.

By default, a String of the result will be returned. To return a Node, NodeSet, Boolean or Number result, the resultType attribute can be set.

#### Attributes of <xpath-extractor-transformer...>

Name	Type	Required	Default	Description
expression	string	no		The XPath expression.
resultType	xpathResultType	no		The XPath result type (e.g. STRING or NODE).

#### Child Elements of <xpath-extractor-transformer...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as prefix and uri attributes. The prefix can then be used inside the expression.

Your Rating: 

Results:  1 rates

## Native Support for JSON

### Native Support for JSON

[ [JSON Transformers Added](#) ] [ [Examples](#) ] [ [Using the JSON Module](#) ] [ [JSON Bindings](#) ]

JSON is now natively supported in Mule, meaning you can work with JSON documents and bind them automatically to Java objects. Further information is available in the [JSON Module](#) configuration reference.

## JSON Transformers Added

JSON transformers have been added to make it easy to work with JSON encoded messages. We have used the excellent [Jackson Framework](#) which means Mule also supports Json /Object bindings.

## Examples

For example, using AJAX, you will usually receive JSON. From here, you can get a request for a javabean from the server side, and you can convert that automatically to JSON.

Another example, if you get a request from outside, such as a Webservice request, your REST type content could be JSON or XML, while internally the components would be javabeans.

In this case, the feature would automatically respond to a JSON request with a JSON response.

## Using the JSON Module

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

### JSON Bindings

Mule support binding JSON data to objects and marshaling Java object to JSON using the [Jackson Framework](#). Jackson uses annotations to describe how data is mapped to a Java object model. For example, lets say we have an JSON file that describes a person. When we receive that JSON data we want to convert it into a Person object. The JSON looks like this-

```
{
    "name": "John Doe",
    "dob": "01/01/1970",
    "emailAddresses": [
        {
            "type": "home",
            "address": "john.doe@gmail.com"
        },
        {
            "type": "work",
            "address": "jdoe@bigco.com"
        }
    ]
}
```

And we have an object Person we want to create from the JSON data. We use annotations to describe how to perform the mapping. We use the `@JsonAutoDetect` to say that field member names map directly to JSON field names -

```

@JsonAutoDetect
public class Person
{
    private String name;
    private String dob;

    private List<EmailAddress> emailAddresses;

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getDob()
    {
        return dob;
    }

    public void setDob(String dob)
    {
        this.dob = dob;
    }

    public List<EmailAddress> getEmailAddresses()
    {
        return emailAddresses;
    }

    public void setEmailAddresses(List<EmailAddress> emailAddresses)
    {
        this.emailAddresses = emailAddresses;
    }
}

```

The `EmailAddress` object that is used in the `emailAddresses` is just another JavaBean with the `@JsonAutoDetect` annotation.

At this point iBeans can figure out whether to perform a JSON transforms based on the parameters of the method being called. For example -

```

public class PersonService {

    public void processPerson(Person person) {
        //tickle person
    }
}

```

Now if we configure this component in a flow:

```

<flow name="processPerson">
    <jms:inbound-endpoint queue="people.queue"/>
    <component class="org.mule.example.PersonService"/>
</flow>

```

Here we could receive the contents of the `people.json` file above on a JMS queue, Mule would see that `Person.class` is an annotated JSON object and that we had received JSON data from the JMS queue and perform the conversion.

### **Using the Transformers Explicitly**

Often you may want to define a transformer explicitly in Mule, this is done by importing the `json` namespace -

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:json="http://www.mulesoft.org/schema/mule/json"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/json
          http://www.mulesoft.org/schema/mule/json/3.1/mule-json.xsd">

</mule>
```

Then simply configuring the transformer like any other transformer. When converting from JSON to an object form the transformer needs to define the `returnClass` This is the class that the Json payload will get transformed into.

```
<json:json-to-object-transformer name="jsonToFruitCollection" returnClass="org.mule.module.json.transformers.FruitCollection"/>
```

When converting an object to Json, you need to specify the expected source class to convert -

```
<json:object-to-json-transformer name="fruitCollectionToJson"
                                 sourceClass="org.mule.module.json.transformers.FruitCollection">
```

### Annotating objects

Jackson uses annotations to describe how to marshal and unmarshal an object to and from JSON, this is similar in concept to JAXB. However, sometimes it may not be possible to annotate the object class you want to marshal (usually because you do not have access to its source code). Instead you can define mixins. A Mixin is an interface or abstract class (needed when doing constructor injection) that defines abstract methods with Jackson annotations. The method signatures must match the methods on the object being marshalled, at runtime the annotations will be 'mixed' with the object type. To configure Mixins, use the `mixin-map` element or configure them on the transformer directly.

```
<json:mixin-map name="myMixins">
    <json:mixin mixinClass="org.mule.module.json.transformers.FruitCollectionMixin"
                targetClass="org.mule.module.json.transformers.FruitCollection"/>
    <json:mixin
        mixinClass="org.mule.module.json.transformers.AppleMixin"
        targetClass="org.mule.tck.testmodels.fruit.Apple"/>
</json:mixin-map>

<json:json-to-object-transformer name="jsonToFruitCollection" returnClass="org.mule.module.json.transformers.FruitCollection" mixins-ref="myMixins">
```

Or on the transformer directly -

```
<json:object-to-json-transformer name="fruitCollectionToJson"
                                 sourceClass="org.mule.module.json.transformers.FruitCollection">
    <json:serialization-mixin
        mixinClass="org.mule.module.json.transformers.AppleMixin"
        targetClass="org.mule.tck.testmodels.fruit.Apple"/>
</json:object-to-json-transformer>
```

Your Rating: 

Results:  0 rates

## XmIPrettyPrinter Transformer

cache: Unexpected program error: java.lang.NullPointerException

## Xml Prettyprinter Transformer

Formats an XML string using the Pretty Printer functionality in `org.dom4j.io.OutputFormat`.

### Attributes of <xml-prettyprinter-transformer...>

Name	Type	Required	Default	Description
encoding	string	no		The encoding format to use, such as UTF-8.
expandEmptyElements	boolean	no		Whether to expand empty elements from <tagName> to <tagName></tagName>.
indentEnabled	boolean	no		Whether to enable indenting of the XML code. If true, the indent string and size are used.
indentString	string	no		The string to use as the indent, usually an empty space.
indentSize	integer	no		The number of indent strings to use for each indent, such as "2" if indentString is set to an empty space and you want to use two empty spaces for each indent.
lineSeparator	string	no		The string to use for new lines, typically "\n".
newLineAfterNTags	integer	no		If the newlines attribute is true, the number of closing tags after which a newline separator is inserted. For example, setting this to "5" will cause a newline to be inserted after the output of five closing tags (including single tags).
newlines	boolean	no		Whether newlines should be printed. If false, the XML is printed all on one line.
omitEncoding	boolean	no		Whether the XML declaration line includes the encoding of the document. It is common to suppress this in protocols such as SOAP.
padText	boolean	no		Whether to ensure that text immediately preceded by or followed by an element will be "padded" with a single space. This is useful when you set trimText to true and want to ensure that "the quick <b>brown</b> fox" does not become "the quick<b>brown</b>fox".
suppressDeclaration	boolean	no		Whether to suppress the XML declaration line. It is common to suppress this in protocols such as SOAP.
trimText	boolean	no		Whether to trim white space in the XML.
XHTML	boolean	no		Whether to use the XHTML standard, which is like HTML but passes an XML parser with real, closed tags, and outputs CDATA sections with CDATA delimiters.

Your Rating: 

Results:  1 rates

## Creating Custom Transformers

### Creating Custom Transformers

Transformers in Mule are used to convert messages from one format to another or to manipulate the message information such as headers and attachments. Mule ESB also provides several standard transformers, including XML transformers (such as XML to Object, XSLT, and DOM to XML), encryption transformers that encrypt and decrypt data, compression transformers that compress and decompress data, and more. For a list of the standard transformers in Mule, see [Using Transformers](#).

Since Mule 3 there are two ways to create a transformer.

1. Use a [Transformer Annotation](#) on a method. This transformer will be automatically discovered and will be available to Mule's [automatic transformation engine](#). Transformers created this way are not usually referenced in your Mule configuration, instead it will be discovered based on the current type of message payload and the required type need (i.e. the parameter type of a service component method parameter).
2. The traditional Mule way of creating custom transformers by extending [AbstractTransformer](#).

Your Rating: 

Results:  0 rates

## Creating Service Objects and Transformers Using Annotations

## Creating Service Objects and Transformers Using Annotations

[ [Annotations for Service Components](#) ] [ [Annotations for Transformers](#) ] [ [Annotations Reference](#) ]

Annotations now allow service objects to be created using purely annotations on the service component. Additionally, there is a new annotation for creating Mule transformers that makes it much easier to define and discover custom transformers.

### Annotations for Service Components

Annotations will especially help if you're using service components, where annotations provide request injection capabilities. This means you can define what data to read from the incoming methods, similar to dependency injection for runtime.

When you have a request to your component, what kind of things can you inject? You can inject message payload, then transform that into an object. You can also inject message headers, attachments, perform actions on payload (using xpath annotation to describe the node you want to get from the incoming document). This reduces the amount of xml you need to create for your configuration, because you can now put it all in the component. This makes your configuration a lot cleaner.

### Annotations for Transformers

Mule 3.0 also introduces a new annotation for creating Mule transformers that makes it much easier to define and discover custom transformers.

Until now, creating transports for Mule required a good deal of XML, even though the most important thing you wanted to see was probably where the data came from and where it went. The new annotation in Mule 3.0 removes the need for some of the xml config required to define a new transport.



When using Parameter annotations remember that every parameter needs to be annotated for the method invocation to work.

### Annotations Reference

Annotation	Description	Type	Module
@ContainsTransformerMethods	Signals that a class contains transformer methods (Since Mule 3.0.1)	Method	annotations
@Transformer	Used to create a Mule transformer from a class method	Method	annotations
@Schedule	Schedules a method for execution using either a simple frequency or cron expression	Method	quartz
@IntegrationBean	A dependency injector annotation used to inject an iBean such as Facebook or Amazon S3 into your component	Field	ibbeans
@MockIntegrationBean	Similar to the @IntegrationBean annotation, this one can be used in tests to create a mock instance of an ibean we is good for testing the bean without actually making requests to the external service	Field	ibbeans
@Lookup	Dependency injection annotation used to retrieve objects from the registry	Field,Parameter	annotations
@Payload	A parameter injection annotation that can be used on component entry points and transformer methods defined using the @Transformer annotation, this annotation controls how the current message payload is passed into a method by performing automatic transformation of the message payload to match the annotated parameter type	Parameter	annotations
@InboundHeaders	Used on component entry points and transformer methods, this annotation controls how the current message inbound headers are passed into a method. The annotation supports, Map, List, single headers and wildcards	Parameter	annotations
@OutboundHeaders	Used on component entry points and transformer methods, this annotation controls how the current message outbound headers are passed into a method. Users can write to this Map to attach headers to the outgoing message	Parameter	annotations
@InboundAttachments	Used on component entry points and transformer methods, this annotation controls how the current message inbound attachments are passed into a method. The annotation supports, Map, List, single headers and wildcards	Parameter	annotations
@OutboundAttachments	Used on component entry points and transformer methods, this annotation controls how the current message outbound attachments are passed into a method. Users can write to this Map to attach attachments to the outgoing message	Parameter	annotations

@Xpath	This annotation can be used to execute an Xpath expression on the message payload with the result being passed into the method.	Parameter	xml
@Groovy	This annotation can be used to execute an Groovy expression on the message payload with the result being passed into the method.	Parameter	scripting
@Mule	A parameter injection annotation that can be used on component entry points and transformer methods, this annotation can be used to execute a Mule Expression	Parameter	annotations
@Function	A parameter injection annotation expression on the message payload with the result being passed into the method, this annotation exposes a common set of functions used in Mule such as a counter, UUID generator, date and timestamps, etc	Parameter	annotations

Your Rating: 

Results:  0 rates

## Function Annotation

### @Function Annotation

A parameter injection annotation expression on the message payload with the result being passed into the method, this annotation exposes a common set of functions used in Mule such as a counter, UUID generator, date and timestamps.

```
public class MyComponent {
    public Object process(@XPath("/Envelope") Document doc
                           @Function("uuid") String id) {
        // do stuff
    }
}
```

## Functions

Function	Description
@Function("now")	Returns a <code>java.sql.Timestamp</code> with the current time
@Function("date")	Returns a <code>java.util.Date</code> with the current time
@Function("dateStamp")	Returns a <code>java.lang.String</code> that contains the current date formatted according as{{dd-MM-yy_HH-mm-ss.SSS}}
@Function("dateStamp-dd-MM-yyyy")	Returns a <code>java.lang.String</code> that contains the current date formatted according as{{dd-MM-yyyy}}
@Function("uuid")	Returns a globally unique identifier as a <code>java.lang.String</code>
@Function("hostname")	Returns the hostname of the machine Mule is running on as a <code>java.lang.String</code>
@Function("ip")	Returns the IP address of the machine Mule is running on as a <code>java.lang.String</code>
@Function("count")	Returns a local count as a <code>java.lang.Long</code> that will increment for each call. If the server is restarted, the counter will return to zero
@Function("payloadClass")	Returns the fully qualified class name of the payload as a <code>java.lang.String</code>
@Function("shortPayloadClass")	Returns just the class name of the payload as a <code>java.lang.String</code>

Your Rating: 

Results:  0 rates

## Groovy Annotation

### @Groovy Annotation

A parameter injection annotation that can be used on component entry points and transformer methods, this annotation can be used to execute an Groovy expression on the message payload with the result being passed into the method. For example, if you are expecting a Java Bean, this can be injected and an Groovy expression evaluated against it. For more information about scripting support in Mule see the [Scripting Module](#)

## Reference.

Lets say our payload is a Person object that has a field called `emailAddress`.

```
public class MyComponent {  
    public Object process(@Groovy("payload.emailAddress") String email) {  
        // do stuff  
    }  
}
```

## Context Bindings

In the example above 'payload' is a context binding that gives you access to the message payload directly in Groovy. There are other context bindings available to Groovy scripts, See [Script Context Bindings](#) for more information.

## Validation

All expressions in Mule have their syntax validated to ensure the expression is well formed. In rare cases you may want to turn this off for Groovy depending on whether the expression has any braces that are not closed. To do this you need to set a system property -

```
-Dmule.validate.expressions=false
```

Your Rating:  Results:  0 rates

## InboundAttachments Annotation

### **@InboundAttachments Annotation**

This annotation controls how the current message inbound attachments are passed into a method. The annotation supports, Map, List, single attachment, wildcards and optional entries. It can be used on component entry points and [@Transformer](#) methods.

### Accessing Message Attachments

A message in Mule can optionally have attachments. The modules in Mule that support attachments are HTTP, Email, Axis and VM. If you need to access the attachments, add a parameter to the component method signature or transformer method signature annotated with the `@InboundAttachments` annotation -

The `**` indicates that all headers should be returned.

```
@Transformer  
public Person xmlToOrder(@Payload Document data,  
                           @InboundAttachments("**) Map<String, DataHandler> headers)
```

Note that the Map type will be `Map<String, DataHandler>`. The attachment information can be queried from the `javax.activation.DataHandler` instance.

Wildcard expressions can be used to match a subset of attachments too -

```
@Transformer  
public Person xmlToOrder(@Payload Document data,  
                           @InboundAttachments("*.pdf") Map<String, DataHandler> headers)
```

Alternatively, you can specify a single header name and just return the value of that attachment:

```

public class OrderService {
    public OrderConfirmation processOrder(@Payload InputStream data,
        @InboundAttachments("shipping-info.doc") DataHandler attachment) {
        //do stuff
    }
}

```

To receive a subset of specific attachments, you can list them as comma-separated values:

```

public class OrderService {
    public OrderConfirmation processOrder(@Payload InputStream data,
        @InboundAttachments("shipping-info.doc, invoice.pdf") Map<String, DataHandler> attachments) {
        //do stuff
    }
}

```

By default an error will be thrown if a listed attachment is not on the response. To avoid this error, mark the header as optional with '?'. For example, `shipping-info.doc` is optional in the following code:

```

public class OrderService {
    public OrderConfirmation processOrder(@Payload InputStream data,
        @InboundAttachments("shipping-info.doc?, invoice.pdf") Map headers) {
        //do stuff
    }
}

```

If the return type for the `@InboundAttachments` param is a `java.util.List`, just the values will be returned.

```

public class OrderService {
    public OrderConfirmation processOrder(@Payload InputStream data,
        @InboundAttachments("shipping-info.doc?, invoice.pdf") List<DataHandler> attachments) {
        //do stuff
    }
}

```

Your Rating:  Results:  0 rates

## InboundHeaders Annotation

### **@InboundHeaders Annotation**

This annotation controls how the current message inbound headers are passed into a method. The annotation supports, Map, List, single headers, wildcards and optional entries. It can be used on component entry points and `@Transformer` methods.

#### **Accessing Message Information**

All messages in Mule have headers. If you need to access the headers, add a parameter to the component method signature or transformer method signature annotated with the `@InboundHeaders` annotation:

The '\*' indicates that all headers should be returned.

```

@Transformer
public Person xmlToPerson(@Payload Document data, @InboundHeaders("*") Map headers)

```

Wildcard expressions can be used to match a subset of headers too -

```
@Transformer  
public Person xmlToPerson(@Payload Document data, @InboundHeaders("X-*") Map headers)
```

Alternatively, you can specify a single header name and just return the value of that header:

```
@Transformer  
public Person xmlToPerson(@Payload InputStream data, @InboundHeaders("Content-Type") String  
contentType)
```

To receive a subset of headers, you can list them as comma-separated values:

```
@Transformer  
public Person xmlToPerson(@Payload InputStream data, @InboundHeaders("Content-Type, Host, X-MyHeader")  
Map headers)
```

By default an error will be thrown if a listed header is not on the response. To avoid this error, mark the header as optional with '?'. For example, X-MyHeader is optional in the following code:

```
@Transformer  
public Person xmlToPerson(@Payload InputStream data, @InboundHeaders("Content-Type, Host, X-MyHeader?")  
Map headers)
```

If the return type for the @InboundHeaders param is a `java.util.List`, just the values will be returned.

```
@Transformer  
public Person xmlToPerson(@Payload InputStream data, @InboundHeaders("Content-Type, Host, X-MyHeader?")  
List headers)
```

Your Rating:  Results:  0 rates

## Lookup Annotation

[ [@Lookup Annotation](#) ] [ [Field injection](#) ] [ [Parameter injection](#) ] [ [JSR-330 Annotations](#) ] [ [Optional lookups](#) ]

### **@Lookup Annotation**

The Lookup annotation is used to inject objects from the Mule registry. These are the objects created from the Mule configuration files. Typically it will be used to inject objects from the registry into custom components and transformers. It can be used in two ways; field injection and parameter injection.

#### **Field injection**

Field injection can be done by type -

```

public class MyComponent {
    @Lookup
    private MuleContext muleContext;
    ...
}
or by name of the object configured in the Mule configuration -
{code:java}
public class MyComponent2 {
    @Lookup("myTransformer")
    private Transformer transformer;
    ...
}

```

### Parameter injection

This injection happens when a component entry point (method call) or when a Transformer method is called. This type of injection is less useful since most of the time you will want to inject objects during initialize phase.

```

public class MyTransformers {
    @Transformer
    public Item xmlToItem(InputStream payload, @Lookup JAXBContext jaxbContext) {
        // do stuff
    }
}

```

Here, when the transformer is invoked a pre-configured JAXBContext will be injected.  
The same injection works for components too.

```

public class MyComponent3 {
    public Object process(@Payload String payload, @Lookup MuleContext muleContext) {
        // do stuff
    }
}

```

### JSR-330 Annotations

If you are familiar with the `javax.inject` or JSR-330 annotations you might be wondering why we didn't use `@Inject` and `@Named`. These annotations are designed for Dependency Injection frameworks such as Spring or Guice. Mule is not a DI container but works with the existing ones, however, sometimes you will want to inject objects configured in Mule into your components and transformers.

### Optional lookups

Finally, lookups are required by default, if a lookup returns null an exception will be thrown. If you want a null to be returned, just set the optional flag -

```

public class MyComponent3 {
    @Lookup(value = "api.key", optional = true)
    private String apiKey;
    ...
}

```

Your Rating:  Results:  0 rates

[ [@Lookup Annotation](#) ] [ [Field injection](#) ] [ [Parameter injection](#) ] [ [JSR-330 Annotations](#) ] [ [Optional lookups](#) ]

### Mule Annotation

## **@Mule Annotation**

A parameter injection annotation that can be used on component entry points and transformer methods, this annotation can be used to execute a Mule Expression on the message payload with the result being passed into the method.

```
public class MyComponent {  
    public Object process(@Mule("message.payload") String payload) {  
        // do stuff  
    }  
}
```

Mule defines a micro language for accessing the current message and Mule environment, more information can be [found here](#).

Your Rating:  Results:  0 rates

## **OutboundAttachments Annotation**

### **@OutboundAttachments**

This parameter annotation is used to inject a attachments map into your component or transformer that can be used to write attachments to an outgoing messaging without having to deal with the `org.mule.api.message.MuleMessage` directly. This makes your code more portable and easier to test in isolation.

There are no parameters for this annotation and the parameter must be `java.util.Map<String, DataHandler>`. Any outbound attachments set by previous elements in a [Flow](#) or [Service](#) will be accessible and writable in this map.

```
public class MyComponent {  
    public Object process(@Payload Document doc,  
                         @OutboundAttachments Map<String, DataHandler> outAttachments) {  
        // do stuff  
    }  
}
```

Your Rating:  Results:  0 rates

## **OutboundHeaders Annotation**

### **@OutboundHeaders**

This parameter annotation is used to inject a headers map into your component or transformer that can be used to write headers to an outgoing messaging without having to deal with the `org.mule.api.message.MuleMessage` directly. This makes your code more portable and easier to test in isolation.

There are no parameters for this annotation and the parameter must be `java.util.Map<String, Object>`. Any outbound headers set by previous elements in a [Flow](#) or [Service](#) will be accessible and writable in this map.

```
public class MyComponent {  
    public Object process(@Payload Document doc,  
                         @OutboundHeaders Map<String, Object>}. outHeaders) {  
        // do stuff  
    }  
}
```

Your Rating:  Results:  0 rates

## **Payload Annotation**

### **@Payload**

A parameter injection annotation that can be used on component entry points and transformer methods defined using the [@Transformer](#)

annotation, this annotation controls how the current message payload is passed into a method by performing automatic transformation of the message payload to match the annotated parameter type. For example, if you are expecting an XML document, this can be injected into a component entry point and automatically converted into a `org.w3.dom.Document`.

```
public class MyComponent {  
    public Object process(@Payload Document doc) {  
        // do stuff  
    }  
}
```

### Annotated Component Methods

Note that when using Annotations on component methods, all parameters need to be annotated for the method to be invoked. You will almost always be using the `@Payload` annotation to mark the message payload parameter.

Your Rating:  Results:  1 rates

## Schedule Annotation

[ [@Schedule Annotation](#) ] [ [Arguments](#) ] [ [Introduction to Cron expressions](#) ] [ [Some examples](#) ]

### @Schedule Annotation

The `@Schedule` annotation is a method level annotation that is used to schedule how often the method is called. To call a method every second you could use the following -

```
public class PingMe {  
    @Schedule(interval = 1000)  
    public String ping()  
    {  
        return "ping!";  
    }  
}
```

The interval is defined in milliseconds. When the `PingMe` class is loaded in the Mule container a schedule will be set up to call the `ping()` method every second.

The `@Schedule` annotation also supports `cron` expressions. These provide a powerful way to express time triggers.

The `getStatus` method below will be called every hour.

```
@Schedule(cron = "* * 0 * * ?") //every hour  
public String getStatus()  
{  
    //use Twitter iBean  
}
```

### Arguments

Argument	Description	Required
cron	A <a href="#">cron command</a> that specifies when to call the method.	You must set either cron or interval.
interval	The number of milliseconds between two scheduled invocations of the method.	You must set either cron or interval

config	A reference to the <code>org.mule.transport.quartz.config.ScheduleConfigBuilder</code> object that is used to configure this scheduler job. This attribute can be used to reference a local schedule configuration builder using the name of the builder. Local configBuilder references can use the <code>id</code> value passed into the <code>ScheduleConfigBuilder</code>	No
startDelay	The number of milliseconds that will elapse before the first event is fired. The default is -1, which means the first event is fired as soon as the service is started.	No

### Introduction to Cron expressions

Cron is a UNIX tool that has been around for ever and is used for scheduling operating system tasks. It uses "cron expressions", which are able to create firing schedules such as: "At 8:00am every Weekday" or "every 5 minutes". Cron expressions are powerful but can be a little confusing so I have provided some examples below. A cron expression consists of 7 fields, one of which is optional, listed below.

Field Name	Mandatory	Allowed Values	Allowed Special Chars
Seconds	YES	0-59	, - * /
Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of Month	YES	1-31	, - * ? / L W C
Month	YES	1-12 or JAN-DEC	, - * /
Day of Week	YES	1-7 or SUN-SAT	, - * ? / L C #
Year	NO	empty, 1970-2099	, - * /

### Some examples

- 0 0 12 \* \* ? Fire at 12pm (noon) every day
- 0 15 10 ? \* \* Fire at 10:15am every day
- 0 15 10 \* \* ? 2009 Fire at 10:15am every day during the year 2009
- 0 \* 14 \* \* ? Fire every minute starting at 2pm and ending at 2:59pm, every day
- 0/0/5 14 \* \* ? Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
- 0 15 10 ? \* 6L Fire at 10:15am on the last Friday of every month
- 0 11 11 11 11 ? Fire every November 11th at 11:11am

The [Quartz documentation](#) also provides an in depth description of what you can do with cron expressions.

[ [@Schedule Annotation](#) ] [ [Arguments](#) ] [ [Introduction to Cron expressions](#) ] [ [Some examples](#) ]

Your Rating:  Results:  0 rates

## Transformer Annotation

### @Transformer Annotation

[ [Creating Transformers](#) ] [ [Working with XML and JSON](#) ] [ [Accessing Message Information](#) ] [ [Explicitly invoking your @Transformer](#) ] [ [Making Mule aware of your @Transformer](#) ] [ [Transformer Rules](#) ]

Transformers in Mule are used to convert messages from one data type to another. As well as the in-built Mule [transformers](#), it is now possible to create custom transformers using the `@Transformer` annotation.

### Creating Transformers

In their simplest form, transformers convert between different Java types. You create a transformer by performing the conversion in a method that you annotate with the `@Transformer` annotation:

```

@ContainsTransformerMethods // since Mule 3.0.1
public class MyTransformers
{
    @Transformer
    public URL stringToURL(String string) throws MalformedURLException
    {
        return new java.net.URL(string);
    }
}

```

At runtime, this method will be discovered and registered with Mule. Then, whenever an Mule needs to convert from a String to a URL, this transformer will be used.

Here we take an input of `java.lang.String`, this is referred to as the *source type* and convert it to a `java.net.URL`, this is referred to as the *return type*.



If you create a transformer with the same source and return types as one of the standard transformers provided with Mule, your custom transformer will be given priority and used instead. Be careful not to create multiple transformers with the same source and return types, or an exception will be thrown due to multiple transformers matching the exact same source and return types.

Note that the method throws `MalformedURLException`, which is an exception specific to creating URL objects. It's good practice to just re-throw any exceptions thrown in a transformer method and let the container handle them.

#### **Working with Collections**

The Mule transformation system support generics with collections, this means that transforms can be matched against collections of a specific type of object. To extend the example above, we could have a transformer that created a List of URL objects from a comma-separated list of URL strings.

```

@Transformer
public List<URL> stringsToURLs(String string) throws MalformedURLException
{
    List<URL> urls = new ArrayList<URL>();
    StringTokenizer tokenizer = new StringTokenizer(string);
    while (tokenizer.hasMoreTokens())
    {
        urls.add(new URL(tokenizer.nextToken()));
    }
    return urls;
}

```



It is good practice to always provide types for any transforms that deal with collections. Doing so will provide more accurate information to Mule about how to match your transformers and it provides better type checking in your code.

#### **Multiple Source Types**

Transformers can only have one return type but we can define multiple *source types*. For example, lets say we might receive the URL string as a `java.lang.String` or `java.io.InputStream` we could add the additional source type to the `@Transformer` annotation. Note that you can add a comma-separated list of source type classes.

```

@Transformer(sourceTypes = {InputStream.class})
public URL transformStringToURL(String string) throws MalformedURLException
{
    return new URL(string);
}

```

Now if a request is made to convert a `java.io.InputStream` to a `java.net.URL`, this transformer would be discovered and behind the scenes Mule would attempt to convert `java.io.InputStream` to `String` before calling the method. Note there needs to be a transformer

registered that will convert a `java.io.InputStream` to `java.lang.String` and Mule provides a selection of default transformers for dealing with JDK types such as String, byte arrays, InputStreams, Xml Documents, etc. But if you need a transform that doesn't exist you can just create a new transform method.

## Working with XML and JSON

Most services deal with data formatted using XML or JSON (JavaScript Object Notation). Mule supports binding objects to XML and vice-versa using JAXB (Java API for XML binding) which is standard in JDK 6. JSON marshaling to objects and back again using a JSON parsing framework called [Jackson](#). These frameworks are supported automatically by Mule. For more information see [Using Bindings](#).

### Accessing Message Information

All messages in Mule have headers. If you need to access the headers, add a parameter to the transformer method signature annotated with the `@InboundHeaders` or `@OutboundHeaders` annotations. If you need access to the message attachments use the `@InboundAttachments` or `@OutboundAttachments` annotations.

```
@Transformer  
public Person xmlToPerson(InputStream data, @InboundHeaders("*) Map headers,  
                           @OutboundAttachments Map<String, DataHandler> attachments)
```

The '\*' indicates that all headers should be returned. The OutboundAttachments Map allows users to write attachments to the outgoing message.

### Explicitly invoking your `@Transformer`

There is no mechanism in Mule 3.x to actually invoke a transformer which is being constructed from an annotated method.

If the datatypes that you are trying to transform are custom objects instead of Java data types, you can potentially use the `<auto-transform>` element.

```
<auto-transformer returnClass="Person.class"/>
```

The auto-transform element will use the transformer discovery service and search for a transformer that will convert the current message payload type into the specified returnClass. As you can see you are not actually invoking your transformer directly but instead using the Mule built-in discovery mechanism to do so.

### Making Mule aware of your `@Transformer`

For performance reasons Mule does not perform class path scanning. This means that even if you annotate your class with `@ContainsTransformerMethods` it won't be recognized automatically by just adding the class in the class path.

You need to register the class in the Mule registry somehow that will effectively also (since it is annotated `@ContainedTransformerMethods`) register every transformer method inside of it.

There are several ways to do so, one way would be to declare the class a Spring bean inside your Mule config as:

```
<spring:bean id="xxx" class="MyTransformer.class"/>
```

Another way would be to add the class to the `registry-bootstrapping.properties` file. Click [here](#) for more information about along side some example.

### Transformer Rules

- Since Mule 3.0.1, this class must be annotated with `@ContainsTransformerMethods`
- If a transformer has state, all transformers defined in that class will share that state.
- Primitive types must not be used for transformer method return types. Only objects can be used.
- For collections use Lists or Sets, not arrays. Generics are supported and should be used where ever possible since the generic types are also used when trying to match transformers.
- The transformer methods must be public and concrete implementations, the `@Transformer` annotation cannot be used on an interface.
- The transform method must have at least one parameter and a non-void return type.
- `java.lang.Object` cannot be used for the parameter types or return type.

Your Rating: 

Results:  0 rates

## XPath Annotation

### @XPath Annotation

A parameter injection annotation that can be used on component entry points and transformer methods, this annotation can be used to execute an Xpath expression on the message payload with the result being passed into the method. For example, if you are expecting an XML document, this can be injected and an XPath expression evaluated against it. Note that any type conversion will be done for you automatically.

```
public class MyComponent {  
    public Object process(@XPath("/Envelope") Document doc) {  
        // do stuff  
    }  
}
```

You can also use multiple expressions -

```
public class MyComponent {  
    public Object process(@XPath("/Envelope") Document doc  
                         @XPath("/Envelope/@id") String id) {  
        // do stuff  
    }  
}
```

### Namespaces

Namespaces can be configured in the Mule Configuration using the [XML Namespaces](#), these will be made available for this annotation.

First declare the namespace using the [Namespace Manager](#) -

```
<mulexml:namespace-manager includeConfigNamespaces="true">  
    <mulexml:namespace prefix="e" uri="http://foo.com/message/envelope"/>  
</mulexml:namespace-manager>
```

Then you can reference the 'e' namespace in the XPath expression -

```
public class MyComponent {  
    public Object process(@XPath("/e:Envelope") Document doc) {  
        // do stuff  
    }  
}
```

Your Rating:

Results: 0 rates

## Creating Custom Transformer Class

### Creating a Customer Transformer Class

[ [Transformer Classes](#) ] [ [Registering Source and Return Types](#) ] [ [Using Transformer Lifecycle Methods](#) ] [ [Creating Discoverable Transformers](#) ] [ [Registering the Transformer](#) ] [ [Examples](#) ]

A custom transformer is a user-defined transformer class that implements [org.mule.api.transformer.Transformer](#). Your class can extend [AbstractTransformer](#) or [AbstractMessageAwareTransformer](#), depending on your needs. This page describes how to create a custom transformer in more detail.

Go here if you are looking for information about creating a transformer with the new [Transformer Annotation](#).

### Transformer Classes

[AbstractTransformer](#) allows you to access and transform the source payload and to specify the encoding to use (if required). It defines methods

for controlling the object types this transformer supports and validates the expected return type, leaving you to implement a single `doTransform()` method.

```
import org.mule.transformer.AbstractTransformer;

public class OrderToHtmlTransformer extends AbstractTransformer
{
    public Object doTransform(Object src, String encoding) throws TransformerException
}
```

If you need to transform the message header and attachments, you can use `AbstractMessageAwareTransformer` instead to change them directly on the message passed in. In this case, you return the transformed message payload by overriding the method `transform(MuleMessage message, String encoding)`. You can use `message.getProperty(Object key)` to retrieve the properties or `message.setProperty(Object key, Object value)` to set properties on the transformed message.

For example:

```
import org.mule.transformer.AbstractMessageAwareTransformer;

public class OrderToHtmlTransformer extends AbstractMessageAwareTransformer
{
    public Object transform(MuleMessage message, String encoding) throws TransformerException
}
```

## Registering Source and Return Types

You can specify which source types a transformer can accept and which type it will return. This allows Mule to validate the incoming message before invoking the transformer and to validate the output before sending the message on. If you create a `discoverable transformer`, you must set the source and return types.

For example, for the Order bean to HTML transformer, you would specify in the constructor that the transformer converts only message payloads of type Order:

```
public class OrderToHtmlTransformer extends AbstractMessageAwareTransformer
{
    public OrderToHtmlTransformer() {
        registerSourceType(DataTypeFactory.create(Order.class));
        setReturnDataType(DataTypeFactory.STRING);
        setName("OrderToHTML");
    }
}
```

Because the source type is specified, you don't need to do any type checking in your `transform()` method. However, if you add more than one source type, you do need to check for each type in your `transform()` method.

Notice that the above code sets the name of the transformer. Usually, you set the transformer name in the XML configuration when the transformer is declared. If no name is set, Mule generates a name based on the first source type and return class, such as "OrderToString" if the above example did not specify the name.

## Using Transformer Lifecycle Methods

All objects in Mule have lifecycles associated with them. For transformers, there are two lifecycle methods that are most useful.

By default `AbstractMessageAwareTransformer` and `AbstractTransformer` both implement the `org.mule.api.lifecycle.Initialisable` interface. After all bean properties are set on the transformer (if any), the `doInitialise()` method is called. This is useful for transformers to do any initialization or validation work. For example, if a transformer requires that an external file be loaded before the transformer can operate, you would load the file via the `doInitialise()` method.

If you want your transformer to clear up resources when the transformer is no longer needed, your transformer must implement `org.mule.api.lifecycle.Disposable` and implement the `dispose()` method.

## Creating Discoverable Transformers

Mule can perform automatic transformations. For example, when you call the `getPayload()` method on a `MuleMessage` and pass in the required type as follows:

```
Document doc = (Document)muleMessage.getPayload(org.dom4j.Document.class);
```

Mule looks at the current payload type and tries to find a transformer that can convert it to an `org.dom4j.Document` object. Mule provides several standard transformers for switching between common types such as strings, `byte[]`, `InputStream`, and more. Also, transports usually have transformers for specific message types such as `JMSMessage` or `HttpRequest`. When creating a custom transformer, you can set its priority higher than the standard transformers so that it takes precedence.

To make a transformer discoverable, it must implement `org.mule.api.transformer.DiscoverableTransformer`. This interface introduces two methods, `getPriorityWeighting()` and `setPriorityWeighting(int weighting)`. Weighting resolves conflicts when two or more transformers match the search criteria. The weighting is a number between 1 and 10, with 10 being the highest priority. As a rule, Mule transformers have a priority of 1 and should always have a lower priority than any custom transformers.

For example, to make the `OrderToHtmlTransformer` discoverable, you would define it as follows:

```
public class OrderToHtmlTransformer
extends AbstractMessageAwareTransformer
implements DiscoverableTransformer
{
    private int weighting = DiscoverableTransformer.DEFAULT_PRIORITY_WEIGHTING + 1;

    int getPriorityWeighting() {
        return weighting;
    }

    void setPriorityWeighting(int weighting) {
        this.weighting = weighting;
    }
}
```

This transformer converts from an `Order` object to a `String`, which the standard `ObjectToString` transformer also does, but `ObjectToHtml` will be used because it has a higher priority weighting. You could test this as follows:

```
MuleMessage message = new DefaultMuleMessage(new Order(...));
String html = (String)muleMessage.getPayload(java.lang.String.class);
```

### Registering the Transformer

After creating a transformer, you must register it so that Mule can discover it at runtime. You can register the transformer simply by configuring it using the `<custom-transformer>` element in your Mule configuration file.

Alternatively, if you want your transformer to be loaded automatically by Mule when your module (JAR) is on the class path, you add a `registry-bootstrap.properties` file to your JAR under the `/META-INF/services/org/mule/config` directory. The contents of `registry-bootstrap.properties` should look something like this:

```
orderToHtml=com.foo.OrderToHtml
```

When Mule starts, it will discover this bootstrap file before loading any configuration and will install any objects listed in the file into the local registry. For more information, see [Bootstrapping the Registry](#).

### Examples

To create an HTML message that includes the `transactionId` from the message header, you would extend `AbstractMessageAwareTransformer` and write the `transform()` method as follows:

```

public Object transform(MuleMessage message, String encoding) throws TransformerException
{
    Order order = (Order)message.getPayload();
    StringBuffer html = new StringBuffer();
    html.append("");
    html.append("");
    html.append("");
    html.append("Dear ").append(order.getCustomer().getName()).append(" ");
    html.append("Thank you for your order. Your transaction reference is: <strong>");
    html.append(message.getProperty("transactionId").append("</strong>"));
    html.append(" ( ");
    return html.toString();
}

```

The [Hello World](#) example defines a custom transformer called `StringToNameString`, which wraps Java string in a custom class called `NameString`:

```

package org.mule.example.hello;

import org.mule.api.transformer.TransformerException;
import org.mule.transformer.AbstractTransformer;
import org.mule.transformer.types.DataTypeFactory;

/**
 * <code>StringToNameString</code> converts from a String to a NameString object.
 */
public class StringToNameString extends AbstractTransformer
{

    public StringToNameString()
    {
        super();
        this.registerSourceType(DataTypeFactory.STRING);
        this.setReturnDataType(DataTypeFactory.create(NameString.class));
    }

    @Override
    public Object doTransform(Object src, String encoding) throws TransformerException
    {
        return new NameString((String) src);
    }
}

```

The transformer is then configured as follows:

```

<custom-transformer name="StringToNameString" class="org.mule.example.hello.StringToNameString"/>
...
<flow name="Hello World">
...
    <vm:inbound-endpoint path="greeter" transformer-refs="StringToNameString" exchange-pattern=
"request-response"/>
...

```

Alternatively you can configure transformer directly in the endpoint, as follows:

```

<flow name="Hello World">
    <vm:inbound-endpoint path="greeter" exchange-pattern="request-response">
        <custom-transformer class="org.mule.example.hello.StringToNameString" />
    </vm:inbound-endpoint>
    ...

```

Your Rating: 

Results:  0 rates

## Connecting SaaS, Social Media, and E-Commerce Using Mule Cloud Connect

### Connecting SaaS, Social Media, and E-Commerce Using Mule Cloud Connect

[ [What is Mule Cloud Connect?](#) ] [ [Using Mule Cloud Connect](#) ] [ [When to use Mule Cloud Connect](#) ] [ [Whats Next?](#) ] [ [Available Cloud Connectors](#) ]

#### What is Mule Cloud Connect?

Mule Cloud Connect is a powerful, lightweight toolset for using a number of SaaS, Social Media and E-Commerce Services with Mule ESB. Because each Cloud Connector is self-contained, you do not need to install anything aside from the Cloud Connector itself. Each Mule Cloud Connector includes a Mule specific schema which provides code completion and inline documentation.

It has always been possible to connect Mule to these services in the past using specific [transports](#). Mule Cloud Connect creates a layer on top of these services to simplify the use of these services even further.

When used with Flows users can create orchestration which can span multiple services and make use of all the functionality provided by Mule. This includes the ability to use Mule routing, exception handling, asynchronous execution abilities - just to name a few.

#### Using Mule Cloud Connect

It is simple to get started with Mule's Cloud Connect. Each Cloud Connector comes with full documentation:

- examples usage
- full configuration reference
- snippets to copy and paste for maven dependencies and xml namespace declarations
- download link

Once installed a Cloud Connect is used like any other feature in Mule. For example:

#### Salesforce Query

```
<salesforce:query query="SELECT Id, Name FROM Account WHERE Name = 'Sandy'" />
```

This uses the SalesForce connector to make a call to SalesForce's query API with the specified query. This will then return a query result from SalesForce which can then be easily referenced.

Each Cloud Connector provides its own XML schema to be used in Mule which is backed by POJO's. Since these Cloud Connectors are POJO's, they can also be used outside of Mule.

#### When to use Mule Cloud Connect

Cloud Connect provide an alternative type of connector to a [Mule Transport](#). Transports typically represent communication protocols whereas Cloud Connect are better suited for connecting Mule to an application API built on top of an underlying protocol already supported by Mule.



**Application API** - An application API allows you to create, query or modify data and services within a specific application. Some popular examples include Twitter, Facebook, LinkedIn, Amazon Web Services, Google, and SalesForce. Today most application APIs are commonly exposed through REST or SOAP web services.

#### Whats Next?

The first step in working with Cloud Connectors is to check out the [Getting Started with Cloud Connect](#) guide. This guide explains how cloud connectors are configured and will get you started using them, whether you use maven or the MuleIDE.

Once you know how to configure cloud connector, you'll be interesting in learning about [Integrating with Cloud Connect](#). Here we'll show you how to use cloud connectors in your Mule flows to implement a variety of different cloud connector integration scenarios.

## Available Cloud Connectors

A categorized list of [available cloud connectors](#) is available and the documentation for each connector will provide you with everything you need to know to successfully use the connector with Mule. This includes how and where to download the connector, details of how to obtain any keys for the service, and information on how to include and use the cloud services operations in your Mule flows.

If a connector is not available for a cloud service you wish to connect to, consider creating your own connector using our easy to follow [How to Build a Cloud Connector](#) guide. This step by step guide includes examples and a tool to help speed up the creation of your cloud connector projects.

Your Rating:  Results:  1 rates

## Integrating with Cloud Connect

### Integrating with Cloud Connect

#### Introduction

At this point you should know how to [configure Mule Cloud Connectors](#), how to add them to a new or existing project.

You're ready to do some integration.

#### Cloud Connector or Transport?

Before learning how to use a Mule Cloud Connector, consider for a moment the various implications of using transports versus cloud connectors.

#### **What do Mule Cloud Connectors and transports have in common?**

- Both process messages
- Both communicate with a remote systems using a protocol
- Both can be configured as part of a Mule flow

#### **How do Cloud Connectors differ from transports?**

- Transports send messages. Cloud connectors **invoke SaaS API operations**.
- Transports implement a specific protocol. Cloud connectors use a protocol under the covers (you don't need to know what's used or work with it)
- Transports are generic. Cloud connectors are service specific. (Service specific transport configuration is reused inside the cloud connector implementation)

In summary **Cloud Connectors** provide a higher level of abstraction that **simplifies** the integration with SAAS applications and cloud infrastructure while promoting **reuse**.

You can mix and match cloud connectors and transports freely within your Mule flows.



**Mule Flow** A Mule Flow is a very flexible way of building integration solutions that is both powerful and intuitive.

### Cloud Connector Integration Scenarios

The following samples show how Cloud Connectors can be used in Mule Flows to perform integration with cloud services. These examples use the [Twitter](#) cloud connector.

#### 1) Proxy a Cloud Operation

In this very first example we simply re-expose a cloud connector operation over a Mule transport of our choice. In the configuration snippet below we a [http](#) inbound endpoint is configured followed by a our Twitter "public-timeline" operation. When we [test this flow in MuleIDE](#), Mule will listen on localhost port 8080 for any requests. When one is received, the Twitter public timeline operation is invoked, and it responds with the results. In the case of Twitter, the results are formatted according to the Twitter connector configuration.

This example also uses the `logger` to output the result set. This optional element can be used for debugging.

```
<twitter:config name="twitter" format="XML"/>

<flow name="twitterPublicTimelineProxy">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <twitter:public-timeline/>
    <logger message="#[payload]" level="INFO"/>
</flow>
```

## 2) Proxy a Cloud Operation with parameters

This example is very similar to the one above, but this time a parameter is provided and used in the invocation of the cloud service operation. To invoke the flow shown below you would use the following url `http://localhost:8080/?query=mule`.

```
<twitter:config name="twitter" format="JSON"/>

<flow name="twitterSearchProxy">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <twitter:search query="#[header:INBOUND:query]"/>
    <logger message="#[payload]" level="INFO"/>
</flow>
```

This example uses a query parameter to pass in the Twitter search term, but if a message was sent to the inbound http endpoint using POST, this value could easily be extracted from the message using another expression evaluator such as XPATH, as shown here:

```
<flow name="twitterSearchProxy">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <twitter:search query="#[xpath:/my/element/with/query]"/>
</flow>
```

This is an example of how an incoming message is used to invoke a cloud service operation using expressions to extract values from the message.

## 3) Invoke a Cloud Service Operation asynchronously

You've now seen two examples where a cloud service operation is invoked as part of a synchronous flow that implements a proxy scenario in which where the result from the cloud service is returned to the client.

Cloud operations can also be used in asynchronous Mule flows, when the result doesn't need to be returned over the same channel.

Next, consider two examples that show asynchronous flow, that is, where the message flow is *one-way*. The first example snippet uses a `http endpoint` which by default uses a *request-response* exchange pattern. In this case, however, the `<async>` element is used to invoke the cloud connect asynchronously.

```

<flow name="asyncUpdateTwitterStatus">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <async>
        <twitter:update-stats status="#{json:/node/with/status}" />
    </async>
</flow>

```

The second example shows the use of a [JMS endpoint|MULE3USER:JMS Transport] to receive new statuses from a JMS queue. JMS endpoints are one-way by default, meaning therefore, you don't need to use the <async> element here.

```

{code:xml}
<flow name="asyncUpdateTwitterStatus">
    <jms:inbound-endpoint queue="twitterStatusQueue"/>
    <twitter:update-stats status="#{xpath:/element/with/status}" />
</flow>

```

#### 4) Polling a Cloud Service

Another way to interact with cloud services is to poll them. Polling can be invoke an outbound endpoint (or any message process) at a defined frequency with the result being used as a new message that invokes the flow. This example shows the use of <poll>:

```

<flow name="cloudPolling">
    <poll frequency="60000">
        <twitter:search query="mule" />
    </poll>
    ...
</flow>

```

The polling example above will produce Twitter search results for "mule" every 60 seconds. You may receive repeated results in each poll.

Because it's part of Mule, we can quickly and easily make this polling into a stream of unique tweets by adding a few lines of XML.

Using [splitter](#) and an [idempotent filter](#), this is quite straightforward. The splitter splits out the search results into individual tweets, and the idempotent filter removes any duplicate tweets using the tweet id as a unique id.

```

<flow name="cloudPolling">
    <poll frequency="60000">
        <twitter:search query="mule" />
    </poll>
    <splitter evaluator="json" expression="results" />
    <idempotent-message-filter idExpression="#{json:id}" />
    <logger message="@#[json:from_user] - #[json:text]" level="INFO" />
</flow>

```

#### 6) Enrich messages using a cloud service

So far, the scenarios you've seen use either a cloud service as a source of messages for the flow or use a cloud service in the flow to perform some operation. What if you want to use a cloud service to add to an existing message based on some information that already exists? This is called enrichment, and again, the brevity of the XML should impress you:

```

<twitter:config name="twitter"/>

<flow name="cloudEnrichment">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <enricher target="#{header:userLang} source="#{json:lang}>
        <twitter:user screenName="#{xpath:/element/with/screenName}" />
    </enricher>
    <http:outbound-endpoint host=".." port="" />
</flow>

```

In this example the message is enriched by adding an a header called `userLang` to the message with the user language code as retrieved from Twitter.

## 7) Routing using cloud services

Instead of enriching the message directly, you can set a variable in the Mule Flow that can be used later by the expression. In this case, the expression is on the `<when>` choice element, and it's used to route the message.

```

<twitter:config name="twitter"/>

<flow name="cloudRouting">
    <http:inbound-endpoint host="localhost" port="8080"/>
    <enricher target="#{variable:userLang} source="#{json:lang}>
        <twitter:user screenName="#{xpath:/element/with/screenName}" />
    </enricher>
    <choice>
        <when evaluator="variable" expression="userLang=en">
            ..
        </when>
        <when evaluator="variable" expression="userLang=es">
            ..
        </when>
        <otherwise>
            ..
        </otherwise>
    </choice>
</flow>

```

The Twitter cloud connect operation that looks up user information is used here to determine the user's language and then route the source message based on this information.

The **variable** expression evaluator/enricher are used to store and retrieve flow-scoped variables.

## 8) Cloud to Cloud Integration

These examples illustrate all the pieces you need to integrate different cloud services in a flow. Remember you have all of Mule **filters**, **transformers** and **routers** etc. to help you implement your flows using all of the **available cloud connectors**.

Your Rating:  Results:  0 rates

## Available Cloud Connectors

[ Available Cloud Connectors ] [ Social Media ] [ CRM Systems ] [ Ecommerce Services ] [ Payment Services ]

## Available Cloud Connectors

A number of Cloud Connectors are offered by MuleSoft as listed below. This list is always growing and we welcome new connectors from the community as well. If you would like to get started with your own connector please check out [How to Build a Cloud Connector](#).

Cloud connectors are grouped by service category.

### Social Media

The proliferation of social media has created opportunities for companies to use APIs to grant their site visitors multiple ways to interact.

MuleSoft offers the following Social Media cloud connectors:

- Twitter connector
- Flickr connector

## CRM Systems

The ubiquity of CRM is now matched by the growing expectation of universal access across devices and connection scenarios.

MuleSoft offers the following for connectivity to CRM systems:

- SalesForce connector

## Ecommerce Services

Running an online store is not a simple task but with the ecommerce products and services today it is simple to setup a ecommerce presence. Setting up a ecommerce presence is only the first step and these products and services need to also integrate with the number of other systems needed to run a business.

MuleSoft offers the following connectivity to ecommerce systems:

- Magento Connector

## Payment Services

New payment capabilities sprout up throughout the online ecosystem daily, along with different bundled offerings and the systems and protocols used to access them.

MuleSoft offers the following connectivity to Payment Services:

- Authorize.Net connector
- Cybersource connector

Your Rating: 

Results:  1 rates

## Authorize.Net

# Authorize.Net<sup>®</sup>

a CyberSource solution

Cloud Connector

Name	Authorize.Net
API URL	<a href="http://www.authorize.net/support/AIM_guide.pdf">http://www.authorize.net/support/AIM_guide.pdf</a>
Download	<a href="http://repository.muleforge.org/org/mule/modules/mule-module-authorize/1.0/mule-module-authorize-1.0.jar">http://repository.muleforge.org/org/mule/modules/mule-module-authorize/1.0/mule-module-authorize-1.0.jar</a>
State	Complete
Supported Mule Versions	3.1.0 - Download!

### Table of Contents

- Introduction
- Install
- Configure
- How to Setup a Test Account
- Example Usage

### Introduction

The Authorize.Net connector provides an easy to use interface to the Authorize.Net payment services. This allows users to create flows which can authorize, capture, void and credit credit card transactions.

## Install

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

### To Install Connector for All Applications

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

### To Install for a Single Application

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```
<repository>
    <id>muleforge-repo</id>
    <name>MuleForge Repository</name>
    <url>http://repository.muleforge.org</url>
    <layout>default</layout>
</repository>
```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```
<dependency>
    <groupId>org.mule.modules</groupId>
    <artifactId>mule-module-authorize</artifactId>
    <version>1.0</version>
</dependency>
```



Are you using the Mule Maven Plugin to build your application? If so please make sure to list it in the inclusions section of the plug-in, for example:

```
<plugin>
    <groupId>org.mule.tools</groupId>
    <artifactId>maven-mule-plugin</artifactId>
    <version>1.5</version>
    <extensions>true</extensions>
    <configuration>
        <inclusions>
            <inclusion>
                <groupId>org.mule.modules</groupId>
                <artifactId>mule-module-authorize</artifactId>
            </inclusion>
        </inclusions>
    </configuration>
</plugin>
```

## Configure

To use the Authorize.Net connector within a flow the namespace to the connector must be included. The resulting flow will look similar to the following:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:auth="http://www.mulesoft.org/schema/mule/authorizenet"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/authorizenet
          http://www.mulesoft.org/schema/mule/authorizenet/3.1/mule-authorizenet.xsd">

```

Once the namespace has been included then the Authorize.Net connector can then be configured by using the config element of the connector. The following is an example of a config element for the SalesForce connector:

```

<auth:config name="auth" merchantLogin="${merchantLogin}" merchantTransactionKey=
"${merchantTransactionKey}" testMode="true"/>

```

Within the config element a name is given along with any required or optional parameters. In this instance a merchantLogin, MerchantTransactionKey and testMode need to be supplied. The next section will cover how to obtain this information for testing purposes.

### **How to Setup a Test Account**

Authorize.Net allows for the creation of a developer account which can be used for development and testing purposes. To sign up for an account go to <https://developer.authorize.net/testaccount>.

### **Example Usage**

The following is an example of processing a payment using the Authorize.NET connector. For a full reference for all the functions offered by the connector please refer to the full schema documentation below:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:auth="http://www.mulesoft.org/schema/mule/authorizenet"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/authorizenet
          http://www.mulesoft.org/schema/mule/authorizenet/3.1/mule-authorizenet.xsd">

    <auth:config merchantLogin=<YOUR MERCHANT LOGIN> merchantTransactionKey=<YOUR MERCHANT
TRANSACTION KEY> name="auth" testMode="true"/>

    <flow name="payment">
        <http:inbound-endpoint address="http://localhost:9898/payment" exchange-pattern=
"request-response"/>

        <auth:authorization-and-capture cardNumber="3700000000000002" expDate="12/12" amount="400"/>

        <expression-transformer>
            <return-argument evaluator="bean" expression="responseReasonText"/>
        </expression-transformer>
    </flow>
</mule>

```

### **Schema Documentation**

cache: Unexpected program error: java.lang.NullPointerException

### **Module (schemadoc:page-title not set)**

## **Config**

### **Attributes of <config...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
merchantLogin	string	no		The merchant login used for all transactions.
merchantTransactionKey	string	no		The merchant transaction key used for all transactions.
testMode	string	no		Setting this to true will force the connector to use the test gateway for all transactions.

### **Child Elements of <config...>**

Name	Cardinality	Description
------	-------------	-------------

## **Authorization and capture**

This is the most common type of credit card transaction and is the default payment gateway transaction type. The amount is sent for authorization, and if approved, is automatically submitted for settlement.

### **Attributes of <authorization-and-capture...>**

Name	Type	Required	Default	Description
amount	string	no		The amount to be charged.
cardNumber	string	no		The credit card number to be used for the transaction.
expDate	string	no		The expiration date associated with the credit card used.
config-ref	string	no		

### **Child Elements of <authorization-and-capture...>**

Name	Cardinality	Description
------	-------------	-------------

## **Authorization only**

This transaction type is sent for authorization only. The transaction will not be sent for settlement until the credit card transaction type Prior Authorization and Capture (see definition below) is submitted, or the transaction is submitted for capture manually in the Merchant Interface. For more information about capturing Authorization Only transactions in the Merchant Interface, see the Merchant Integration Guide at <http://www.authorize.net/support/merchant/>.

If action for the Authorization Only transaction is not taken on the payment gateway within 30 days, the authorization expires and is no longer available for capture. A new Authorization Only transaction would then have to be submitted to obtain a new authorization code.

Merchants can submit Authorization Only transactions if they want to verify the availability of funds on the customer's credit card before finalizing the transaction. This transaction type can also be submitted if the merchant does not currently have an item in stock or wants to review orders before shipping goods.

### **Attributes of <authorization-only...>**

Name	Type	Required	Default	Description
amount	string	no		The amount to be charged.
cardNumber	string	no		The credit card number to be used for the transaction.

expDate	string	no		The expiration date associated with the credit card used.
config-ref	string	no		

#### Child Elements of <authorization-only...>

Name	Cardinality	Description
------	-------------	-------------

#### **Prior authorization and capture**

This transaction type is sent for authorization only. The transaction will not be sent for settlement until the credit card transaction type Prior Authorization and Capture (see definition below) is submitted, or the transaction is submitted for capture manually in the Merchant Interface. For more information about capturing Authorization Only transactions in the Merchant Interface, see the Merchant Integration Guide at <http://www.authorize.net/support/merchant/>.

If action for the Authorization Only transaction is not taken on the payment gateway within 30 days, the authorization expires and is no longer available for capture. A new Authorization Only transaction would then have to be submitted to obtain a new authorization code.

Merchants can submit Authorization Only transactions if they want to verify the availability of funds on the customer's credit card before finalizing the transaction. This transaction type can also be submitted if the merchant does not currently have an item in stock or wants to review orders before shipping goods.

#### Attributes of <prior-authorization-and-capture...>

Name	Type	Required	Default	Description
amount	string	no		Up to 15 digits with a decimal point (no dollar symbol) Ex. 8.95 This is the total amount and must include tax, shipping, and any other charges. The amount can either be hard-coded or posted to a script.
transactionId	string	no		The payment gateway assigned transaction ID of an original transaction
config-ref	string	no		

#### Child Elements of <prior-authorization-and-capture...>

Name	Cardinality	Description
------	-------------	-------------

#### **Capture only**

This transaction type is used to complete a previously authorized transaction that was not originally submitted through the payment gateway or that requires voice authorization.

#### Attributes of <capture-only...>

Name	Type	Required	Default	Description
authenticationCode	string	no		The authorization code of an original transaction not authorized on the payment gateway
amount	string	no		The amount to be charged.
cardNumber	string	no		The credit card number to be used for the transaction.
expDate	string	no		The expiration date associated with the credit card used.
config-ref	string	no		

#### Child Elements of <capture-only...>

Name	Cardinality	Description
------	-------------	-------------

#### **Credit**

This transaction type is used to refund a customer for a transaction that was originally processed and

successfully settled through the payment gateway.

#### Attributes of <credit...>

Name	Type	Required	Default	Description
amount	string	no		The amount to be credited.
cardNumber	string	no		The credit card number to be used for the transaction.
expDate	string	no		The expiration date associated with the credit card used.
transactionId	string	no		The payment gateway-assigned transaction ID of an original transaction
config-ref	string	no		

#### Child Elements of <credit...>

Name	Cardinality	Description

#### **Void transaction**

This transaction type can be used to cancel either an original transaction that is not yet settled, or an entire order composed of more than one transaction. A void prevents the transaction or order from being sent for settlement. A Void can be submitted against any other transaction type.

#### Attributes of <void-transaction...>

Name	Type	Required	Default	Description
transactionId	string	no		The payment gateway-assigned transaction ID of an original transaction
config-ref	string	no		

#### Child Elements of <void-transaction...>

Name	Cardinality	Description

## Cybersource

# CyberSource®

#### Cloud Connector

Name	CyberSource
API URL	<a href="http://apps.cybersource.com/cgi-bin/pages/dev_kits.cgi?kit=Java/All_Platforms">http://apps.cybersource.com/cgi-bin/pages/dev_kits.cgi?kit=Java/All_Platforms</a>
Download	<a href="http://repository.cybersrc.muleforge.org/org/mule/modules/mule-module-cybersrc/1.1/mule-module-cybersrc-1.1.jar">http://repository.cybersrc.muleforge.org/org/mule/modules/mule-module-cybersrc/1.1/mule-module-cybersrc-1.1.jar</a>
State	Complete
Supported Mule Versions	3.1.0 - Download!

#### Table of Contents

- Introduction
- Install
- Configure
- How to Setup a Test Account
- Example Usage

#### **Introduction**

The CyberSource connector provides an easy to use interface to the CyberSource payment services. This allows users to create flows which can

authorize, capture, void and credit credit card transactions.

## Install

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

### To Install Connector for All Applications

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

### To Install for a Single Application

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```
<repository>
  <id>cybersrc-repo</id>
  <name>CyberSource Repository</name>
  <url>http://repository.cybersrc.muleforge.org</url>
  <layout>default</layout>
</repository>
```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-cybersrc</artifactId>
  <version>1.0</version>
</dependency>
```



Are you using the Mule Maven Plugin to build your application? If so please make sure to list it in the inclusions section of the plug-in, for example:

```
<plugin>
  <groupId>org.mule.tools</groupId>
  <artifactId>maven-mule-plugin</artifactId>
  <version>1.5</version>
  <extensions>true</extensions>
  <configuration>
    <inclusions>
      <inclusion>
        <groupId>org.mule.modules</groupId>
        <artifactId>mule-module-cybersrc</artifactId>
      </inclusion>
    </inclusions>
  </configuration>
</plugin>
```

## Configure

To use the CyberSource connector within a flow the namespace to the connector must be included. The resulting flow will look similar to the following:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:cybsrc="http://www.mulesoft.org/schema/mule/cybersource"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/cybersource
          http://www.mulesoft.org/schema/mule/cybersource/3.1/mule-cybersource.xsd">

```

Once the namespace has been included then the CyberSource connector can then be configured by using the config element of the connector. The following is an example of a config element for the SalesForce connector:

```

<cybsrc:config name="cyber" keysDirectory="${app.home}/keys/" merchantId="mule"
testMode="true"/>

```

Within the config element a name is given along with any required or optional parameters. In this instance a keysDirectory and merchantId must be specified. The keysDirectory is the directory in which the generate key which you obtain from CyberSource which is covered in the next section of how to setup a test account. The testMode option allows for the connector to go against the test gateway for development and testing purposes.

### **How to Setup a Test Account**

CyberSource allows for the creation of a developer account which can be used for development and testing purposes. To sign up for an account go to [https://apps.cybersource.com/cgi-bin/register/reg\\_form.pl](https://apps.cybersource.com/cgi-bin/register/reg_form.pl). Once you have registered for a test account you will need to generate a key which will be used for authentication to the payment gateway. Information on how to generate this key can be obtained here: [http://apps.cybersource.com/library/documentation/dev\\_guides/Simple\\_Order\\_API\\_Client\\_Java/20100219\\_Java\\_Simple\\_Order\\_API.pdf](http://apps.cybersource.com/library/documentation/dev_guides/Simple_Order_API_Client_Java/20100219_Java_Simple_Order_API.pdf).

### **Example Usage**

The following is an example of processing a payment using the CyberSource connector. For a full reference for all the functions offered by the connector please refer to the full schema documentation below:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:cybsrc="http://www.mulesoft.org/schema/mule/cybersource"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/cybersource
          http://www.mulesoft.org/schema/mule/cybersource/3.1/mule-cybersource.xsd">

    <cybsrc:config name="cybersource" keysDirectory="${app.home}/keys/" merchantId="mule" testMode="true"/>

    <flow name="payment">
        <http:inbound-endpoint address="http://localhost:9898/payment" exchange-pattern="request-response"/>

        <cybsrc:make-payment address="30 Maiden Lane" amount="300"
            city="San Francisco" country="US"
            creditCardNumber="378282246310005" email="muleman@mulesoft.org"
            expirationDate="12/2012" firstName="Mule"
            lastName="Man" postalCode="94108"
            state="CA"/>
        <choice>
            <when evaluator="groovy" expression="payload['decision']=='ACCEPT' ">
                <expression-transformer>
                    <return-argument evaluator="string" expression="Payment was successful!"/>
                </expression-transformer>
            </when>
            <otherwise>
                <expression-transformer>
                    <return-argument evaluator="string" expression="There was a problem with your
payment!"/>
                </expression-transformer>
            </otherwise>
        </choice>
    </flow>
</mule>

```

## Schema Documentation

cache: Unexpected program error: java.lang.NullPointerException

## Module (schemadoc:page-title not set)

### Config

#### Attributes of <config...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
merchantId	string	no		The merchant Id used for all transactions.
keysDirectory	string	no		The path to the directory containing the p12 key needed for all transactions.
testMode	string	no		Setting this to true will force the connector to use the test gateway for all transactions.

#### Child Elements of <config...>

Name	Cardinality	Description
------	-------------	-------------

#### **Make payment**

This is the most common type of credit card transaction and is the default payment gateway transaction type. The amount is sent for authorization, and if approved, is automatically submitted for settlement.

#### Attributes of <make-payment...>

Name	Type	Required	Default	Description
amount	string	no		The amount to be charged.
creditCardNumber	string	no		The credit card number to be used for the transaction.
expirationDate	string	no		The expiration date associated with the credit card used.
firstName	string	no		First name associated with the credit card.
lastName	string	no		Last name associated with the credit card.
address	string	no		Billing address that is associated with the credit card.
city	string	no		Billing city that is associated with the credit card.
state	string	no		Billing state that is associated with the credit card.
postalCode	string	no		Billing postal code that is associated with the credit card.
email	string	no		Email address associated with the credit card.
country	string	no		Billing country that is associated with the credit card.
config-ref	string	no		

#### Child Elements of <make-payment...>

Name	Cardinality	Description
------	-------------	-------------

#### **Authorize payment**

Online authorization means that when you submit an order using a credit card, you receive an immediate confirmation about the availability of the funds. If the funds are available, the issuing bank reduces your customer's open to buy, which is the amount of credit available on the card. Most of the common credit cards are processed online. For online authorizations, you will typically start the process of order fulfillment soon after you receive confirmation of the order.

Online authorizations expire with the issuing bank after a specific length of time if they have not been captured and settled. Most authorizations expire within five to seven days. The issuing bank determines the length of time.

#### Attributes of <authorize-payment...>

Name	Type	Required	Default	Description
amount	string	no		The amount to be charged.
creditCardNumber	string	no		The credit card number to be used for the transaction.
expirationDate	string	no		The expiration date associated with the credit card used.
firstName	string	no		First name associated with the credit card.
lastName	string	no		Last name associated with the credit card.
address	string	no		Billing address that is associated with the credit card.
city	string	no		Billing city that is associated with the credit card.

state	string	no		Billing state that is associated with the credit card.
postalCode	string	no		Billing postal code that is associated with the credit card.
email	string	no		Email address associated with the credit card.
country	string	no		Billing country that is associated with the credit card.
config-ref	string	no		

#### Child Elements of <authorize-payment...>

Name	Cardinality	Description
------	-------------	-------------

#### *Capture payment*

When you are ready to fulfill a customer's order and transfer funds from the customer's bank to your bank, capture the authorization for that order.

If you can fulfill only part of a customer's order, do not capture the full amount of the authorization. Capture only the cost of the items that you ship. When you ship the remaining items, request a new authorization, then capture the new authorization.

Due to the potential delay between authorization and capture, the authorization might expire with the issuing bank before you request capture. Most authorizations expire within five to seven days. If an authorization expires with the issuing bank before you request the capture, your bank or processor might require you to resubmit an authorization request and include a request for capture in the same message.

#### Attributes of <capture-payment...>

Name	Type	Required	Default	Description
authRequestId	string	no		The authorization code of an original transaction not authorized on the payment gateway
amount	string	no		The amount to be charged.
config-ref	string	no		

#### Child Elements of <capture-payment...>

Name	Cardinality	Description
------	-------------	-------------

#### *Credit*

CyberSource supports credits for all processors except CyberSource Latin American Processing.

Request a credit when you need to give the customer a refund. When your request for a credit is successful, the issuing bank for the credit card takes money out of your merchant bank account and returns it to the customer. It usually takes two to four days for your acquiring bank to transfer funds from your merchant bank account.

#### Attributes of <credit...>

Name	Type	Required	Default	Description
amount	string	no		The amount to be credited.
cardNumber	string	no		The credit card number to be used for the transaction.
expDate	string	no		The expiration date associated with the credit card used.
orderRequestToken	string	no		The payment gateway-assigned transaction ID of an original transaction
config-ref	string	no		

#### Child Elements of <credit...>

Name	Cardinality	Description
------	-------------	-------------

### **Void transaction**

This transaction type can be used to cancel either an original transaction that is not yet settled, or an entire order composed of more than one transaction. A void prevents the transaction or order from being sent for settlement. A Void can be submitted against any other transaction type.

#### Attributes of <void-transaction...>

Name	Type	Required	Default	Description
requestId	string	no		The payment gateway-assigned request ID of an original transaction
orderRequestToken	string	no		The payment gateway-assigned order Request Token of an original transaction
config-ref	string	no		

#### Child Elements of <void-transaction...>

Name	Cardinality	Description
------	-------------	-------------

## Flickr



<b>Name</b>	Flickr Connector
<b>API URL</b>	<a href="http://www.flickr.com/services/api/">http://www.flickr.com/services/api/</a>
<b>Download</b>	<a href="http://repository.muleforge.org/release/org/mule/modules/mule-module-flickr/">http://repository.muleforge.org/release/org/mule/modules/mule-module-flickr/</a>
<b>State</b>	
<b>Supported Mule Versions</b>	3.1.0 - Download!

### **Table of Contents**

- Intro
- Install
- Configure
- How to get the Flickr api and secret keys
- Example Usage
- Schema Documentation

### **Intro**

The Flickr connector provides an easy to integrate with the Flickr API using Mule flows.

### **Install**

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

#### **To Install Connector for All Applications**

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

#### **To Install for a Single Application**

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge snapshot repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```

<repository>
  <id>muleforge-releases</id>
  <name>MuleForge Release Repository</name>
  <url>http://repository.muleforge.org/release/</url>
</repository>

```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```

<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-flickr</artifactId>
  <version>1.1</version>
</dependency>

```

## Configure

To use the Flickr connector within a flow the namespace for the connector must be included. The resulting configuration file header will look similar to the following:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:flickr="http://www.mulesoft.org/schema/mule/flickr"
  xsi:schemaLocation="
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/flickr
    http://www.mulesoft.org/schema/mule/flickr/1.0/mule-flickr.xsd">

```

Once the namespace has been included then the Flickr connector can then be configured by using the config element of the connector. The following is an example of a config element for the Flickr connector:

```
<flickr:config name="flickr" apiKey="XXX" secretKey="XXX" format="JSON" />
```

Within the config element a name is given along with any required or optional parameters. In this instance a secret key, and api key and response format need to be supplied.

## How to get the Flickr api and secret keys

*Describe all the steps required to create accounts, register application, request keys etc. in order to use the connector in Mule.*

## Example Usage

The following are examples of the usage of the Flickr connector. For a full reference please check out the schema reference. For more information of different integration scenarios you can implement using cloud connectors see [Integrating with Cloud Connect](#).

*Show 2+ examples usages of this connector with explanations*

## Schema Documentation

The following is the full schema documentation for the Flickr connector:

- [Module \(schemadoc:page-title not set\)](#)

cache: Unexpected program error: java.lang.NullPointerException

## Module (schemadoc:page-title not set)

This schema was auto-generated. Do not edit.

## Config

#### Attributes of <config...>

Name	Type	Required	Default	Description
apiKey	string	yes		
secretKey	string	yes		
format	FORMATEnum	yes		

#### Child Elements of <config...>

Name	Cardinality	Description

#### **Search tags**

Return a list of photos with one or more matching tags. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

#### Attributes of <search-tags...>

Name	Type	Required	Default	Description
tags	string	no		A comma-delimited list of tags. Photos with one or more of the tags listed will be returned.
tagMode	string	no		

#### Child Elements of <search-tags...>

Name	Cardinality	Description

#### **Search**

Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

#### Attributes of <search...>

Name	Type	Required	Default	Description
text	string	no		A free text search. Photos who's title, description or tags contain the text will be returned.

#### Child Elements of <search...>

Name	Cardinality	Description

#### **Paged search**

Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

#### Attributes of <paged-search...>

Name	Type	Required	Default	Description
text	string	no		A free text search. Photos who's title, description or tags contain the text will be returned.
perPage	integer	no		
page	integer	no		(Optional) - The page of results to return. If this argument is omitted, it defaults to 1.

#### Child Elements of <paged-search...>

Name	Cardinality	Description
------	-------------	-------------

## Advanced search

Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

### Attributes of <advanced-search...>

Name	Type	Required	Default	Description
params	string	no		<p>one or more of the parameters listed below:</p> <ul style="list-style-type: none"> <li>user_id (Optional) - The NSID of the user who's photo to search. If this parameter isn't passed then everybody's public photos will be searched. A value of "me" will search against the calling user's photos for authenticated calls.</li> <li>tags (Optional) - A comma-delimited list of tags. Photos with one or more of the tags listed will be returned.</li> <li>tag_mode (Optional) - Either 'any' for an OR combination of tags, or 'all' for an AND combination. Defaults to 'any' if not specified.</li> <li>text (Optional) - A free text search. Photos whose title, description or tags contain the text will be returned.</li> <li>min_upload_date (Optional) - Minimum upload date. Photos with an upload date greater than or equal to this value will be returned. The date should be in the form of a unix timestamp.</li> <li>max_upload_date (Optional) - Maximum upload date. Photos with an upload date less than or equal to this value will be returned. The date should be in the form of a unix timestamp.</li> <li>min_taken_date (Optional) - Minimum taken date. Photos with an taken date greater than or equal to this value will be returned. The date should be in the form of a mysql datetime.</li> <li>max_taken_date (Optional) - Maximum taken date. Photos with an taken date less than or equal to this value will be returned. The date should be in the form of a mysql datetime.</li> <li>license (Optional) - The license id for photos (for possible values see the flickr.photos.licenses.getInfo method). Multiple licenses may be comma-separated.</li> <li>sort (Optional) - The order in which to sort returned photos. Defaults to date-posted-desc (unless you are doing a radial geo query, in which case the default sorting is by ascending distance from the point specified). The possible values are: date-posted-asc, date-posted-desc, date-taken-asc, date-taken-desc, interestingness-desc, interestingness-asc, and relevance.</li> <li>privacy_filter (Optional) - Return photos only matching a certain privacy level. This only applies when making an authenticated call to view photos you own. Valid values are: 1 public photos 2 private photos visible to friends 3 private photos visible to family 4 private photos visible to friends &amp; family 5 completely private photos</li> <li>&lt;p&gt; bbox (Optional) - A comma-delimited list of 4 values defining the Bounding Box of the area that will be searched. &lt;p&gt; The 4 values represent the bottom-left corner of the box and the top-right corner, minimum_longitude, minimum_latitude, maximum_longitude, maximum_latitude. &lt;p&gt; Longitude has a range of -180 to 180 , latitude of -90 to 90. Defaults to -180, -90, 180, 90 if not specified. &lt;p&gt; Unlike standard photo queries, geo (or bounding box) queries will only return 250 results per page. &lt;p&gt; Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component.</li> <li>&lt;p&gt; A tag, for instance, is considered a limiting agent as are user defined min_date_taken and min_date_upload parameters. If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future).</li> <li>accuracy (Optional) Recorded accuracy level of the location information. Current range is 1-16 : &lt;p&gt; World level is 1 Country is ~3 Region is ~6 City is ~11 Street is ~16 &lt;p&gt; Defaults to maximum value if not specified.</li> <li>safe_search (Optional) Safe search setting: 1 for safe. 2 for moderate. 3 for restricted. &lt;p&gt; (Please note: Un-authed calls can only see Safe content.)</li> <li>content_type (Optional) - Content Type setting: 1 for photos only. 2 for screenshots only. 3 for 'other' only. 4 for photos and screenshots. 5 for screenshots and 'other'. 6 for photos and 'other'. 7 for photos, screenshots, and 'other' (all).</li> <li>&lt;p&gt; machine_tags (Optional) - Aside from passing in a fully formed machine tag, there is a special syntax for searching on specific properties : &lt;p&gt; Find photos using the 'dc' namespace : "machine_tags" =&gt; "dc." Find photos with a title in the 'dc' namespace : "machine_tags" =&gt; "dc:title=" Find photos titled "mr. camera" in the 'dc' namespace : "machine_tags" =&gt; "dc:title=\"mr. camera\"". Find photos whose value is "mr. camera" : "machine_tags" =&gt; ":=\"mr. camera\"" Find photos that have a title, in any namespace : "machine_tags" =&gt; ":title=" <b>Find photos that have a title, in any namespace, whose value is "mr. camera"</b> : "machine_tags" =&gt; ":title=\"mr. camera\"" Find photos, in the 'dc' namespace whose value is "mr. camera" : "machine_tags" =&gt; "dc:*=\"mr. camera\"". &lt;p&gt; Multiple machine tags may be queried by passing a comma-separated list. The number of machine tags you can pass in a single query depends on the tag mode (AND or OR) that you are querying with. "AND" queries are limited to (16) machine tags. "OR" queries are limited to (8).</li> <li>machine_tag_mode (Required) - Either 'any' for an OR combination of tags, or 'all' for an AND combination. Defaults to 'any' if not specified.</li> <li>group_id (Optional) - The id of a group who's pool to search. If specified, only matching photos posted to the group's pool will be returned.</li> <li>contacts (Optional) - Search your contacts. Either 'all' or 'ff' for just friends and family.</li> <li>(Experimental) woe_id (Optional) - A 32-bit identifier that uniquely represents spatial entities. (not used if bbox argument is present).</li> <li>&lt;p&gt; Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component.</li> <li>&lt;p&gt; A tag, for instance, is considered a limiting agent as are user defined min_date_taken and min_date_upload parameters &amp;emdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future).</li> <li>place_id (Optional) - A Flickr place id. (not used if bbox argument is present).</li> <li>&lt;p&gt; Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component.</li> <li>&lt;p&gt; A tag, for instance, is considered a limiting agent as are user</li> </ul>

defined min\_date\_taken and min\_date\_upload parameters &mdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future). media (Optional) - Filter results by media type. Possible values are all (default), photos or videos has\_geo (Optional) Any photo that has been geotagged, or if the value is "0" any photo that has not been geotagged. <p/> Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component. <p/> A tag, for instance, is considered a limiting agent as are user defined min\_date\_taken and min\_date\_upload parameters &mdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future). geo\_context (Optional) - Geo context is a numeric value representing the photo's geotagginess beyond latitude and longitude. For example, you may wish to search for photos that were taken "indoors" or "outdoors". <p/> The current list of context IDs is : <p/> 0, not defined. 1, indoors. 2, outdoors. <p/> <p/> Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component. <p/> A tag, for instance, is considered a limiting agent as are user defined min\_date\_taken and min\_date\_upload parameters &mdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future). lat (Optional) - A valid latitude, in decimal format, for doing radial geo queries. <p/> Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component. <p/> A tag, for instance, is considered a limiting agent as are user defined min\_date\_taken and min\_date\_upload parameters &mdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future). lon (Optional) - A valid longitude, in decimal format, for doing radial geo queries. <p/> Geo queries require some sort of limiting agent in order to prevent the database from crying. This is basically like the check against "parameterless searches" for queries without a geo component. <p/> A tag, for instance, is considered a limiting agent as are user defined min\_date\_taken and min\_date\_upload parameters &mdash; If no limiting factor is passed we return only photos added in the last 12 hours (though we may extend the limit in the future). radius (Optional) - A valid radius used for geo queries, greater than zero and less than 20 miles (or 32 kilometers), for use with point-based geo queries. The default value is 5 (km). radius\_units (Optional) - The unit of measure when doing radial geo queries. Valid options are "mi" (miles) and "km" (kilometers). The default is "km". is\_commons (Optional) - Limit the scope of the search to only photos that are part of the Flickr Commons project. Default is false. extras (Optional) - A comma-delimited list of extra information to fetch for each returned record. Currently supported fields are: license, date\_upload, date\_taken, owner\_name, icon\_server, original\_format, last\_update, geo, tags, machine\_tags, o\_dims,

			views, media, path_alias, url_sq, url_t, url_s, url_m, url_o per_page (Optional) - Number of photos to return per page. If this argument is omitted, it defaults to 100. The maximum allowed value is 500. page (Optional) - The page of results to return. If this argument is omitted, it defaults to 1.
--	--	--	--

#### Child Elements of <advanced-search...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get photo**

Loads a Photo from Flickr as a

Unknown macro: {@link java.awt.image.BufferedImage}

#### Attributes of <get-photo...>

Name	Type	Required	Default	Description
photoUrl	string	no		the Photo URL to download. Typically this method is used in conjunction with Unknown macro: {@link #getPhotoURL(org.w3c.dom.Node, FlickrSearchIBean.IMAGE_SIZE, FlickrSearchIBean.IMAGE_TYPE)} or Unknown macro: {@link #getPhotoURL(org.w3c.dom.Node)}

#### Child Elements of <get-photo...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get photo url**

Will construct a Photo URL from a photo node retuend from a search

#### Attributes of <get-photo-url...>

Name	Type	Required	Default	Description
server	string	no		
id	string	no		
secret	string	no		

#### Child Elements of <get-photo-url...>

Name	Cardinality	Description
------	-------------	-------------

## Magento

### Magento Cloud Connector

<b>Name</b>	Magento connector
<b>API URL</b>	<a href="http://www.magentocommerce.com/wiki/doc/webservices-api/api">http://www.magentocommerce.com/wiki/doc/webservices-api/api</a>
<b>Download</b>	<a href="http://repository.muleforge.org/release/org/mule/modules/mule-module-magento/1.0/mule-module-magento-1.0.jar">http://repository.muleforge.org/release/org/mule/modules/mule-module-magento/1.0/mule-module-magento-1.0.jar</a>
<b>Source</b>	<a href="https://github.com/mulesoft/magento-connector">https://github.com/mulesoft/magento-connector</a>
<b>State</b>	The following APIS are currently supported: <ul style="list-style-type: none"> <li>• sales_order</li> <li>• sales_order_shipment</li> <li>• sales_order_invoice</li> </ul>

## Table of Contents

- Intro
- Install
- Supported APIs
- Configure
- Example Usage
- Schema Documentation

### **Intro**

The Magento connector provides an easy to integrate with the Magento API using Mule flows.

### **Install**

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

#### **To Install Connector for All Applications**

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

#### **To Install for a Single Application**

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge snapshot repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```
<repository>
  <id>muleforge-repo</id>
  <name>MuleForge Repository</name>
  <url>http://repository.muleforge.org/release</url>
  <layout>default</layout>
</repository>
```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-magento</artifactId>
  <version>1.0</version>
</dependency>
```



Are you using the Mule Maven Plugin to build your application? If so please make sure to list it in the inclusions section of the plug-in, for example:

```
<plugin>
  <groupId>org.mule.tools</groupId>
  <artifactId>maven-mule-plugin</artifactId>
  <version>1.5</version>
  <extensions>true</extensions>
  <configuration>
    <inclusions>
      <inclusion>
        <groupId>org.mule.modules</groupId>
        <artifactId>mule-module-magento</artifactId>
      </inclusion>
    </inclusions>
  </configuration>
</plugin>
```

## Supported APIs

The following Magento APIs are supported at the moment:

- sales\_order
- sales\_order\_shipment
- sales\_order\_invoice

## Configure

To use the Magento connector within a flow the namespace for the connector must be included. The resulting configuration file header will look similar to the following:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sfdc="http://www.mulesoft.org/schema/mule/magento"
      xsi:schemaLocation="
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/magento
        http://www.mulesoft.org/schema/mule/magento/3.1/mule-magento.xsd">
```

Once the namespace has been included then the Magento connector can then be configured by using the config element of the connector. The following is an example of a config element for the Magento connector:

```
<magento:config name="MagentoTest" username="joe" password="secretpasswd" address=
  "http://localhost:8080/magento"/>
```

## Example Usage

The following are examples of the usage of the Magento connector. For a full reference please check out the schema reference. For more information of different integration scenarios you can implement using cloud connectors see [Integrating with Cloud Connect](#).

### List Orders

To list orders by filters you can do the following:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:magento="http://www.mulesoft.org/schema/mule/magento"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/magento
          http://www.mulesoft.org/schema/mule/magento/3.1/mule-magento.xsd
      ">
    <magento:config name="MagentoTest" username="${username}" password="${password}" address="${address}" />

    <flow name="MagentoOrdersList">
        <inbound-endpoint address="vm://getSalesOrdersList" exchange-pattern="request-response"/>
    <magento:sales-orders-list>
        <magento:filters>
            <magento:complex-filter key="orderId">
                <magento:filter key="eq" value="#[map:payload:orderId]" />
            </magento:complex-filter>
        </magento:filters>
    </magento:sales-orders-list>
    </flow>
</mule>

```

## Schema Documentation

The following is the full schema documentation for the Magento connector:

- Module (schemadoc:page-title not set)

cache: Unexpected program error: java.lang.NullPointerException

### Module (schemadoc:page-title not set)

This schema was auto-generated. Do not edit.

#### Config

##### Attributes of <config...>

Name	Type	Required	Default	Description
name	string	no		Give a name to this configuration so it can be later referenced by config-ref.
username	string	yes		
password	string	yes		
address	string	yes		

##### Child Elements of <config...>

Name	Cardinality	Description
------	-------------	-------------

#### Add order shipment comment

Adds a comment to the shipment.

Example:

##### Attributes of <add-order-shipment-comment...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
shipmentId	string	yes		the shipment's increment id
comment	string	yes		the comment to add
sendEmail	string	no	false	if an email must be sent after shipment creation
includeCommentInEmail	string	no	false	if the comment must be sent in the email

Child Elements of <add-order-shipment-comment...>

Name	Cardinality	Description
------	-------------	-------------

### Add order shipment track

Adds a new tracking number

Example:

Attributes of <add-order-shipment-track...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
shipmentId	string	yes		the shipment id
carrierCode	string	yes		the new track's carrier code
title	string	yes		the new track's title
trackId	string	yes		

Child Elements of <add-order-shipment-track...>

Name	Cardinality	Description
------	-------------	-------------

### Cancel order

Cancels an order

Example:

Attributes of <cancel-order...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the order to cancel

Child Elements of <cancel-order...>

Name	Cardinality	Description
------	-------------	-------------

### Create order shipment

Creates a shipment for order

Example:

Attributes of <create-order-shipment...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the order increment id
comment	string	no		an optional comment
sendEmail	string	no	false	if an email must be sent after shipment creation
includeCommentInEmail	string	no	false	if the comment must be sent in the email

#### Child Elements of <create-order-shipment...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get order**

Answers the order properties for the given orderId

Example:

#### Attributes of <get-order...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the order whose properties to fetch

#### Child Elements of <get-order...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get order invoice**

Retrieves order invoice information

Example:

#### Attributes of <get-order-invoice...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
invoiceld	string	yes		the target invoiceld

#### Child Elements of <get-order-invoice...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get order shipment carriers**

Creates an invoice for the given order

Example:

#### Attributes of <get-order-shipment-carriers...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the target order id

#### Child Elements of <get-order-shipment-carriers...>

Name	Cardinality	Description
------	-------------	-------------

### **Get order shipment**

Adds a comment to the given order's invoice

Example:

#### Attributes of <get-order-shipment...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
shipmentId	string	yes		

#### Child Elements of <get-order-shipment...>

Name	Cardinality	Description
------	-------------	-------------

### **Hold order**

Puts order on hold. This operation can be reverted with unholdOrder.

Example:

#### Attributes of <hold-order...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the order to put on hold state

#### Child Elements of <hold-order...>

Name	Cardinality	Description
------	-------------	-------------

### **List orders**

Lists order attributes that match the given filtering expression.

Example

#### Attributes of <list-orders...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
filter	string	no		optional filtering expression - one or more comma-separated unary or binary predicates, one for each filter, in the form filterType(attributeName, value), for binary filters or filterType(attributeName), for unary filters, where filterType is istrue, isfalse or any of the Magento standard filters. Non-numeric values need to be escaped using simple quotes.

#### Child Elements of <list-orders...>

Name	Cardinality	Description
------	-------------	-------------

### **List orders invoices**

Lists order invoices that match the given filtering expression

Example:

#### Attributes of <list-orders-invoices...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
filter	string	no		optional filtering expression - one or more comma-separated unary or binary predicates, one for each filter, in the form filterType(attributeName, value), for binary filters or filterType(attributeName), for unary filters, where filterType is istrue, isfalse or any of the Magento standard filters. Non-numeric values need to be escaped using simple quotes.

#### Child Elements of <list-orders-invoices...>

Name	Cardinality	Description
------	-------------	-------------

#### ***List orders shipments***

Lists order shipment attributes that match the given optional filtering expression

Example:

#### Attributes of <list-orders-shipments...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
filter	string	no		optional filtering expression - one or more comma-separated unary or binary predicates, one for each filter, in the form filterType(attributeName, value), for binary filters or filterType(attributeName), for unary filters, where filterType is istrue, isfalse or any of the Magento standard filters. Non-numeric values need to be escaped using simple quotes.

#### Child Elements of <list-orders-shipments...>

Name	Cardinality	Description
------	-------------	-------------

#### ***Delete order shipment track***

Deletes the given track of the given order's shipment

```
<magento:delete-order-shipment-track  
shipmentId="#[map\payload:shipmentId]" trackId="#[map\payload:trackId]" />
```

#### Attributes of <delete-order-shipment-track...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
shipmentId	string	yes		the target shipment id
trackId	string	yes		the id of the track to delete

#### Child Elements of <delete-order-shipment-track...>

Name	Cardinality	Description
------	-------------	-------------

#### ***Add order comment***

Adds a comment to the given order id

#### Attributes of <add-order-comment...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the order id
status	string	yes		the comment status
comment	string	yes		the comment
sendEmail	string	no	false	if an email must be sent after shipment creation

Child Elements of <add-order-comment...>

Name	Cardinality	Description
------	-------------	-------------

#### ***Unhold order***

Releases order

Example:

Attributes of <unhold-order...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		the id of the order to remove from hold state

Child Elements of <unhold-order...>

Name	Cardinality	Description
------	-------------	-------------

#### ***Create order invoice***

Creates an invoice for the given order

Attributes of <create-order-invoice...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
orderId	string	yes		
comment	string	no		an optional comment
sendEmail	string	no	false	if an email must be sent after shipment creation
includeCommentInEmail	string	no	false	if the comment must be sent in the email

Child Elements of <create-order-invoice...>

Name	Cardinality	Description
------	-------------	-------------

#### ***Add order invoice comment***

Adds a comment to the given order's invoice

Example:

Attributes of <add-order-invoice-comment...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
invoiceld	string	yes		the invoice id

comment	string	yes		the comment to add
sendEmail	string	no	false	if an email must be sent after shipment creation
includeCommentInEmail	string	no	false	if the comment must be sent in the email

Child Elements of <add-order-invoice-comment...>

Name	Cardinality	Description
------	-------------	-------------

### ***Capture order invoice***

Captures and invoice

Example:

Attributes of <capture-order-invoice...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
invoiceld	string	yes		the invoice to capture

Child Elements of <capture-order-invoice...>

Name	Cardinality	Description
------	-------------	-------------

### ***Void order invoice***

Voids an invoice

Example:

Attributes of <void-order-invoice...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
invoiceld	string	yes		the invoice id

Child Elements of <void-order-invoice...>

Name	Cardinality	Description
------	-------------	-------------

### ***Cancel order invoice***

Cancels an order's invoice

Example:

Attributes of <cancel-order-invoice...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
invoiceld	string	yes		the invoice id

Child Elements of <cancel-order-invoice...>

Name	Cardinality	Description
------	-------------	-------------

### ***Create customer address***

Creates a new address for the given customer using the given address attributes

#### Attributes of <create-customer-address...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
customerId	string	yes		the customer

#### Child Elements of <create-customer-address...>

Name	Cardinality	Description
------	-------------	-------------

#### **Create customer**

Creates a customer with the given attributes

Example:

#### Attributes of <create-customer...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

#### Child Elements of <create-customer...>

Name	Cardinality	Description
------	-------------	-------------

#### **Delete customer**

Deletes a customer given its id

Example:

#### Attributes of <delete-customer...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
customerId	string	yes		the customer to delete

#### Child Elements of <delete-customer...>

Name	Cardinality	Description
------	-------------	-------------

#### **Delete customer address**

Deletes a Customer Address

Example:

#### Attributes of <delete-customer-address...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
addressId	string	yes		

#### Child Elements of <delete-customer-address...>

Name	Cardinality	Description
------	-------------	-------------

### ***Get customer***

Answers customer attributes for the given id. Only the selected attributes are retrieved

Example:

#### **Attributes of <get-customer...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
customerId	string	yes		

#### **Child Elements of <get-customer...>**

Name	Cardinality	Description
------	-------------	-------------

### ***Get customer address***

Answers the customer address attributes

Example:

#### **Attributes of <get-customer-address...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
addressId	string	yes		

#### **Child Elements of <get-customer-address...>**

Name	Cardinality	Description
------	-------------	-------------

### ***List customer addresses***

Lists the customer address for a given customer id

Example:

#### **Attributes of <list-customer-addresses...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
customerId	string	yes		the id of the customer

#### **Child Elements of <list-customer-addresses...>**

Name	Cardinality	Description
------	-------------	-------------

### ***List customer groups***

Lists all the customer groups

Example:

#### **Attributes of <list-customer-groups...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

Child Elements of <list-customer-groups...>

Name	Cardinality	Description
------	-------------	-------------

### ***List customers***

Answers a list of customer attributes for the given filter expression.

Example:

Attributes of <list-customers...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
filter	string	no		optional filtering expression - one or more comma-separated unary or binary predicates, one for each filter, in the form filterType(attributeName, value), for binary filters or filterType(attributeName), for unary filters, where filterType is true, false or any of the Magento standard filters. Non-numeric values need to be escaped using simple quotes.

Child Elements of <list-customers...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update customer***

Updates the given customer attributes, for the given customer id. Password can not be updated using this method

Example:

Attributes of <update-customer....>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
customerId	string	yes		the target customer to update

Child Elements of <update-customer...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update customer address***

Updates the given map of customer address attributes, for the given customer address

Example:

Attributes of <update-customer-address...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
addressId	string	yes		the customer address to update

Child Elements of <update-customer-address...>

Name	Cardinality	Description
------	-------------	-------------

## ***List stock items***

Retrieve stock data by product ids

Example:

### **Attributes of <list-stock-items...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

### **Child Elements of <list-stock-items...>**

Name	Cardinality	Description

## ***Update stock item***

Update product stock data given its id or sku

Example:

### **Attributes of <update-stock-item...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productIdOrSku	string	yes		the product id or sku of the product to update

### **Child Elements of <update-stock-item...>**

Name	Cardinality	Description

## ***List directory countries***

Answers the list of countries

Example:

### **Attributes of <list-directory-countries...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

### **Child Elements of <list-directory-countries...>**

Name	Cardinality	Description

## ***List directory regions***

Answers a list of regions for the given country

Example:

### **Attributes of <list-directory-regions...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
countryId	string	yes		the country code, in ISO2 or ISO3 format

### **Child Elements of <list-directory-regions...>**

Name	Cardinality	Description
------	-------------	-------------

### Add product link

Links two products, given its source and destination productIdOrSku.  
Example:

#### Attributes of <add-product-link...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		the product type
productId	string	no		the id of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product id
productSku	string	no		the sku of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product sku
productIdOrSku	string	no		the id or sku of the source product.
linkedProductIdOrSku	string	yes		the destination product id or sku.

#### Child Elements of <add-product-link...>

Name	Cardinality	Description
------	-------------	-------------

### Create product attribute media

Creates a new product media. See catalog-product-attribute-media-create SOAP method.  
Example:

#### Attributes of <create-product-attribute-media...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store
content	string	yes		the image to upload. It may be a file, an input stream or a byte array
mimeType	MediaMimeTypeEnum	yes		the mimetype
baseFileName	string	no		the base name of the new remote image

#### Child Elements of <create-product-attribute-media...>

Name	Cardinality	Description
------	-------------	-------------

### Delete product attribute media

Removes a product image. See catalog-product-attribute-media-remove SOAP method.  
Example:

#### Attributes of <delete-product-attribute-media...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
fileName	string	yes		the remote media file to delete

#### Child Elements of <delete-product-attribute-media...>

Name	Cardinality	Description
------	-------------	-------------

#### **Delete product link**

Deletes a product's link.

Example:

#### Attributes of <delete-product-link...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		link type
productId	string	no		the id of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product id
productSku	string	no		the sku of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product sku
productIdOrSku	string	no		the id or sku of the source product.
linkedProductIdOrSku	string	yes		

#### Child Elements of <delete-product-link...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get product attribute media**

Lists linked products to the given product and for the given link type.

Example:

#### Attributes of <get-product-attribute-media...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
fileName	string	yes		
storeViewIdOrCode	string	no		

#### Child Elements of <get-product-attribute-media...>

Name	Cardinality	Description
------	-------------	-------------

### **Get catalog current store view**

Answers the current default catalog store view id for this session

#### Attributes of <get-catalog-current-store-view...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

#### Child Elements of <get-catalog-current-store-view...>

Name	Cardinality	Description
------	-------------	-------------

### **Update category attribute store view**

Set the default catalog store view for this session

#### Attributes of <update-category-attribute-store-view...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
storeViewIdOrCode	string	yes		the id or code of the target store. Left unspecified for using current store the id or code of the store view to set as default for this session

#### Child Elements of <update-category-attribute-store-view...>

Name	Cardinality	Description
------	-------------	-------------

### **List category attributes**

Retrieve product image types. See catalog-product-attribute-media-types SOAP method.

Example:

#### Attributes of <list-category-attributes...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

#### Child Elements of <list-category-attributes...>

Name	Cardinality	Description
------	-------------	-------------

### **List category attribute options**

Retrieves attribute options. See catalog-category-attribute-options SOAP method.

Example:

#### Attributes of <list-category-attribute-options...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

attributeld	string	yes		the target attribute whose options will be retrieved
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <list-category-attribute-options...>

Name	Cardinality	Description
------	-------------	-------------

#### ***List product attribute media***

Retrieves product image list. See catalog-product-attribute-media-list SOAP method

Example:

Attributes of <list-product-attribute-media...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <list-product-attribute-media...>

Name	Cardinality	Description
------	-------------	-------------

#### ***List product attribute media types***

Retrieve product image types. See catalog-product-attribute-media-types SOAP method.

Example:

Attributes of <list-product-attribute-media-types...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
setId	string	yes		the setId

Child Elements of <list-product-attribute-media-types...>

Name	Cardinality	Description
------	-------------	-------------

#### ***List product attribute options***

Answers the product attribute options. See catalog-product-attribute-options SOAP method.

Example:

Attributes of <list-product-attribute-options...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

attributeld	string	yes		the target attribute whose options will be listed
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <list-product-attribute-options...>

Name	Cardinality	Description
------	-------------	-------------

### ***List product attributes***

Retrieves product attributes list. See catalog-product-attribute-list SOAP methods

Example:

Attributes of <list-product-attributes...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
setId	string	yes		

Child Elements of <list-product-attributes...>

Name	Cardinality	Description
------	-------------	-------------

### ***List product attribute sets***

Retrieves product attribute sets. See catalog-product-attribute-set-list SOAP method.

Example:

Attributes of <list-product-attribute-sets...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

Child Elements of <list-product-attribute-sets...>

Name	Cardinality	Description
------	-------------	-------------

### ***List product attribute tier prices***

Retrieve product tier prices. See catalog-product-attribute-tier-price-info SOAP Method.

Example:

Attributes of <list-product-attribute-tier-prices...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

Child Elements of <list-product-attribute-tier-prices...>

Name	Cardinality	Description
------	-------------	-------------

### ***List product link***

Lists linked products to the given product and for the given link type

#### **Attributes of <list-product-link...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		the link type
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

#### **Child Elements of <list-product-link...>**

Name	Cardinality	Description
------	-------------	-------------

### ***List product link attributes***

Lists all the attributes for the given product link type

Example:

#### **Attributes of <list-product-link-attributes...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		the product type

#### **Child Elements of <list-product-link-attributes...>**

Name	Cardinality	Description
------	-------------	-------------

### ***List product link types***

Answers product link types

Example:

#### **Attributes of <list-product-link-types...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

#### **Child Elements of <list-product-link-types...>**

Name	Cardinality	Description
------	-------------	-------------

### ***List product types***

Answers product types. See catalog-product-type-list SOAP method

Example:

#### **Attributes of <list-product-types...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

Child Elements of <list-product-types...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update product attribute media***

Updates product media. See catalog-product-attribute-media-update

Example:

Attributes of <update-product-attribute-media...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
fileName	string	yes		the name of the remote media file to update
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <update-product-attribute-media...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update product attribute tier price***

Updates a single product tier price. See catalog-product-attribute-tier-price-update  
SOAP method.

Example:

Attributes of <update-product-attribute-tier-price...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

Child Elements of <update-product-attribute-tier-price...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update product link***

Update product link

Example:

Attributes of <update-product-link...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		the link type
productId	string	no		the id of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product id
productSku	string	no		the sku of the source product. Use it instead of productIdOrSku in case you are sure the source product identifier is a product sku
productIdOrSku	string	no		the id or sku of the source product.
linkedProductIdOrSku	string	yes		the destination product id or sku.

Child Elements of <update-product-link...>

Name	Cardinality	Description
------	-------------	-------------

### ***List category products***

Lists product of a given category. See catalog-category-assignedProducts SOAP method.  
Example:

Attributes of <list-category-products...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category

Child Elements of <list-category-products...>

Name	Cardinality	Description
------	-------------	-------------

### ***Add category product***

Assign product to category. See catalog-category-assignProduct SOAP method

Attributes of <add-category-product...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category where the given product will be added
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
position	string	yes		

Child Elements of <add-category-product...>

Name	Cardinality	Description
------	-------------	-------------

### ***Create category***

Creates a new category. See catalog-category-create SOAP method.

Attributes of <create-category...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
parentId	string	yes		the parent category id
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <create-category...>

Name	Cardinality	Description
------	-------------	-------------

### **Delete category**

Deletes a category. See catalog-category-delete SOAP method

Example:

Attributes of <delete-category...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category to delete

Child Elements of <delete-category...>

Name	Cardinality	Description
------	-------------	-------------

### **Get category**

Answers category attributes. See catalog-category-info SOAP method.

Example:

Attributes of <get-category...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category whose attributes will be retrieved
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <get-category...>

Name	Cardinality	Description
------	-------------	-------------

### **List category levels**

Answers levels of categories for a website, store view and parent category

Example:

Attributes of <list-category-levels...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
website	string	no		
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store
parentCategoryId	string	no		the parent category of the categories that will be listed

#### Child Elements of <list-category-levels...>

Name	Cardinality	Description
------	-------------	-------------

#### **Move category**

Move category in tree. See catalog-category-move SOAP method. Please make sure that you are not moving category to any of its own children. There are no extra checks to prevent doing it through webservices API, and you won't be able to fix this from admin interface then .

Example:

#### Attributes of <move-category...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the id of the category to be moved
parentId	string	yes		the new parent category id
afterId	string	no		an optional category id for use as reference in the positioning of the moved category

#### Child Elements of <move-category...>

Name	Cardinality	Description
------	-------------	-------------

#### **Delete category product**

Remove a product assignment. See catalog-category-removeProduct SOAP method.

Example:

#### Attributes of <delete-category-product...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category to delete
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

#### Child Elements of <delete-category-product...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get category tree**

Answers the category tree.

See catalog-category-tree SOAP method.

#### Attributes of <get-category-tree...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
parentId	string	yes		
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

#### Child Elements of <get-category-tree...>

Name	Cardinality	Description
------	-------------	-------------

#### **Update category**

Updates a category. See catalog-category-update SOAP method

Example:

#### Attributes of <update-category...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category to update
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

#### Child Elements of <update-category...>

Name	Cardinality	Description
------	-------------	-------------

#### **Update category product**

Updates a category product

Example:

#### Attributes of <update-category-product...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
categoryId	string	yes		the category id
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
position	string	yes		the category position for ordering the category inside its level

#### Child Elements of <update-category-product...>

Name	Cardinality	Description
------	-------------	-------------

#### **List inventory stock items**

Lists inventory stock items.

Example:

#### Attributes of <list-inventory-stock-items...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

#### Child Elements of <list-inventory-stock-items...>

Name	Cardinality	Description
------	-------------	-------------

### ***Update inventory stock item***

Updates an stock inventory item

#### **Attributes of <update-inventory-stock-item...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

#### **Child Elements of <update-inventory-stock-item...>**

Name	Cardinality	Description
------	-------------	-------------

### ***Create product***

Creates a new product

Example:

#### **Attributes of <create-product...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
type	string	yes		the new product's type
set	string	yes		the new product's set
sku	string	yes		the new product's sku
storeIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

#### **Child Elements of <create-product...>**

Name	Cardinality	Description
------	-------------	-------------

### ***Delete product***

Deletes a product. See catalog-product-delete SOAP method.

Example:

#### **Attributes of <delete-product...>**

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.

#### Child Elements of <delete-product...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get product special price**

Answers a product special price. See catalog-product-getSpecialPrice SOAP method.

Example:

#### Attributes of <get-product-special-price...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

#### Child Elements of <get-product-special-price...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get product**

Answers a product's specified attributes. At least one of attributNames or additionalAttributeNames must be non null and non empty. See catalog-product-info SOAP method

#### Attributes of <get-product...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

#### Child Elements of <get-product...>

Name	Cardinality	Description
------	-------------	-------------

#### **List products**

Retrieve products list by filters. See catalog-product-list SOAP method.

Example:

#### Attributes of <list-products...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation

filter	string	no		optional filtering expression - one or more comma-separated unary or binary predicates, one for each filter, in the form filterType(attributeName, value), for binary filters or filterType(attributeName), for unary filters, where filterType is istrue, isfalse or any of the Magento standard filters. Non-numeric values need to be escaped using simple quotes.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <list-products...>

Name	Cardinality	Description
------	-------------	-------------

### **Update product special price**

Sets a product special price. See catalog-product-setSpecialPrice SOAP method

Example:

Attributes of <update-product-special-price....>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
specialPrice	string	yes		the special price to set
fromDate	string	no		optional start date of the special price period
toDate	string	no		optional end date of the special price period
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <update-product-special-price...>

Name	Cardinality	Description
------	-------------	-------------

### **Update product**

Updates a product. See catalog-category-updateProduct SOAP method

Example:

Attributes of <update-product...>

Name	Type	Required	Default	Description
config-ref	string	no		Specify which configuration to use for this invocation
productId	string	no		the id of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product id
productSku	string	no		the sku of the product. Use it instead of productIdOrSku in case you are sure the product identifier is a product sku
productIdOrSku	string	no		the id or sku of the product.
storeViewIdOrCode	string	no		the id or code of the target store. Left unspecified for using current store

Child Elements of <update-product...>

Name	Cardinality	Description
------	-------------	-------------

## SalesForce



SalesForce Cloud Connector

Name	SalesForce Connector
API URL	<a href="http://www.salesforce.com/us/developer/docs/api/index.htm">http://www.salesforce.com/us/developer/docs/api/index.htm</a>
Download	<a href="http://repository.muleforge.org/release/org/mule/modules/mule-module-sfdc/3.0.2/mule-module-sfdc-3.0.2.jar">http://repository.muleforge.org/release/org/mule/modules/mule-module-sfdc/3.0.2/mule-module-sfdc-3.0.2.jar</a>
Source	<a href="https://github.com/mulesoft/salesforce-connector">https://github.com/mulesoft/salesforce-connector</a>
State	Complete
Supported Mule Versions	3.1.0 - Download!

### Table of Contents

- Introduction
- Install
- Configure
- How to Setup a Test Account
- Example Usage
- Troubleshooting
- Schema Documentation

### Introduction

The SalesForce connector provides an easy way to interface with the SalesForce API. This allows users to create flows which can query, create and update information in SalesForce.

### Install

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

#### To Install Connector for All Applications

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

#### To Install for a Single Application

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```
<repository>
  <id>muleforge-repo</id>
  <name>MuleForge Repository</name>
  <url>http://repository.mulesoft.org/releases</url>
  <layout>default</layout>
</repository>
```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-sfdc</artifactId>
  <version>3.0.2</version>
</dependency>
```



Are you using the Mule Maven Plugin to build your application? If so please make sure to list it in the inclusions section of the plug-in, for example:

```
<plugin>
  <groupId>org.mule.tools</groupId>
  <artifactId>maven-mule-plugin</artifactId>
  <version>1.5</version>
  <extensions>true</extensions>
  <configuration>
    <inclusions>
      <inclusion>
        <groupId>org.mule.modules</groupId>
        <artifactId>mule-module-sfdc</artifactId>
      </inclusion>
    </inclusions>
  </configuration>
</plugin>
```

## Configure

To use the SalesForce connector within a flow the namespace to the connector must be included. The resulting flow will look similar to the following:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sfdc="http://www.mulesoft.org/schema/mule/salesforce"
  xsi:schemaLocation="
    http://www.mulesoft.org/schema/mule/core
    http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/salesforce
    http://www.mulesoft.org/schema/mule/salesforce/3.1/mule-salesforce.xsd">
```

Once the namespace has been included then the SalesForce connector can then be configured by using the config element of the connector. The following is an example of a config element for the SalesForce connector:

```
<sfdc:config name="salesforce" username="${login}" password="${password}" securityToken=
"${securityKey}" />
```

Within the config element a name is given along with any required or optional parameters. In this instance a username, password and security token need to be supplied. A security token can be obtained by going to [Your Name | My Personal Information | Reset My Security Token](#).

## How to Setup a Test Account

SalesForce allows for the creation of developer accounts which can then be used for development and testing purposes. To sign up for an account go to <http://developer.force.com/>.

## Example Usage

The following are examples of the usage of the SalesForce connector. For a full reference please check out the schema reference.

### Query

To perform a query against SalesForce using SalesForce's SOQL you can do the following in a config:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:sfdc="http://www.mulesoft.org/schema/mule/salesforce"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/salesforce
          http://www.mulesoft.org/schema/mule/salesforce/3.1/mule-salesforce.xsd">

    <sfdc:config name="salesforce" username="${login}" password="${password}" securityToken=
"${securityKey}" />

    <flow name="main">
        <http:inbound-endpoint address="http://localhost:9898/sftest" exchange-pattern=
"request-response"/>
        <response>
            <sfdc:mule-sobjects-to-string-transformer/>
        </response>

        <enricher target="#[variable:query]" source="#[payload]">
            <sfdc:query query="SELECT Id, Name FROM Account WHERE Name='GenePoint'" batchsize="1"/>
        </enricher>

        <expression-transformer>
            <return-argument evaluator="variable" expression="query"/>
        </expression-transformer>
    </flow>
</mule>

```

#### **Creating a SObject in SalesForce**

The following example shows a complete configuration file for creating a contact in SalesForce. The same can be done for any other type of object in SalesForce.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:sfdc="http://www.mulesoft.org/schema/mule/salesforce"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/salesforce
          http://www.mulesoft.org/schema/mule/salesforce/3.1/mule-salesforce.xsd">

    <sfdc:config name="salesforce" username="${login}" password="${password}" securityToken=
    "${securityKey}" />

    <flow name="main">
        <http:inbound-endpoint address="http://localhost:9898/sfcreate" exchange-pattern=
        "request-response"/>

        <sfdc:create type="Contact">
            <sfdc:sObject>
                <sfdc:field key="FirstName" value="Mule" />
                <sfdc:field key="LastName" value="Man" />
                <sfdc:field key="Department" value="Engineering" />
                <sfdc:field key="Title" value="Mule" />
                <sfdc:field key="Phone" value="123-456-7899" />
            </sfdc:sObject>
        </sfdc:create>

        <expression-transformer>
            <return-argument evaluator="string" expression="Successfully created a contact!" />
        </expression-transformer>
    </flow>
</mule>

```

## Troubleshooting

Why am I getting a `java.lang.NoClassDefFoundError: com.sun.org.apache.xerces.internal.dom.ElementNSImpl` error when I run the SalesForce connector in Eclipse?

**Answer:** To address this issue you must endorse the JDK with a proper JAXP (Java API for XML Processing) implementation. To do this, download Apache Xerces and Xalan and drop the JARs into your JVM's jre/lib/endorsed directory. If that directory does not yet exist, create it.

## Schema Documentation

The following is the full schema documentation for the SalesForce connector:

- **Module (schemadoc:page-title not set)**

cache: Unexpected program error: `java.lang.NullPointerException`

## Module (schemadoc:page-title not set)

### Config

The config element allows you to set the username, password and security token for the connection to Salesforce.

#### Attributes of <config...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
username	string	yes		Login username.
password	string	yes		Login password associated with the specified username.
securityToken	string	yes		Login security token generated by Salesforce.

#### Child Elements of <config...>

Name	Cardinality	Description
------	-------------	-------------

#### Create

Adds one or more new records to your organization's data.

#### Attributes of <create...>

Name	Type	Required	Default	Description
type	string	yes		The type of sObject to be created.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <create...>

Name	Cardinality	Description
sObject	1..*	The sObjects to be created in SalesForce.

#### Create sobjects

#### Attributes of <create-sobjects...>

Name	Type	Required	Default	Description
sObjects	string	yes		This is a list of SalesForce specific sObjects to be created.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <create-sobjects...>

Name	Cardinality	Description
------	-------------	-------------

#### Convert lead

Converts a Lead into an Account, Contact, or (optionally) an Opportunity.

#### Attributes of <convert-lead...>

Name	Type	Required	Default	Description
leadId	string	yes		ID of the Lead to convert. Required. For information on IDs, see ID Field Type.
contactId	string	yes		ID of the Contact into which the lead will be merged (this contact must be associated with the specified accountId, and an accountId must be specified). Required only when updating an existing contact. IMPORTANT if you are converting a lead into a person account, do not specify the contactId or an error will result. Specify only the accountId of the person account. If no contactID is specified, then the API creates a new contact that is implicitly associated with the Account. To create a new contact, the client application must be logged in with sufficient access rights. To merge a lead into an existing contact, the client application must be logged in with read/write access to the specified contact. The contact name and other existing data are not overwritten (unless overwriteLeadSource is set to true, in which case only the LeadSource field is overwritten). For information on IDs, see ID Field Type.

accountId	string	yes		ID of the Account into which the lead will be merged. Required only when updating an existing account, including person accounts. If no accountId is specified, then the API creates a new account. To create a new account, the client application must be logged in with sufficient access rights. To merge a lead into an existing account, the client application must be logged in with read/write access to the specified account. The account name and other existing data are not overwritten. For information on IDs, see ID Field Type.
overWriteLeadSource	string	yes		Specifies whether to overwrite the LeadSource field on the target Contact object with the contents of the LeadSource field in the source Lead object (true), or not (false, the default). To set this field to true, the client application must specify a contactId for the target contact.
doNotCreateOpportunity	string	yes		Specifies whether to create an Opportunity during lead conversion (false, the default) or not (true). Set this flag to true only if you do not want to create an opportunity from the lead. An opportunity is created by default.
opportunityName	string	yes		Name of the opportunity to create. If no name is specified, then this value defaults to the company name of the lead. The maximum length of this field is 80 characters. If doNotCreateOpportunity argument is true, then no Opportunity is created and this field must be left blank; otherwise, an error is returned.
convertedStatus	string	yes		Valid LeadStatus value for a converted lead. Required. To obtain the list of possible values, the client application queries the LeadStatus object, as in: Select Id, MasterLabel from LeadStatus where IsConverted=true
sendEmailToOwner	string	yes		ID of the Lead to convert. Required. For information on IDs, see ID Field Type.
config-ref	string	no		Specifies whether to send a notification email to the owner specified in the ownerId (true) or not (false, the default).

#### Child Elements of <convert-lead...>

Name	Cardinality	Description
------	-------------	-------------

#### Delete

Use delete() to delete one or more existing records, such as individual accounts or contacts, in your organization's data. The delete() call is analogous to the DELETE statement in SQL.

#### Attributes of <delete...>

Name	Type	Required	Default	Description
ids	string	yes		A list of ids for objects that are to be deleted.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <delete...>

Name	Cardinality	Description
------	-------------	-------------

#### Empty recycle bin

The recycle bin lets you view and restore recently deleted records for 30 days before they are permanently deleted. Your organization can have up to 5000 records per license in the Recycle Bin at any one time. For example, if your organization has five user licenses, 25,000 records can be stored in the Recycle Bin. If your organization reaches its Recycle Bin limit, Salesforce.com automatically removes the oldest records, as long as they have been in the recycle bin for at least two hours.

#### Attributes of <empty-recycle-bin...>

Name	Type	Required	Default	Description
ids	string	yes		Array of one or more IDs associated with the records to delete from the recycle bin. Maximum number of records is 200.
config-ref	string	no		The name of the config element for Salesforce.

## Child Elements of <empty-recycle-bin...>

Name	Cardinality	Description
------	-------------	-------------

### Get deleted range

Retrieves the list of individual records that have been deleted within the given timespan for the specified object.

#### Attributes of <get-deleted-range...>

Name	Type	Required	Default	Description
type	string	yes		Object type. The specified value must be a valid object for your organization.
startTime	string	yes		Starting date/time (Coordinated Universal Time (UTC)—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:30:15 is interpreted as 12:30:00 UTC).
endTime	string	yes		Ending date/time (Coordinated Universal Time (UTC)—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:35:15 is interpreted as 12:35:00 UTC).
config-ref	string	no		The name of the config element for Salesforce.

## Child Elements of <get-deleted-range...>

Name	Cardinality	Description
------	-------------	-------------

### Get deleted

Retrieves the list of individual records that have been deleted within the given timespan for the specified object.

#### Attributes of <get-deleted...>

Name	Type	Required	Default	Description
type	string	yes		Object type. The specified value must be a valid object for your organization.
duration	string	yes		The amount of time in minutes before now for which to return records from.
config-ref	string	no		The name of the config element for Salesforce.

## Child Elements of <get-deleted...>

Name	Cardinality	Description
------	-------------	-------------

### Get updated range

Retrieves the list of individual objects that have been updated (added or changed) within the given timespan for the specified object.

#### Attributes of <get-updated-range...>

Name	Type	Required	Default	Description
type	string	yes		Object type. The specified value must be a valid object for your organization.
startTime	string	yes		Starting date/time (Coordinated Universal Time (UTC)—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:30:15 is interpreted as 12:30:00 UTC).
endTime	string	yes		Ending date/time (Coordinated Universal Time (UTC)—not local—timezone) of the timespan for which to retrieve the data. The API ignores the seconds portion of the specified dateTime value (for example, 12:35:15 is interpreted as 12:35:00 UTC).
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <get-updated-range...>

Name	Cardinality	Description
------	-------------	-------------

#### **Get updated**

Retrieves the list of individual records that have been deleted within the given timespan for the specified object.

#### Attributes of <get-updated...>

Name	Type	Required	Default	Description
type	string	yes		Object type. The specified value must be a valid object for your organization.
duration	string	yes		The amount of time in minutes before now for which to return records from.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <get-updated...>

Name	Cardinality	Description
------	-------------	-------------

#### **Query**

Executes a query against the specified object and returns data that matches the specified criteria.

#### Attributes of <query...>

Name	Type	Required	Default	Description
query	string	yes		Query string.
batchsize	string	yes		Number of records to return.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <query...>

Name	Cardinality	Description
------	-------------	-------------

#### **Query subject**

Executes a query against the specified object and returns data that matches the specified criteria.

#### Attributes of <query-subject...>

Name	Type	Required	Default	Description
query	string	yes		Query string.
batchsize	string	yes		Number of records to return.
config-ref	string	no		The name of the config element for Salesforce.

#### Child Elements of <query-subject...>

Name	Cardinality	Description
------	-------------	-------------

#### **Retrieve**

Retrieves one or more records based on the specified IDs.

#### Attributes of <retrieve...>

Name	Type	Required	Default	Description
fields	string	yes		List of one or more fields in the specified object, separated by commas. You must specify valid field names and must have read-level permissions to each specified field. The fieldList defines the ordering of fields in the result.
type	string	yes		The type of sObject to be returned.
ids	string	yes		Array of one or more IDs of the objects to retrieve. You can pass a maximum of 2000 object IDs to the retrieve() call.
config-ref	string	no		The name of the config element for Salesforce.

Child Elements of <retrieve...>

Name	Cardinality	Description
------	-------------	-------------

### RetrieveSObjects

Attributes of <retrieveSObjects...>

Name	Type	Required	Default	Description
fields	string	no		List of fields to be returned in a comma seperated list.
type	string	yes		The type of sObject to be returned.
ids	string	yes		List of the ids for the objects to return.
config-ref	string	no		The name of the config element for Salesforce.

Child Elements of <retrieveSObjects...>

Name	Cardinality	Description
------	-------------	-------------

### Update objects

Use this call to update one or more existing records, such as accounts or contacts, in your organization's data. The update() call is analogous to the UPDATE statement in SQL.

Attributes of <update-objects...>

Name	Type	Required	Default	Description
sObjects	string	yes		A list of sObjects to be updated.
config-ref	string	no		The name of the config element for Salesforce.

Child Elements of <update-objects...>

Name	Cardinality	Description
------	-------------	-------------

### Update

Use this call to update one or more existing records, such as accounts or contacts, in your organization's data. The update() call is analogous to the UPDATE statement in SQL.

Attributes of <update...>

Name	Type	Required	Default	Description
config-ref	string	no		The name of the config element for Salesforce.
type	string	yes		Object type. The specified value must be a valid object for your organization.

Child Elements of <update...>

Name	Cardinality	Description
sObject	1..*	

## Upsert

Creates new records and updates existing records; uses a custom field to determine the presence of existing records. In most cases, we recommend that you use upsert() instead of create() to avoid creating unwanted duplicate records (idempotent). Available in the API version 7.0 and later. You can process records for one or more than object type in an create() or update() call, but all records must have the same object type in an upsert() call.

### Attributes of <upsert...>

Name	Type	Required	Default	Description
config-ref	string	no		The name of the config element for Salesforce.
type	string	yes		Object type. The specified value must be a valid object for your organization.
externalIdFieldName	string	yes		Contains the name of the field on this object with the external ID field attribute for custom objects or the idLookup field property for standard objects. The idLookup field property is usually on a field that is the object's ID field or name field, but there are exceptions, so check for the presence of the property in the object you wish to upsert().

### Child Elements of <upsert...>

Name	Cardinality	Description
sObject	1..*	

## Upset sobjects

Use this call to update one or more existing records, such as accounts or contacts, in your organization's data. The update() call is analogous to the UPDATE statement in SQL.

### Attributes of <upsert-sobjects...>

Name	Type	Required	Default	Description
sObjects	string	yes		A list of sObjects to be upserted.
config-ref	string	no		The name of the config element for Salesforce.
externalIdFieldName	string	yes		Contains the name of the field on this object with the external ID field attribute for custom objects or the idLookup field property for standard objects. The idLookup field property is usually on a field that is the object's ID field or name field, but there are exceptions, so check for the presence of the property in the object you wish to upsert().

### Child Elements of <upsert-sobjects...>

Name	Cardinality	Description

## Sobject to string transformer

A transformer that gives a human-readable description of an SObject (useful for debugging).

### Child Elements of <sobject-to-string-transformer...>

Name	Cardinality	Description

## Mule sobjects to string transformer

A transformer that gives a human-readable description of a list of MuleSObject (useful for debugging).

### Child Elements of <mule-sobjects-to-string-transformer...>

Name	Cardinality	Description
------	-------------	-------------

## Twitter



Twitter Cloud Connector

<b>Name</b>	Twitter Connector
<b>API URL</b>	<a href="http://dev.twitter.com/doc">http://dev.twitter.com/doc</a>
<b>Download</b>	<a href="http://repository.ibbeans.muleforge.org/org/mule/ibbeans/twitter-ibean/1.1/twitter-ibean-1.1.jar">http://repository.ibbeans.muleforge.org/org/mule/ibbeans/twitter-ibean/1.1/twitter-ibean-1.1.jar</a>
<b>State</b>	
<b>Supported Mule Versions</b>	3.1.x - <a href="#">Download!</a>

### Table of Contents

- Intro
- Install
- Configure
- How to get the twitter oauth consumer key/secret
- How to get the twitter oauth access key/secret
- Example Usage
- Schema Documentation

#### **Intro**

The Twitter connector provides an easy way to integrate with the Twitter API using Mule flows.

#### **Install**

The connector can either be installed for all applications running within the Mule instance or can be setup to be used for a single application.

##### **To Install Connector for All Applications**

Download the connector from the link above and place the resulting jar file in /lib/user directory of the Mule installation folder.

##### **To Install for a Single Application**

To make the connector available only to single application then place it in the lib directory of the application otherwise if using Maven to compile and deploy your application the following can be done.

1. Add the MuleForge snapshot repository to your pom.xml file for your project. This can be done by adding the following under the <repositories> element:

```
<repository>
  <id>ibeans-repo</id>
  <name>iBeans Repository</name>
  <url>http://repository.ibbeans.muleforge.org</url>
  <layout>default</layout>
</repository>
```

2. Add the connector as a dependency to your project. This can be done by adding the following under the <dependencies> element in the pom.xml file of the application:

```

<dependency>
    <groupId>org.mule.ibbeans</groupId>
    <artifactId>twitter-ibean</artifactId>
    <version>1.1</version>
</dependency>

```



Are you using the Mule Maven Plugin to build your application? If so please make sure to list it in the inclusions section of the plug-in, for example:

```

<plugin>
    <groupId>org.mule.tools</groupId>
    <artifactId>maven-mule-plugin</artifactId>
    <version>1.5</version>
    <extensions>true</extensions>
    <configuration>
        <inclusions>
            <inclusion>
                <groupId>org.mule.ibbeans</groupId>
                <artifactId>twitter-ibean</artifactId>
            </inclusion>
        </inclusions>
    </configuration>
</plugin>

```

## Configure

To use the Twitter connector within a flow the namespace for the connector must be included. The resulting configuration file header will look similar to the following:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/twitter
          http://www.mulesoft.org/schema/mule/twitter/3.1/mule-twitter.xsd">

```

Once the namespace has been included then the Twitter connector can then be configured by using the config element of the connector. The following is an example of a config element for the Twitter connector:

```

<twitter:config name="twitter" format="JSON" consumerKey="${twitter.consumer.key}" consumerSecret=
"${twitter.consumer.secret}"/>

```

Within the config element a name is given along with any required or optional parameters. The Twitter config element has attribute for the format to be used (JSON,XML,ATOM) as well as for the oauth consumer key and secret. The oauth access key/secret can either be configured globally in the config element or on an operation element directly.

Property placeholders can be used to externalize configuration values that can then be set using a properties file or system properties.

**Note:** Different twitter operations support direction formats, not all formats are supported with all operations. Also some operations require authentication whereas others don't.

### How to get the twitter oauth consumer key/secret

#### 1) Create a Twitter account

If you don't already have a twitter account you'll need to create one [here](#)

## 2) Register a new Twitter application

You can register a new application with Twitter [here](#). The required Oauth keys are then available from the registered application details.

### How to get the twitter oauth access key/secret

These keys are user specific, such that each twitter user has their own key/secret pair. If everything you will be doing with twitter will be done on behalf of a single twitter user then the oauth access key and secret of this user can be configured globally on the config element. You can find the values for these by following the link "My Access Token" from the page where you found the the OAuth consumer key/secret.

If twitter is going to be used with different users then the OAuth access key/secret needs to be provided and configured on the operation elements. Typically the web front-end of your application will perform the OAuth authentication with twitter and have the user sign into twitter and can provide these values as part of the incoming message.

### Example Usage

The following are examples of the usage of the Twitter connector. For a full reference please check out the schema reference. For more information of different integration scenarios you can implement using cloud connectors see [Integrating with Cloud Connect](#).

#### Search

To perform a twitter timeline search and return results when a Mule http endpoint is invoked you would do the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/twitter
          http://www.mulesoft.org/schema/mule/twitter/3.1/mule-twitter.xsd">

    <twitter:config name="twitter"/>

    <flow name="queryFlow">
        <http:inbound-endpoint host="myHost" port="80"/>
        <twitter:public-timeline />
    </flow>
</mule>
```

#### Status Update

To update your twitter status you would do the following:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:twitter="http://www.mulesoft.org/schema/mule/twitter"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/twitter
          http://www.mulesoft.org/schema/mule/twitter/3.1/mule-twitter.xsd">

    <twitter:config name="twitter" format="JSON" consumerKey="${twitter.consumer.key}"
                     consumerSecret="${twitter.consumer.secret}" oauthToken="${twitter.access.token}"
                     oauthTokenSecret="${twitter.access.secret}" />

    <flow name="updateStatusFlow">
        <http:inbound-endpoint host="myHost" port="80"/>
        <twitter:update-status status=#*[header:INBOUND:status] />
    </flow>
</mule>

```

## **Schema Documentation**

The following is the full schema documentation for the Twitter connector:

- **Module (schemadoc:page-title not set)**

cache: Unexpected program error: java.lang.NullPointerException

### **Module (schemadoc:page-title not set)**

#### **Config**

##### **Public timeline**

Returns the 20 most recent statuses from non-protected users who have set a custom user icon. The public timeline is cached for 60 seconds so requesting it more often than that is a waste of resources.

##### **Attributes of <public-timeline...>**

Name	Type	Required	Default	Description
config-ref	string	no		

##### **Child Elements of <public-timeline...>**

Name	Cardinality	Description

##### **Friends timeline**

Returns the most recent statuses posted by the authenticating user and that user's friends. This is the equivalent of /timeline/home on the Web.

##### **Attributes of <friends-timeline...>**

Name	Type	Required	Default	Description
config-ref	string	no		

consumerKey	string	no		Specifies oauth consumer key. Alternatively individual operation elements can provide authentication configuration.
consumerSecret	string	no		Specifies oauth consumer secret. Alternatively individual operation elements can provide authentication configuration.
count		no		Specifies the number of statuses to retrieve. May not be greater than 200.
sincId	positiveInteger	no		Returns only statuses with an ID greater than (that is, more recent than) the specified ID.
maxId	positiveInteger	no		Returns only statuses with an ID less than (that is, older than) or equal to the specified ID
page	positiveInteger	no		Specifies the page of results to retrieve.

#### Child Elements of <friends-timeline...>

Name	Cardinality	Description
------	-------------	-------------

#### User timeline

Returns the 20 most recent statuses posted from the authenticating user. It's also possible to request another user's timeline via the id parameter. This is the equivalent of the Web user page for your own user, or the profile page for a third party.

#### Mentions

Returns the most recent mentions (status containing @username) for the authenticating user.

#### Attributes of <mentions...>

Name	Type	Required	Default	Description
config-ref	string	no		
consumerKey	string	no		Specifies oauth consumer key. Alternatively individual operation elements can provide authentication configuration.
consumerSecret	string	no		Specifies oauth consumer secret. Alternatively individual operation elements can provide authentication configuration.
count		no		Specifies the number of statuses to retrieve. May not be greater than 200.
sincId	positiveInteger	no		Returns only statuses with an ID greater than (that is, more recent than) the specified ID.
maxId	positiveInteger	no		Returns only statuses with an ID less than (that is, older than) or equal to the specified ID
page	positiveInteger	no		Specifies the page of results to retrieve.

#### Child Elements of <mentions...>

Name	Cardinality	Description
------	-------------	-------------

#### Search

Returns tweets that match a specified query.

#### User

Returns extended information of a given user, specified by ID or screen name as per the required id parameter. The author's most recent status will be returned inline.

#### Update status

Updates the authenticating user's status.

## Show status

Returns a single status, specified by the id parameter

# Getting Started with Cloud Connect

## Getting Started with Cloud Connect

In much the same way that Mule makes it easy to use transports with lower level resources such as JMS queues or file directories, Mule Cloud Connectors radically simplify the act of integrating with cloud services.

## Configuration

Cloud connectors are used in a similar way to transports with service-specific elements being used wherever you need to integrate with a particular service. Cloud connectors have a '<config>' element used for configuring the connector e.g. security tokens and an element for each available service operation. The service operation elements are specific. The following is an example (with attributes excluded).

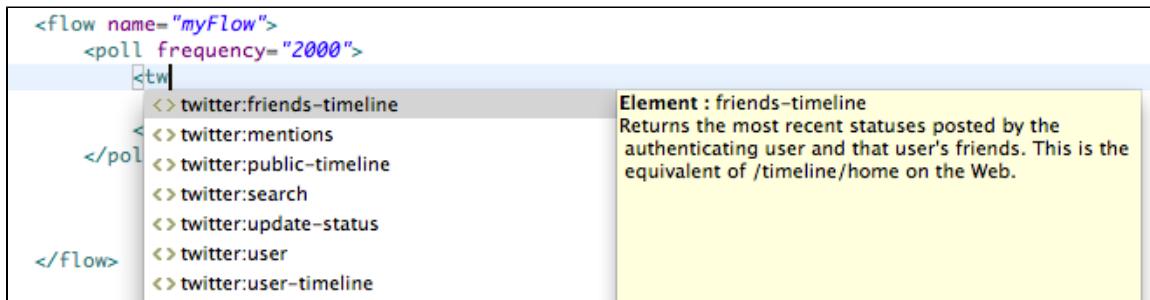
### Twitter Cloud Connect Configuration Fragment

```
<twitter:config ... /> (configured globally, optional)  
  
<twitter:public-timeline ... /> (configured in flows)  
<twitter:user-timeline ... /> (configured in flows)
```

Using cloud connectors in this way has all the same advantages of "transport-specific endpoints" available since Mule 2.0. More specifically:

- Auto completion in your favorite IDE or xml editor
- Context-sensitive documentation
- Documentation of defaults and valid values.

And here's a screenshot from the configuration editor in MuleIDE:



## Getting Started

To begin using Cloud Connectors, you'll take three simple steps:

1. Download the Cloud Connector you wish to use (you'll find a download link at the top of the connector documentation page)
2. Add the connector's namespace to your configuration file.
3. Use connector operation elements in your flows

These steps differ slightly depending on your development environment, here we'll assume you are either using MuleIDE or are working with a maven based project.

### MuleIDE

1. Drop the downloaded cloud connector jar file into the */lib/user* directory of your Mule distribution.
2. Create a new Mule project if you don't already have one created
3. Create a new Mule configuration file and select the mule transport, modules and cloud connectors you wish to use in the wizard, this will automatically add the correct namespaces to your configuration file
4. Use the XML editor in MuleIDE to build your flows using Cloud Connectors
5. Add any other Mule functionality you need

## **Maven**

1. Add dependency entries for the cloud connectors you will be using to your pom.xml. (Each cloud connectors documentation page provides you with maven dependency XML snippets that you can simply copy and paste.)
2. For each cloud connectors you will be using add the namespace and schema location declarations to your Mule configuration files. Each cloud connectors documentation will give you a namespace snippets that you can simply copy and paste.
3. Use an XML editor to build your flows using Cloud Connectors
4. Add any other Mule functionality you need

(You can copy and paste snippets for maven dependency and namespace declarations from the cloud connector documentation pages straight into your config file.)

## **What's Next?**

Now you know how to download cloud connectors and get started with them you are probably interested in learning more about what you can do with them. [Integrating with Cloud Connect](#) will show you how to use cloud connectors in your Mule flows to implement a variety of different cloud connector integration scenarios.

## **Need Help?**

If you run into any issues or just have general questions about cloud connectors, visit the MuleSoft Forums. Submit Cloud Connector questions to [Cloud Connector Forum](#) and Mule specific questions to [Mule User Forum](#)

Your Rating: 

Results:  1 rates

## **How to Build a Cloud Connector**

### **How to Build a Cloud Connector**

The web has introduced thousands of new applications for business, IT, collaboration, social media, e-commerce and productivity and many of them now expose web APIs to integrate and automate the use of these applications and the data they manage.

Mule ESB supports a growing number of [Cloud Connectors](#) but you or your customers might require connectivity to a application API which is not currently listed. To help with this, Mule has created a development kit which guides the developer though the setup and creation of a Cloud Connector.

In most cases, developing a Cloud Connector requires no knowledge of Mule ESB and the development kit will provide most of the libraries and infrastructure you need to build your connector.

**Mule Cloud Connect  
Development Kit**

**DOWNLOAD NOW!**

4.7Mb - Version: 2.0.16

## **Getting Help**

Each API can be a little different, so this documentation provides a number of suggested best practices and tips for how to build connectors and handle these differences. You can also request help or ask questions on the [Cloud Connectors forum](#). If you have any suggestions for improvements or additions to the development kit, let us know by filing a feature request or bug report [here](#).

## **Where should I start?**

Look at our [Creating a Cloud Connector](#) section to get started. Make sure that you have downloaded our development kit first.

## **Need Help?**

If you need help while developing your Cloud Connector you can always visit the [Cloud Connector Forum](#) and post your question to the community.

## **Found a bug?**

If you have found a bug in one of the cloud connectors or the cloud connector development kit itself, please report it with our [bug tracker](#).

Your Rating:  0 rates

Results:  0 rates

## Creating a Cloud Connector

### Creating a Cloud Connector

So, you want to create a cloud connector right? You have come to the right place. First take a look at our Prerequisites section below to make sure that you have everything to get started and then you can jump into our [Using the Archetype](#) guide which will explain how to user our Maven archetype, or you might use our [pre-packaged distribution](#) to generate a skeleton project.

Alternatively if you have an existing project and you want to convert it to a cloud connector see our section [Convert your Existing Project](#).

### Prerequisites

In order to get started building a cloud connector, you need to check that you have a few items first:

The development kit uses Java and Apaches Maven to compile and package your cloud connector. It also will handle installing any additional libraries that your connector will require. You do not need to be an experienced user of Maven to use the development kit as we have wrapped most of the commands in simple scripts and document them here.

The best way to check to see if you have Maven installed and set up correctly is to run the following command from your console:

```
mvn -version
```

If you have version 2.0.9 or later of Maven installed (Maven 3.x also works fine) you are set. Otherwise please download and install the latest version of Maven. More information and download links for Maven can be found at the [Apache Maven site](#).

As mentioned, Java is also required and you should make sure you have a recent (Java 1.6 or later) software development kit installed.

Your Rating: 

Results:  0 rates

## Convert your Existing Project

### Convert your Existing Project

If you have an existing project, then converting it to be a cloud connector is as easy as counting to three. The first thing that you need to make sure of is that the project is using [Maven](#) since our development kit was developed as a Maven plugin.

If your project is not Maven-based then you should first Mavenize your project and then come back here.

#### 1. Change the packaging

Our Maven development kit uses a custom lifecycle and therefore it requires for you to change the packaging from jar to cloud-connector.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.mule.module</groupId>
    <artifactId>mule-module-flickr</artifactId>
    <packaging>cloud-connector</packaging> <!-- CHANGE YOUR PACKAGING FROM JAR TO CLOUD-CONNECTOR -->
    <version>1.0-SNAPSHOT</version>
    <name>Flickr Cloud Connector</name>

    ...

```

Also, make sure that you add our maven plugin as extension to your build:

```

<build>
<extensions>
    <extension>
        <groupId>org.mule.tools</groupId>
        <artifactId>mule-cloud-connector-maven-plugin</artifactId>
        <version>2.0.4</version>
    </extension>
</extensions>
</build>

```

## 2. Annotate your Connector

Annotate your connector class. See our section [Connector Annotations](#) to see all the available annotations and their usage.

## 3. Done

That wasn't so hard, was it? That is all you have to do. Our dev kit will automatically generate a namespace handler and an xml schema and will package everything together along with your own classes.

### What's next?

You may want to make sure that the generated namespace handler works as expected with your connector. See our [Testing Your Connector](#) section for more information on how to accomplish this.

Your Rating:  Results:  0 rates

## Using the Archetype

### *Using the Archetype*

We offer a Maven Archetype as part of our development kit that will greatly improve the time you spent setting up new projects.

The archetype is host at our own public repositories and therefore it requires for you to add them to your Maven's settings.xml before you can actually execute it:

```

<mirrors>
    <mirror>
        <id>muleforge-release</id>
        <mirrorOf>muleforge-release</mirrorOf>
        <name>Muleforge Release Repository</name>
        <url>http://repository.muleforge.org/release/</url>
    </mirror>
    <mirror>
        <id>muleforge-snapshot</id>
        <mirrorOf>muleforge-snapshot</mirrorOf>
        <name>Muleforge Snapshot Repository</name>
        <url>http://repository.muleforge.org/snapshot/</url>
    </mirror>
</mirrors>

```

Our archetype receives several parameters, let's look at a sample command line before we dive into each parameter:

```

mvn archetype:generate -B -DarchetypeRepository=muleforge-release -DarchetypeGroupId=org.mule.tools
-DarchetypeArtifactId=mule-cloud-connector-archetype -DarchetypeVersion=2.0.7
-DgroupId=<YOUR GROUP ID> -DartifactId=<YOUR ARTIFACT ID> -Dversion=1.0-SNAPSHOT
-Dpackage=<YOUR PACKAGE> -DmuleVersion=3.1 -DcloudService=<CLOUD SERVICE ID>
-DcloudServiceType=<CLOUD SERVICE TYPE>

```

As you can see there are several parameters, so here is the breakdown:

Parameter	Description
-----------	-------------

groupId	The group identifier of the project you are about to create
artifactId	The artifact identifier of the project you are about to create
package	The package under which the skeleton cloud connector will live
muleVersion	The version of Mule ESB you are targeting. Usually 3.1.
cloudService	The name of the service your cloud connector will connect to. This is usually things like: PayPal, eBay, Facebook. This string will be used as a base name for your cloud connector class and also for the prefix of the namespace.
cloudServiceType	The kind of connectivity that the service employs. See our <a href="#">Connector Types</a> for the right answer.

After this step you will have a complete Maven project setup for you. If you prefer to work in Eclipse please see the section [Using Eclipse](#) or if you prefer IntelliJ see our section [Using IntelliJ](#) which covers the steps needed to import the project into IDE of choice.

#### What's next?

Now get started with the development of your connector. See [Connector Annotations](#) for a more detailed description on how to annotate your connector.

Your Rating:  Results:  0 rates

## Connector Types

### Connector Types

After following through the [Getting Started with Cloud Connect](#) a developer must choose which type of connector they are building. This section presents the three different options for connectors:

Type	Description
HTTP based	Many services today provide HTTP based services which might be RESTful in nature or could be more RPC. To work with these types of services in Cloud Connect it is recommended to use the <a href="#">Jersey Client</a> which is included as a dependency to your project when you choose to create a HTTP based connector. A good example of this type of connector is the <a href="#">Authorize.Net</a> connector which makes use of Jersey for calling out to the service.
Java API based	Choose this type of connector when you already have existing code or a library you want to use to build the cloud connector. When the <code>create-cloud-connector</code> script asks you for the type of the project select a Java based project. You should copy your existing code into the generated project. Alternatively, you can follow our guide on how to convert an existing project into a cloud connector.
SOAP based	Cloud connectors can be written using an existing SOAP web service with a defined WSDL. Choose this approach if you want to build a simpler API to an existing web service.

Your Rating:  Results:  0 rates

## Using the Distribution

### Using the Distribution

We packaged a custom distribution which will simplify the creation of the cloud connector. The distributions are made available thru our Maven repository at this <http://repository.muleforge.org/release/org/mule/distributions/mule-cloud-connector-dev-kit/> address.

You will need to download and unpack the distribution. You will find inside of it, two scripts called **create-cloud-connector**. One of the scripts is written in Bash (which is suitable for Unix-variants like MacOS X and Linux) and the other one is written in Windows' shell script (ending with extension .BAT).

Once you executed the script you will be asked a series of questions that will shape your connector, let's go question by question.

```
# ./create-cloud-connector

Name of the cloud service the new connector uses
[default: ]
> myconnector
```

The first question is the name of the cloud service that the connector you are building will consume. This name will be used to shape the

namespace of your connector, the name of the POJO that we will generate, among other things.



Example of good names are the following: Facebook, Flickr, EBay, LinkedIn, SalesForce, Twitter, etc.

```
Maven group id of the new connector  
[default: org.mule.modules]  
>
```

The next question is the groupId of the Maven artifact. The default is **org.mule.modules** but you can pick which ever groupId you want.

```
Maven artifact id of the new connector  
(should use naming convention mule-module-<yourname>)  
[default: mule-module-myconnector]  
>
```

Now, its time to tell what our Maven artifact should be called. The standard is to call it "mule-module-" and the name of the cloud service. Again, this is the standard but any artifactId will be fine.

```
Maven version of the new connector  
[default: 1.0-SNAPSHOT]  
>
```

This one is pretty much self-explanatory, it will set the version of the Maven artifact. Defaults to **1.0-SNAPSHOT**.



Keep in mind that this version will also set the version of your connector schema, unless otherwise specified.

```
Java package for the new connector  
(should use the naming convention org.mule.module.<yourname>)  
[default: org.mule.module.myconnector]  
>
```

This variable will dictate under which Java package your connector's code will live. It defaults to **org.mule.module**. and the name of your connector. Your skeleton connector POJO will be generated under this package.

```
Type of project  
Can be either be  
* based on existing Java code  
* a HTTP based service  
* a WSDL  
Options: [j], [h] or [w]  
[default: ]  
> h
```

The last question and the most important one. It will tell our script what kind of connector you are trying to build. For a detailed description of each visit our [Connector Types](#) page.

Since that was our last question, the script will now proceed to generate the code for you. If everything goes well, you should see something like this as its output.

```

[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] >>> maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.0:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] Archetype repository missing. Using the one from
[org.mule.tools:mule-cloud-connector-archetype:1.0] found in catalog remote
Downloading: http:
//repol.maven.org/maven2/org/mule/tools/mule-cloud-connector-archetype/1.0/mule-cloud-connector-archetype
Downloaded: http:
//repol.maven.org/maven2/org/mule/tools/mule-cloud-connector-archetype/1.0/mule-cloud-connector-archetype
(17 KB at 23.5 KB/sec)
Downloading: http:
//repol.maven.org/maven2/org/mule/tools/mule-cloud-connector-archetype/1.0/mule-cloud-connector-archetype
Downloaded: http:
//repol.maven.org/maven2/org/mule/tools/mule-cloud-connector-archetype/1.0/mule-cloud-connector-archetype
(3 KB at 4.3 KB/sec)
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.822s
[INFO] Finished at: Mon Mar 14 12:45:13 GMT-03:00 2011
[INFO] Final Memory: 9M/81M
[INFO] -----

```

Now you should have a complete Maven project setup for you. If you prefer to work in Eclipse please see the section [Using Eclipse](#) or if your prefer IntelliJ see our section [Using IntelliJ](#) which covers the steps needed to import the project into IDE of choice.

#### **What's next?**

Now get started with the development of your connector. See [Connector Annotations](#) for a more detailed description on how to annotate your connector.

Your Rating:  Results:  0 rates

## **Connector Annotations**

### **Connector Annotations**

This section will describe what a connector looks like and how you can use annotations to describe your connection.

A cloud connector is basically a plain old java object with custom annotations that help the generators construct the namespace handler and the xml schema.

```

package org.mule.modules.myconnector;

import org.mule.tools.cloudconnect.annotations.Operation;
import org.mule.tools.cloudconnect.annotations.Connector;
import org.mule.tools.cloudconnect.annotations.Parameter;

@Connector(namespacePrefix = "movie", namespaceUri = "http://www.mulesoft.org/schema/mule/movie")
public class MovieConnector
{
    /**
     * The response format used by the operations of this connector, can be "XML" or "JSON"
     */
    @Property
    private String format;

    /**
     * Search my movies using the specified free text
     * @param text The text to search
     * @param year Year of the movie
     */
    @Operation
    public String search(String text, @Parameter(optional=true) int year)
    {
    }

    /* The format property setters and getters have been omitted for brevity */
}

```

## @Connector

The `@Connector` annotation should be used on each class that will get exported to be accessible as a cloud connector. You can have more than one but each will have a different namespace handler and schema.

Parameter	Description
namespacePrefix	The XML namespace prefix
namesapceUri	The URI under which the schema will be published
factory	<b>Optional.</b> This parameter will allow you to specify a custom factory bean for your connector, use only under cases where you need to do custom creation and or initialization of your connector. The class should implement <a href="#">FactoryBean</a> .

## @Operation

The `@Operation` annotation should be placed on all methods that are to be exported from your connector. There are a couple of rules on what methods qualify for being exportable, see the following list:

- It must be public.
- It must **NOT** be static.
- Every parameter must be of a type supported by our type mapper.

Parameter	Description
name	<b>Optional.</b> You may specify an alternate name for your operation. <b>This name must be in camel case.</b>

Usage Example:

```

@Operation(name="advancedSearch")
public String search(String text);

```

## @Property

The `@Property` annotation is used to signal the generator for all the bean properties that will be configurable for each instance of your connector.

Parameter	Description
name	<b>Optional.</b> You may specify an alternate name for your property. <b>This name must be in camel case.</b>

Usage Example:

```
@Property(name="apiKey")
public String key;
```

### @Parameter

The @Parameter annotation is used to describe special scenarios for operation arguments. It allows you for example to set an argument as optional or even specify a default value.

Parameter	Description
optional	This is self-explanatory. If the argument is a primitive type you also need to specify a defaultValue, otherwise it will assign null to the parameter if it is not specified.
defaultValue	The default value of the parameter if its not specified.

### What's Next

As you develop your connector make sure to document all your methods using standard Javadoc annotations as they will be brought into the generated schema. More information about how to best document your code can be found on [Documenting Your Connector](#).

Another important aspect is testing of your connector. More information about how to test your connector can be found on [Testing Your Connector](#).

Finally once you have developed your full connector you will want to make it available to the full community. More information can be found at [Submitting a Cloud Connector](#).

Your Rating:  Results:  0 rates

## Using Eclipse

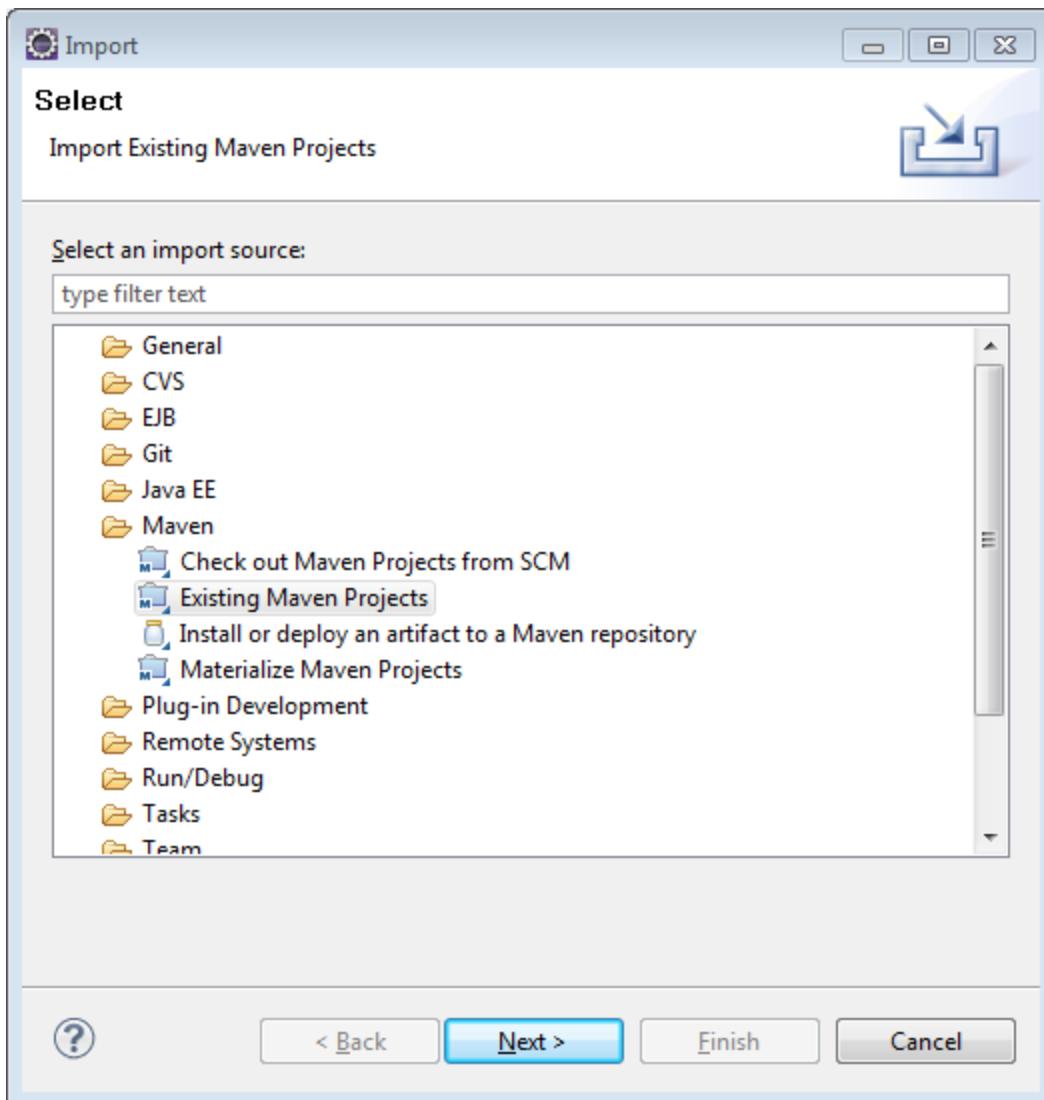
The first step after creating your project on the command line is to import it into your favourite IDE. We show the steps for Eclipse here.

### Prerequisites

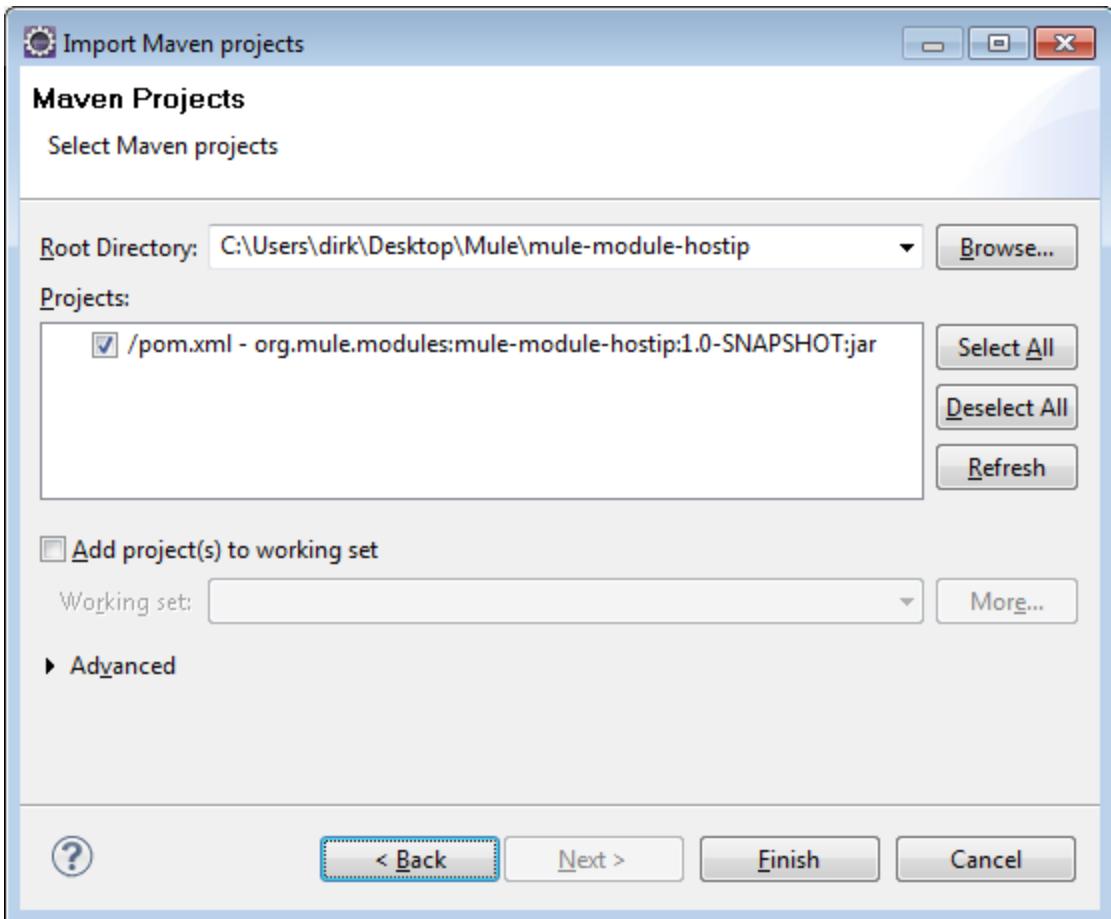
Your new cloud connector is generated as a [Maven](#) project. So you must have the Maven tooling for Eclipse installed first. Follow the [m2eclipse installation guide](#) to install the Maven tooling.

### Importing your project

1. Start Eclipse and set up a workspace for your cloud connector if you haven't done that already. (Make sure your workspace does not have a space in the directory path)
2. In the workbench view, choose **File > Import ...**
3. On the next dialog, select **Existing Maven Projects** from the **Maven** section.

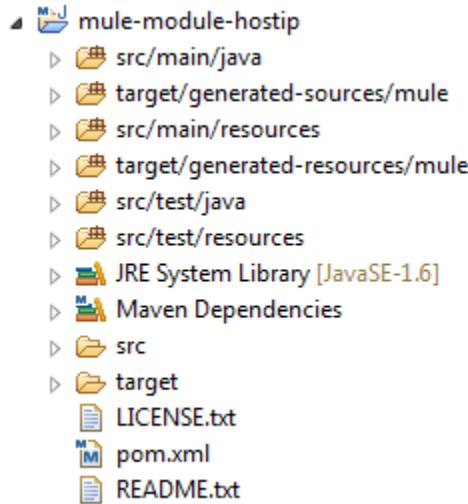


4. On the next dialog navigate to the new cloud connector project.



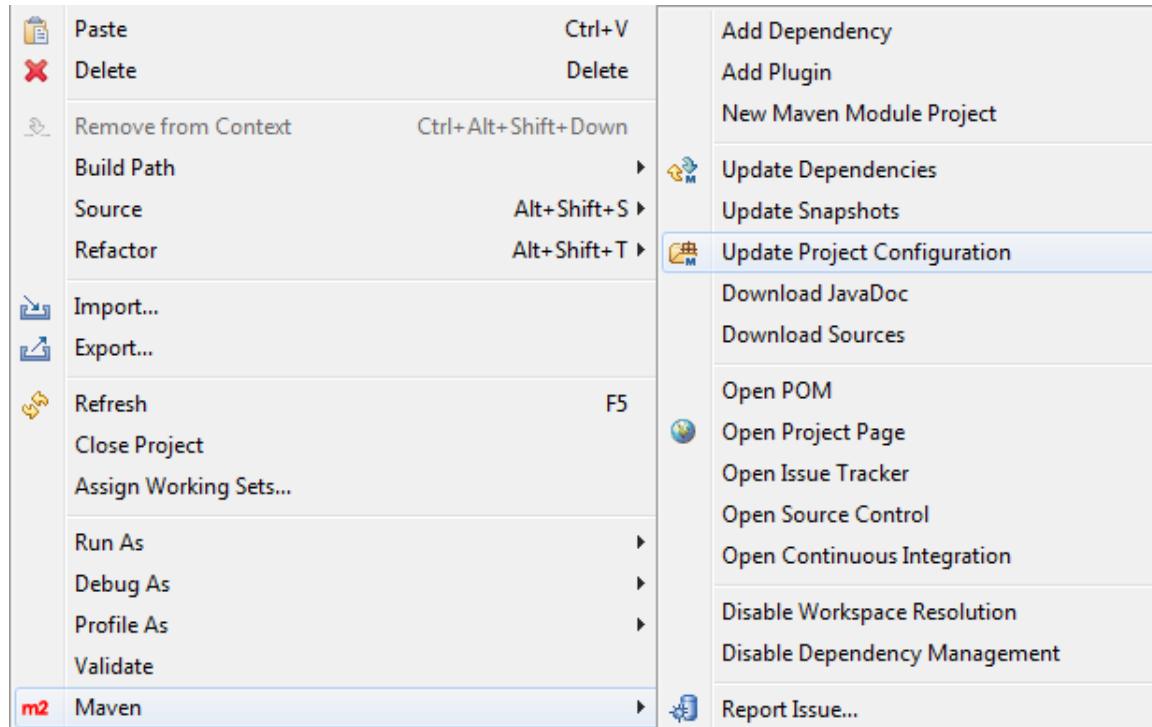
5. Click **Finish** to import the project into your workspace.

Building your project for the first time may take a long time since the Eclipse Maven integration will download all the required libraries from the internet. By the time the Maven tooling for Eclipse has finished building your project it should look like this:



#### Updating your project

The Mule schema and namespace handler for your cloud connector are automatically generated by the Maven build. The Maven tooling for Eclipse, however, does not perform the full build every time you save the project. As a consequence you have to manually update your project every time you add or remove methods to your cloud connector class. Right-click on the cloud connector project and from the **Maven** submenu select **Update Project Configuration**.



Your Rating: Results: 0 rates

## Using IntelliJ

### Using IntelliJ

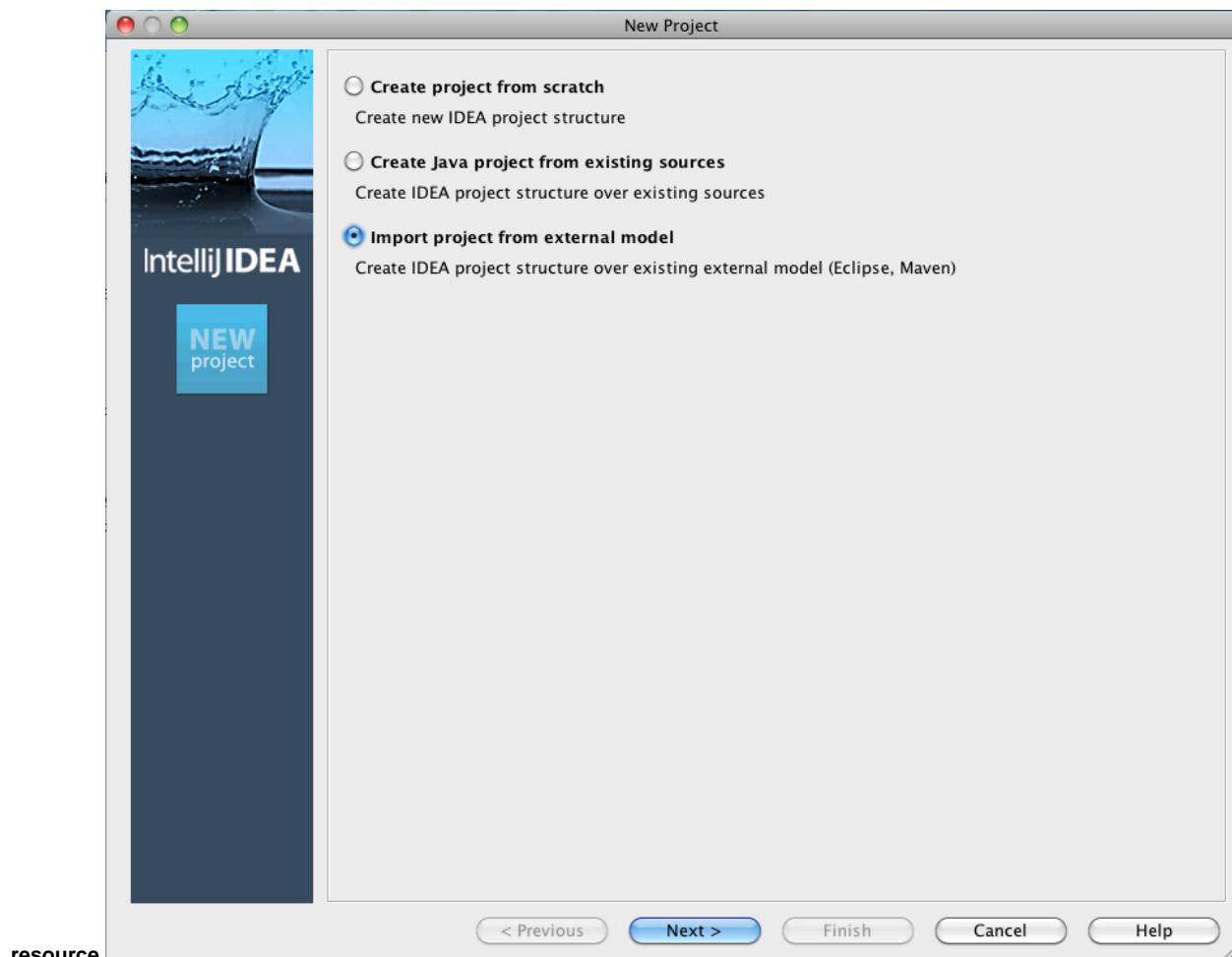
IntelliJ includes out of the box Maven support therefore it doesn't require for you to download any extra plugins.



This guide was built using IntelliJ X but earlier or later versions should work as well

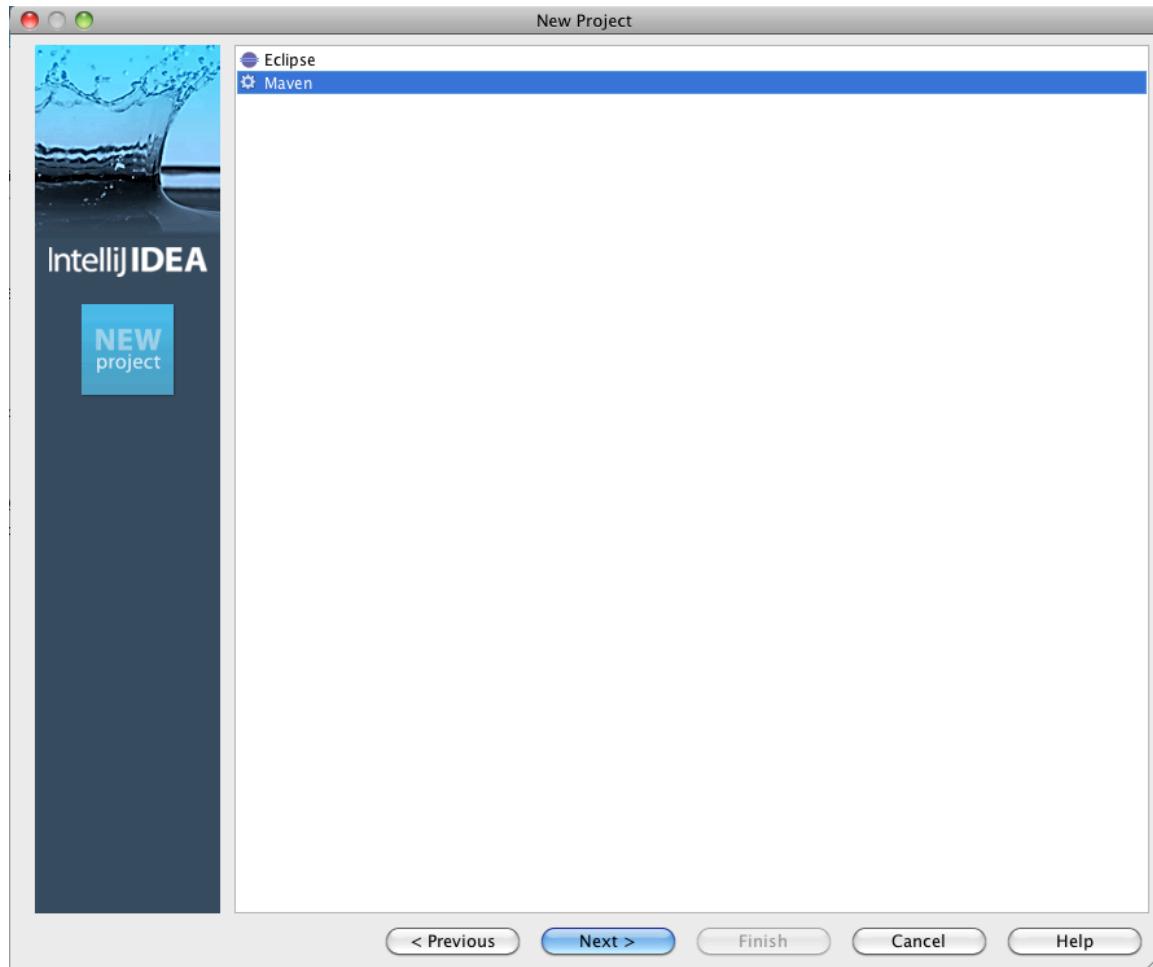
### Importing the project

After you have generated your project with our Maven archetype, choose **File -> New Project** and select **Import project from an external**

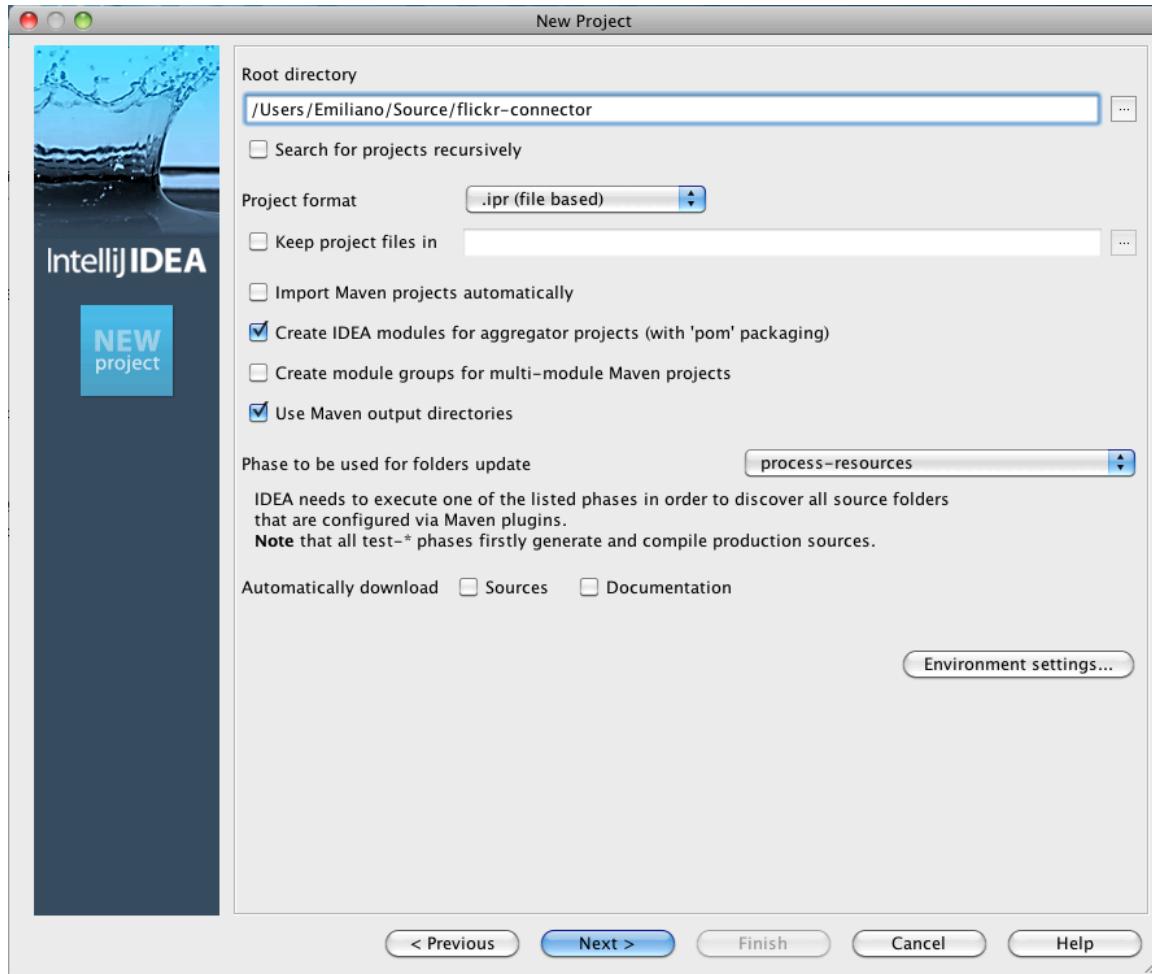


resource

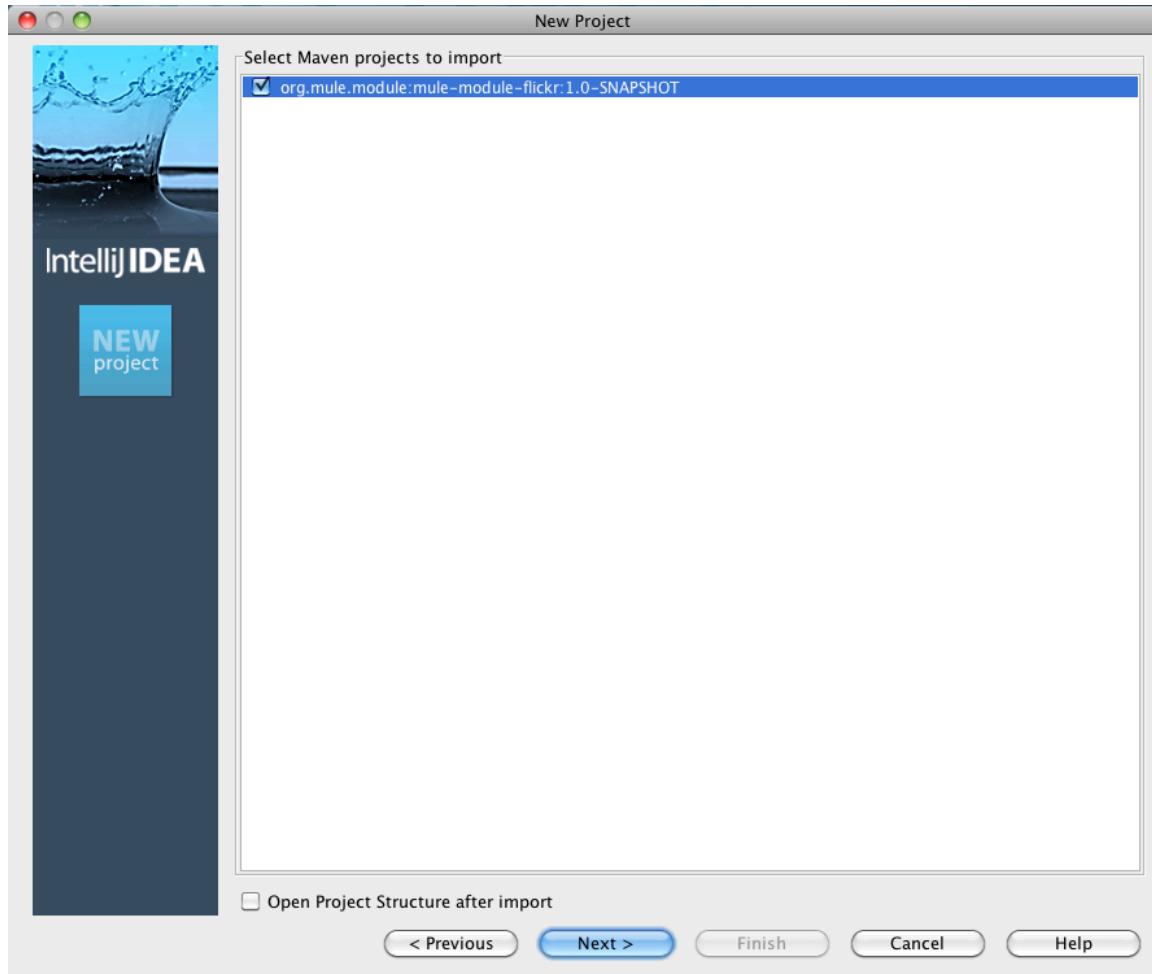
Click Next and choose Maven



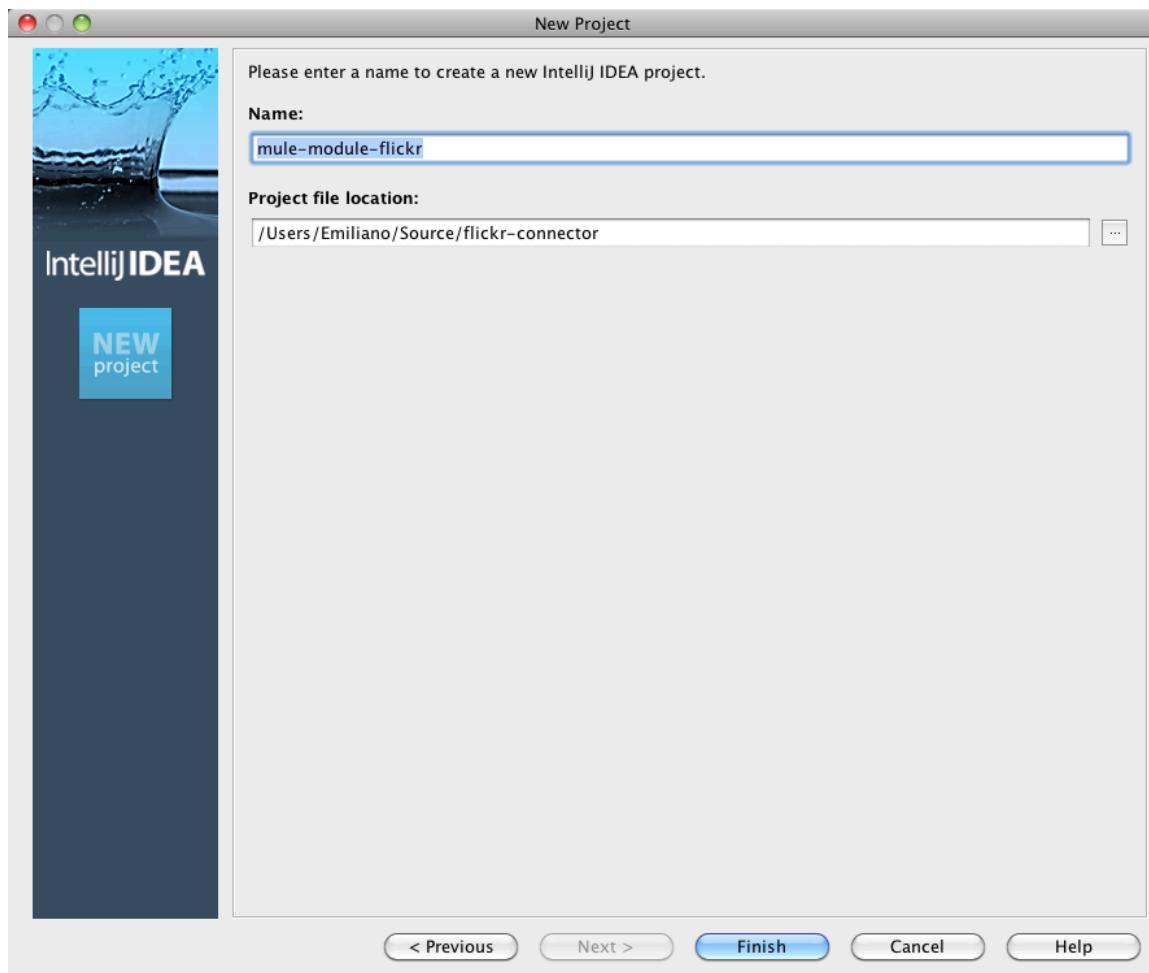
Click Next and input the source folder where your project was generated. Leave everything else as default.



Click Next and pick your main artifact



Click Next



Now, click Finish.

Your IDE should start importing your Maven dependencies.

### **Resolving the schema**

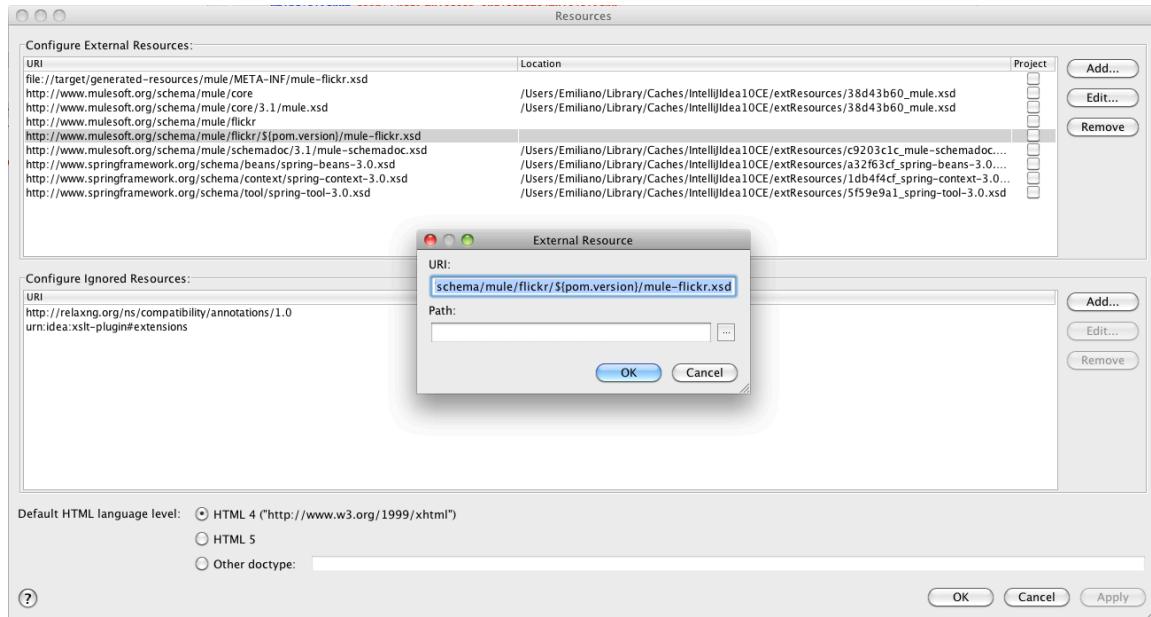
Now, you need to instruct your IDE to help it find you newly generated schema so you get all the benefits from auto complete and validation.

Open the namespace handler xml that was generated for you by the archetype.

You should see something along the lines of this:



Select **Manually Setup External Resource** and pick the schema under target/generated-resources/mule.



That should be it. Now you're \*done\*.

Your Rating: Results: 0 rates

## Testing Your Connector

### Testing Your Connector

An important part of developing your cloud connector is to write unit tests. Unit tests fall into several different categories, all covered below.

#### Testing the validation of input data

As your cloud service accepts data that will be sent to the cloud service it may implement consistency checks for its input. Some parameters may be optional, some must have a meaningful value for the cloud service. You should implement unit tests that specify an invalid input to each of your cloud connector's methods and check that the proper input validation is performed.

#### Testing the interaction with the cloud service

The main purpose of your cloud connector is integration of a local API with a remote one. Through the local API developers can use the web service in the same way they use a local service.

The cloud connector, however, typically talks over the internet with some remote service. This means that errors in the connection with the service may occur such as timeouts or network packet loss. Your cloud connector could handle these types of errors either gracefully by retrying or simply throw an exception. Whatever type of error handling you choose, make sure you test the error handling. Often, you may need to mock out the web service you are testing against as you won't be able to reproduce certain error conditions (e.g. timeouts, connection loss) from your side.

When you create a new project a test case skeleton is created in the project.

#### Testing Mule integration

When you create a new project a Mule namespacehandler class and a schema will be automatically generated from your cloud connector class. Along with the other unit tests you should also test this Mule integration. As a rule of thumb, add at least one test for each element that is generated into the Mule schema.

A test class for the namespace handler is created in the project when it is created. The test class has some helper methods and comments that give you an idea of how to get started with writing unit tests using flows in Mule configuration.

Let's walk thru a typical test class:

```

package org.mule.module.movies;

import org.mule.api.MuleEvent;
import org.mule.construct.SimpleFlowConstruct;
import org.mule.tck.FunctionalTestCase;

public class MovieConnectorTest extends FunctionalTestCase
{

    @Override
    protected String getConfigResources()
    {
        return "config/movie-search.xml";
    }

    public void testSearch() throws Exception
    {
        String payload = "<movie/>";
        SimpleFlowConstruct flow = lookupFlowConstruct("search");
        MuleEvent event = getTestEvent(payload);
        MuleEvent responseEvent = flow.process(event);
    }

    private SimpleFlowConstruct lookupFlowConstruct(String name)
    {
        return (SimpleFlowConstruct) muleContext.getRegistry().lookupFlowConstruct(name);
    }
}

```

First of all, you will notice that it inherits from `FunctionalTestCase`. This is base class for all functional tests within Mule ESB.

The `getConfigResources` must return a resource within the classpath that contains the XML describing the Mule configuration. It is generally located within `src/main/resources`.

Next is the test method itself. Let's walk thru it line by line.

```
String payload = "<movie/>";
```

Nothing fancy here, it is just setting a variable with some payload. We do recommend that you acquire this content from a resource in your test folder.

```
SimpleFlowConstruct flow = lookupFlowConstruct("search");
```

This line will use the convienience method `lookupFlowConstruct` (that in turns uses Mule's `Registry`) to lookup for the flow that you defined inside your XML configuration.

```
MuleEvent event = getTestEvent(payload);
MuleEvent responseEvent = flow.process(event);
```

Now, again this is pretty simple. The first line will generate a test event using the payload you specified and the second one will initiate the flow using the generated test event. Assuming that your flow calls your connector all should be good.

You can alternatively get the output payload from `responseEvent` and assert as needed.

The following is an example Mule configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:movie="http://www.mulesoft.org/schema/mule/movie"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/movie mule-movie.xsd">

    <movie:config apiKey="ded7fc9a3f7607459664c8d4931772ea0" />

    <flow name="search">
        <movie:search text="The Matrix"/>
    </flow>
</mule>

```

More information about Mule functional testing can be found [here](#).

Your Rating:  Results:  0 rates

## Documenting Your Connector

### Documenting Your Connector

Documenting your connector is key, especially when you wish to submit it for wider use. For each method you wish to make available to users it is highly suggested that you fully document the method and all arguments to the method.

#### **Configurable Properties**

Your connector properties can be configured using standard JavaDoc comment. Look at this example:

```

/**
 * The username that will be used for authentication against the cloud service
 */
@Property
private String username;

```

As you can see this Java field (which is also a connector property), has a standard JavaDoc comment. This comment will be extracted and added as documentation to the connector property also.

#### **Operations**

Operations are a little more complex to document than properties. For the most part we use standard JavaDoc comments to extract the documentation of the operation itself and that of its parameters.

```

/**
 * Creates a new document in the repository.
 *
 * {@code
 * <cmis:create-document-by-id folderId="#[bean:folderId]"
 *                               filename="myfilename"
 *                               content="#[bean:content]"
 *                               mimeType="text/html"
 *                               versioningState="none" />
 * }
 *
 * @param folderId      Folder Object Id
 * @param filename       Name of the file
 * @param content        File content (no byte array or input stream for now)
 * @param mimeType       Stream content-type
 * @param versioningState An enumeration specifying what the versioing state of the newly-created
 * object MUST be. If the repository does not support versioning, the repository MUST ignore the
 * versioningState parameter.
 * @return The object id of the created
 */
@Operation
public ObjectId createDocumentById(final String folderId,
                                    final String filename,
                                    final Object content,
                                    final String mimeType,
                                    final VersioningState versioningState,
                                    final String objectType)

```

The first sentence of the doc comment should be a summary sentence, containing a concise but complete description of the operation.

Next there is an `@code` section. This special section is supposed to contain an example invocation of the operation. This will be used to fill the usage section in the documentation.

The rest is standard JavaDoc comment, containing a description for each parameter and also a description for the return value.

## Schema documentation

The XML schema for your cloud connector is automatically generated from the source code of your cloud connector class. The schema generator extracts the javadoc documentation of each method and puts it as documentation on the generated schema element. This approach makes sure that your documentation is consistent with your code.

## GitHub Readme

If you plan to make your connector available in source via GitHub then you are in luck, because there is a custom README generator. This generator will parse your project and your connectors and then build a comprehensive README including Installation, Configuration and Usage sections.

To generate this README do the following at your command line:

```
mvn mule-cloud-connector:readme-generate
```

Keep in mind that the following sections need to be present in your POM for this plugin to work:

- name
- description
- distributionManagement
- scm

You will find a README.md afterwards.

Your Rating:  Results:  0 rates

## Integration with Mule

## Integration with Mule

You can always use your cloud connector on its own but integrating the cloud connector to be used in Mule allows it to be part of more complex integration patterns. This section covers what is needed for the integration with Mule. If you used the development kit to generate your project all of these files will have been auto generated for you.

In order for the cloud connector to be used in Mule the following files must be present:

- `spring.handlers` (in `src/main/resources/META-INF`)
- `spring.schemas` (in `src/main/resources/META-INF`)
- Mule schema (automatically generated in `target/generated-resources/mule`)
- Mule namespace handler (automatically generated in `target/generated-sources/mule`)

The `spring.handlers` and `spring.schemas` files are configuration for the Spring framework. They specify which namespace handler class is responsible for this project's schema. Read all about Spring's extensible XML authoring in the [Spring documentation](#).

Since maintaining the schema and the namespace handler can be quite tedious the cloud connector development kit puts some tooling into your project's `pom.xml`. The `mule-cloud-connector-generator-plugin` generates a Mule schema and a namespace handler each time the project is built. It operates on your cloud connector class and converts all of its public methods to XML elements in the schema. Methods that start with `get` or `set` are considered bean properties for configuration purposes that will not become XML elements.

### Configuration of your cloud connector

Most of the cloud APIs require some kind of configuration e.g. an api key or a login along with a password. The `mule-cloud-connector-generator-plugin` looks for public methods in your cloud connector that start with `set` and that take a single parameter. If any matching methods are found, a an XML element named `config` is put into the generated schema and namespace handler.



The set methods should only use simple Java types like `String`, `int` or `boolean`. Other object types are not supported yet.

If the cloud connector does not require any configuration (and thus does not have any `set` methods for configuration values) you will see the following error when running your namespace handler tests:

| *Initialisation Failure: No instance of 'class org.mule.module.hostip.HostIpCloudConnector' was found in the registry*

This is due to the fact that your cloud connector class is not automatically instantiated by Mule. Add a file named `registry-bootstrap.properties` to your `src/main/resources/META-INF/services/org/mule/config` directory. Create the missing directories manually. The file contains a single key (choose any key you like, best practice is to follow the naming convention `<service name>cloudConnector`) with the value of the fully qualified class name of the cloud connector class . For example, the cloud service is "HostIp" then the `registry-bootstrap.properties` would look like this:

```
hostipCloudConnector = org.mule.module.hostip.HostIpCloudConnector
```

Read all about the `registry-bootstrap.properties` in [this section of the user guide](#).

Your Rating:

Results: 0 rates

## Submitting a Cloud Connector

### Ready for the World!

Have you developed a Cloud Connector and wish to submit it to MuleSoft to make it available to the entire community? If so, please drop us a note at [info@mulesoft.com](mailto:info@mulesoft.com) about your connector and we would be more the happy to help guide your through the submittal process.

## Recipes

### Recipes

This section is filled with practical recipes, tips, knowledge, and wisdom. We hope that you will use the material on this section to construct great connectors.

- Proper Error Handling
- Multiple Connector Configurations

Your Rating:  5 stars

Results:  0 rates

## Multiple Connector Configurations

### *Multiple Connector Configurations*

Your connector can have multiple configurations inside a Mule flow. See the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:flickr="http://www.mulesoft.org/schema/mule/flickr"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/flickr
          http://www.mulesoft.org/schema/mule/flickr/${pom.version}/mule-flickr.xsd">

    <flickr:config name="account1" apiKey="ded7fc9a3f760745944c8d4931772ea0" secretKey="mySecretKey"
format="JSON"/>
    <flickr:config name="account2" apiKey="ded7fc9a3f760745944c8d4931772ea0" secretKey="mySecretKey"
format="JSON"/>

    ...

```

As you can see from the above example, we have to flickr connector configurations. Each of them has a **name** attribute which adds an identifier to each configuration. Underneath the covers, Mule will instance two copies of your connector and register them whit-in its Registry with the supplied name.

As a developer you should not worry about this, since this ability is native to Mule and available to all connectors. There is nothing special that you need to do on the connector side to support this scenario.

```
...
<flickr:search-tags config-ref="account1" tags="mulesoft" tagMode="any" />
<flickr:search-tags config-ref="account2" tags="mule esb" tagMode="any" />

...
```

Calling operations on each configuration is exactly as it is done before, with the difference that each operation has an attached **config-ref** attribute that will signal the configuration to use for that operation.

## Proper Error Handling

### *Proper Error Handling*

The cloud connector talks via the internet to some remote service. There are a number of error scenarios that may happen during that communication: the service may not be reachable at all or the connection may be lost while data is transferred from or to the service. At a basic level, your connector should be able to detect that kind of errors. A more advanced implementation could implement retry if a connection loss was detected or could store messages to be delivered later.

At the application protocol there is often an error code or error message embedded in the resulting JSON or XML data. The HTTP status code is sometimes used, too. Your connector should check for errors on the application protocol layer and translate them to appropriate exceptions in code.

Your Rating:  5 stars

Results:  0 rates

## Mule Query Language

## Mule Query Language

MQL is a LINQ inspired query language for Java and Mule. With it you can **filter, join and transform data from Mule messages, cloud connectors, and Spring beans** in very concise code. And with the `<mql:query-service>` support inside Mule, you can instantly turn that query into a REST service. MQL is currently in beta.

Here are some quick samples:

Description	Query
Filter Data	<pre>from userManager.users   where division = 'Sales' and     (email like 'mulesoft.com' or email like 'mulesource.com')</pre> <p>Queries userManager.getUsers() and returns all users which are in Sales and have an email which contains "mulesoft.com" or "mulesource.com"</p>
Transform data into a Map/POJO	<pre>from userManager.users as u select new {   href = 'http://localhost/users/' + u.id,   name = u.firstName + ' ' + u.lastName,   division = u.division }</pre> <p>Queries userManager.getUsers() and creates new objects which have the href, name and division properties. This object will be a Map by default, but can also be a POJO by using syntax like <code>new(com.foo.UserSummary)</code> in the select statement.</p>
Join data from Salesforce	<pre>from userManager.users as user   join salesforce.query('SELECT Company, MobilePhone     FROM Lead     WHERE Email = \'' + user.email + '\'', 1)       as sfuser select new {   name = user.name,   email = user.email,   company = sfuser[0].?Company,   mobilePhone = sfuser[0].?MobilePhone }</pre> <p>Queries userManager.getUsers() and for each user calls the SalesForce cloud connector to retrieve user information based on the user's email address. In the select statement, we're creating a new object based on both the local data (name and email) and the SalesForce data (company and mobile phone).</p>
Instant REST service	<pre>&lt;mql:query-service name="UsersService"   address="http://localhost:9002/mulesoft-users"   query="from userManager.users as u where email like 'mulesoft.com'"/&gt;</pre> <p>Creates a REST services on the specified address which executes the specified query. Data is returned in JSON format.</p>
Transform Mule messages	<pre>&lt;mql:transform query="select new { name = u.name, email = u.email}"/&gt;</pre> <p>Takes a Mule message payload (POJO or JSON data) and transforms it into a new object (POJO or JSON). If the payload is a Collection, it will transform each individual object. If the payload is a single object, it will return a single transformed object.</p>

## Use Cases

Name	Description
------	-------------

Query Java Objects	Query local POJOs and transform them into different objects.
Merging Datasets	In this example, see how you can take activity streams from Twitter and Yammer and merge them into a common format.
Enrich Data	In this example, local data is enriched with data from SalesForce
Service Versioning	Easily handle two different data formats (version 1.0 and version 2.0) for your service.

## Resources

- Download
- Support
- Mule Integration
- Spring Integration
- Syntax Guide
- Javadoc
- Roadmap

## MQL Download

### Download

[MQL 0.9 Distribution](#)

### Maven

To use MQL with your Maven projects, add the following dependencies:

```
<dependency>
    <groupId>org.mule.mql</groupId>
    <artifactId>mql</artifactId>
    <version>0.9.1</version>
</dependency>
<dependency>
    <groupId>org.mule.mql</groupId>
    <artifactId>mql-examples</artifactId>
    <version>0.9.1</version>
</dependency>
```

Also, add the following repositories:

```
<repository>
    <id>muleforge-repo</id>
    <name>MuleForge Repository</name>
    <url>http://repository.mulesoft.org/releases/</url>
</repository>
<repository>
    <id>muleforge-snapshot</id>
    <name>MuleForge Release Repository</name>
    <url>http://repository.mulesoft.org/snapshots/</url>
    <snapshots>
        <enabled>true</enabled>
    </snapshots>
</repository>
```

## Installation

The JARs required to run MQL are included in the lib/ directory in the distribution. Simply drop these into your application classpath. If you're using Mule, you can place these jars in your application lib/ directory.

## MQL Enrich Data

## Enriching local data with data from the cloud

It's very common that you have data stored in multiple locations and you need to join it together into a cohesive whole. This example will show you how to take a local object declared in Spring and join it together with data from the [SalesForce cloud connector](#) to get all the information about a user in a single location.

In our example, we have a local database of User objects where we have the name and email. In salesforce we have the phone number and company name. We're going to be joining the data based on the user email. Our user object looks like this:

```
public class User {  
    private String name;  
    private String email;  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
    public String getEmail() { return email; }  
    public void setEmail(String email) { this.email = email; }  
}
```

And we have a UserManager object with a getUsers method:

```
public class UserManager {  
    // a Map of Users keyed by their email  
    private Map<String,User> users = new HashMap<String,User>();  
  
    public Collection<User> getUsers() {  
        return users.values();  
    }  
    ...  
}
```

Which is declared as a spring object in our Mule configuration:

```
<spring:bean name="userManager" class="com.mulesoft.mql.example.UserManager"/>
```

Next up, we need to configure Mule to talk to Salesforce by adding a configuration for the cloud connector:

```
<salesforce:config name="salesforce"  
    username="${salesforce.username}"  
    password="${salesforce.password}"  
    securityToken="${salesforce.securityToken}"/>
```

Your Salesforce securityToken can be obtained through the process described below.

Next, let's create a query which joins together data from Salesforce with our local UserManager. For the sake of this example, assume that you've got the name and email locally and you're trying to join information from Salesforce about the user's phone number and company. This query would query just the local UserManager:

```
from userManager.users as user  
select new {  
    name = user.name,  
    email = user.email  
}
```

The SalesForce connector has a [query method] which we can use to pass in a Salesforce select query. Since we declared a config object with the name "salesforce" above, we can refer to it inside MQL as "salesforce". We can then simply call query() as part of our join statement like this:

```
join salesforce.query('SELECT Id, Company, MobilePhone FROM Lead WHERE Email = \'' + user.email + '\'', 100) as sfuser
```

And finally, we can put this all together to create a RESTful service which listens on a URL and returns the result of this query as JSON:

```
<mql:query-service  
name="JoinCompanyAndPhone"  
address="http://localhost:9002/users"  
query="from userManager.users as user  
join salesforce.query('SELECT Id, Company, MobilePhone FROM Lead WHERE Email = \'' +  
user.email + '\'', 100) as sfuser  
select new {  
    name = user.name,  
    email = user.email,  
    company = sfuser[0].?Company,  
    mobile = sfuser[0].?MobilePhone  
}"  
/>
```

Now, if you run a query against this URL, you will receive a JSON response back with the joined data.

```
$ curl http://localhost:9002/users  
[{"email":"joe@schmoe.com", "company": "Schmoe Co", "name": "Joe Schmoe", "mobile": "(555)555-5555"}]
```

## Running the example

Next, we'll configure and start the examples application:

- [Download Mule](#) and extract the distribution
- [Download MQL](#) and extract the distribution
- Copy the examples/mql-examples-0.9.1.zip file from the MQL distribution to MULE\_HOME/apps
- Retrievre your Salesforce security token. If you don't have a Salesforce security token, log in to Salesforce, click on your name on the top right, and click Setup. Click the arrow next to Personal Information then click "Reset Security Token." Finally, click the "Rest Security Token" button and retrieve the token from your inbox.
- Start Mule with your OAuth tokens:

```
$ cd mule-standalone-3.1.2/bin  
$ ./mule -M-Dsalesforce.username=YOUR_SALESFORCE_USERNAME  
-M-Dsalesforce.password=YOUR_SALESFORCE_PASSWORD \  
-M-Dsalesforce.securityToken=YOUR_SALESFORCE_SECURITY_TOKEN
```

- Run the query:

```
$ curl http://localhost:9002/users  
[{"email":"joe@schmoe.com", "company": "Schmoe Co", "name": "Joe Schmoe", "mobile": "(555)555-5555"}]
```

## MQL Merge Datasets

### MQL Merging Datasets

If you've ever had two datasets which you need to merge into a common data format, you know that it can be a pain. (Ever used XSLT?) In this example, we take a look at what it takes to get two activity streams, [Twitter](#) and [Yammer](#), into a common format.

Here are some abbreviated snippets of each individual format:

Twitter	Yammer
<pre>{   "coordinates": null,   "favorited": false,   "created_at": "Thu Jul 15 23:26:04 +0000 2010",   "truncated": false,   "text": "My Twitter message.",   "contributors": null,   "id": 18639444000,   "geo": null,   "in_reply_to_user_id": null ,   "place": null,   "in_reply_to_screen_name": null,   "user": { ... } }</pre>	<pre>{   "client_url": "https://www.yammer.com/",   "created_at": "2011/03/28 20:39:12 +0000",   "system_message": false,   "body": {     "parsed": "My yammer message.",     "plain": "My Yammer message."   },   "sender_type": "user",   "network_id": 104604,   "thread_id": 84402777,   "web_url": "https://www.yammer.com/yammerdeveloperstestcommunity/messages/84402777" ,   "direct_message": false,   "id": 84402777,   "url": "https://www.yammer.com/api/v1/messages/84402777",   "client_type": "Web",   "message_type": "update",   "sender_id": 4022984,   "replied_to_id": null,   "attachments": ...,   "liked_by": ...,   "privacy": "public" }</pre>

Here is what we'd like to see instead:

```
[{"text": "My Twitter Message.", "source": "twitter"}, {"text": "My Yammer Message.", "source": "yammer"}]
```

Given that there are Cloud Connectors for both of these, we can write two simple queries to get a timeline from each service and put them in the desired data format:

```
from yammer.messages select new { text = body.plain, source = 'yammer' }
from twitter.publicTimeline as tweet select new { text = tweet.text, source = 'twitter' }
```

"yammer" and "twitter" can become variables that we can refer to by declaring them in our Mule application XML:

```
<twitter:config name="twitter"
  format="JSON"
  consumerKey="${twitter.consumer.key}"
  consumerSecret="${twitter.consumer.secret}"/>

<yammer:config name="yammer"
  consumerKey="${yammer.consumer.key}"
  consumerSecret="${yammer.consumer.secret}" />
```

Now, we can put this in a Mule which aggregates both data sets:

```

<flow name="get-activity">
    <inbound-endpoint address="http://localhost:9002/activity" exchange-pattern="request-response"/>
    <all>
        <mql:transform query="from yammer.messages select new { text = body/plain }" />
        <mql:transform query="from twitter.publicTimeline as tweet select new { text = tweet/text }" />
    />
    </all>
    <combine-collections-transformer/>
    <response>
        <json:object-to-json-transformer/>
    </response>
</flow>

```

Step by step, here is what this flow does:

1. Listens for requests on <http://localhost:9002/activity>
2. When a request comes in, do two queries - one to pull yammer messages and one to pull messages from the twitter public timeline. Each query transforms the result into the desired format.
3. Combines the Lists of messages from Yammer and Twitter into a single list
4. Returns the response as JSON objects

## Running the example

To run this example, you need to first register the application with Twitter and Yammer so you can use OAuth.

- [Create a Twitter account](#)
- Register your application with Twitter [http://dev.twitter.com/login?redirect\\_after\\_login=%2Fapps%2Fnew](http://dev.twitter.com/login?redirect_after_login=%2Fapps%2Fnew)
- Record your OAuth consumer key and consumer secret.
- [Create a Yammer account](#)
- Register your application with Yammer [https://www.yammer.com/client\\_applications/new](https://www.yammer.com/client_applications/new)
- Record your OAuth consumer key and consumer secret.

Next, we'll configure and start the examples application:

- [Download Mule](#) and extract the distribution
- [Download MQL](#) and extract the distribution
- Copy the examples/mql-examples-0.9.zip file from the MQL distribution to MULE\_HOME/apps
- Start Mule with your OAuth tokens:

```

$ cd mule-standalone-3.1.2/bin
$ ./mule -M-Dyammer.consumer.key=YAMMER_CONSUMER_KEY
-M-Dyammer.consumer.secret=YAMMER_CONSUMER_SECRET \
-M-Dtwitter.consumer.key=TWITTER_CONSUMER_KEY -M-Dtwitter.consumer.secret=TWITTER_CONSUMER_SECRET

```

Now, we need to perform OAuth authentication for Yammer:

- Go to <http://localhost:9002/yammer/request-authorization> in your browser
- After your browser redirects you to Yammer, click "Authorize" to authorize this application to talk to Yammer
- Take the code in the resulting page and enter the following URL in your browser:  
<http://localhost:9002/yammer/set-oauth-verifier?verifier=CODE>

Finally, retrieve the combined Yammer and Twitter activity streams:

```
$ curl -v http://localhost:9002/activity
```

## MQL Mule Integration

### Mule Integration

There are two primary ways you can use MQL inside of Mule.

1. Transformations - the <mql:transform> element allows you to perform a filter/join/transformation in a flow.
2. Query service - the <mql:query-service> element easily builds a RESTful service for you from a query

When using MQL inside of Mule, there are several special variables you can use:

Name	Description
payload	The current Mule message payload
mule	An object that allows you to do joins using mule endpoints.
Any message property name	Refer to any message property inline. For example, from payload where SomeProperty = 'foo'

To use the Mule integration, be sure to declare the MQL namespace in your Mule config file:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:mql="http://www.mulesoft.org/schema/mule/mql"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/mql
          http://www.mulesoft.org/schema/mule/mql/3.1/mule-mql.xsd
      " />
```

## Transformations

The <mql:transform> element allows you to perform a filter/join/transformation in a flow. For example:

```
<mql:transform query="from payload as p select new { name = p.firstName + ' ' + p.lastName }" />
```

The <mql:transform> element takes an optional type argument which specifies how you want to process the Mule payload.

Type	Description
POJO	The input type is a POJO that requires no transformation
JSON	The input type is JSON and should be transformed into a set of objects MQL can recognize.
AUTO	Auto detect the input type based on the Content-Type header and the data type of the Mule payload. This is the default value.

For example:

```
<mql:transform type="POJO" query="from payload as p select new { name = p.firstName + ' ' + p.lastName }" />
```

## Query service

The <mql:query-service> element easily builds a Mule service for you on a particular endpoint. With this, you can easily build RESTful JSON services. For example, this will query an object and return the data as JSON:

```

<mql:query-service
    name="Users"
    address="http://localhost:9002/users"
    query="from userManager.users as user
        select new {
            name = user.name,
            email = user.email,
            company = sfuser[0].?Company,
            mobile = sfuser[0].?MobilePhone
        }"
/>

```

You can also build services which process POJOs:

```

<mql:query-service
    name="Users"
    address="vm://users"
    type="POJO"
    query="from userManager.users as user
        select new {
            name = user.name,
            email = user.email,
            company = sfuser[0].?Company,
            mobile = sfuser[0].?MobilePhone
        }"
/>

```

## Mule Client

Sometimes, you want to query an endpoint for data within a query. You can do this as part of a from statement or a join. For example, this query will join data from an endpoint:

```

<flow name="join">
    <inbound-endpoint address="vm://join"
        exchange-pattern="request-response" />
    <mql:transform
        query="from payload as p
            join mule.send('vm://locationService', p.geoIp) as location
            select new {
                name = firstName + ' ' + lastName,
                location = location
            }" />
</flow>

```

This code will query the vm://locationService endpoint with the property "geolp" on each object. The result will be stored in the location variable for use in the select statement.

## MQL Query Java Objects

### Query Java Objects

MQL allows you to easily filter, transform and join objects from within your Java code. This example will show you how to filter a List of User objects.

Let's say your User data base has a List of users like this:

```
List<User> users = new ArrayList<User>();
users.add(new User("Dan", "Diephouse", "MuleSoft", "Engineering"));
users.add(new User("Joe", "Sales", "MuleSoft", "Sales"));
```

where your User object has fields along with getters and setters for the fields:

```
public class User {
    private String firstName;
    private String lastName;
    private String company;
    private String division;
    ...
}
```

And you would like to filter these users to only include engineers.

First, populate the context:

```
Map<String, Object> context = new HashMap<String, Object>();
context.put("users", users);
```

Then, execute your query to select them:

```
Collection<User> result =
Query.execute("from users where division = 'Engineering'", context);
```

You'll get a single object back in the collection with the user "Dan Diephouse."

## Transformation

Now, let's say, you want to create an object with a name field which combines firstName and lastName. You can create a new Map of properties by doing this:

```
Collection<Map> result =
Query.execute("from users select new { name = firstName + ' ' + lastName }", context);
```

Each Map in the collection will have a single "name" key with the firstName concatenated to the lastName.

## MQL Reference Guide

### MQL Reference Guide

#### Overview

MQL expressions follow the following structure:

```
from object as o
join ...
where ...
select new {
    property = object_expression
}
```

If you wish to only transform objects, you can use the shortcut notation which omits the from, join and where statements:

```
select new {
    property = object_expression
}
```

## Where clause

The where clause allows you to filter out objects. For example:

```
from people where age > 21 and (firstName = 'Joe' or firstName = 'Jane')
```

- The following comparators are supported: =, <, >, !=, like.
- Like performs a case insensitive search of the right hand word in the left hand object.
- Clause precedence is determine by parentheses.
- The left and right hand sides of individual where clauses can be any [object expression](#).
- You can refer to properties on the selected object with and without the "as" identifier. For example, both of these expressions are equivalent:

```
from people as p where age > 21
from people as p where p.age > 21
```

## Join clause

The join clause allows you to join data from another data source and select it. It follows the form:

```
join object_expression as variable on condition async(threads)|sync
```

Clause	Description
expression	Any object expression.
variable	The name of the variable which to put the result of the join in
condition	The condition on whether or not to join the data. An MVEL expression which returns a boolean. The on clause is optional
async/sync	Whether or not to do the join asynchronously. If asynchronous, an executor will be created to join the data. The executor can optionally be specified using the <code>Query.setExecutor</code> method. The number of threads can also be specified as part of the <code>async</code> statement, e.g. <code>async(5)</code> . "sync" specifies that the join should happen synchronously. By default it is <code>async(10)</code> .

Example:

```
from users as u join twitter.getTweetCount(u.twitterId) as tweetCount on u.twitterId
select new {
    name = u.name,
    tweetCount = tweetCount
}
```

## Selecting new objects

The select statement creates a new object for each object that is queried. If no select statement is present, the original query object is returned. The left hand side of the equals statement is the name of the property that is created. The right hand side is any [object expression](#).

### Selecting Maps

You can create new Map objects with different properties by using `select new`.

```
from users as u
select new {
    name = u.firstName + ' ' + u.lastName
}
```

## Selecting POJOs

You can instantiate instances of POJOs and set properties on them using the `select new(className)` syntax:

```
from users as u
select new(com.mulesoft.examples.UserSummary) {
    name = u.firstName + ' ' + u.lastName
}
```

## Selecting Child objects

You can also create child relationships, for example:

```
from users as u
select new {
    name = u.name,
    address = new {
        address1 = u.address1,
        address2 = u.address2,
        state = u.state,
        city = u.city
    }
}
```

This will yield a sub object for the address field which is made up of the address1, address2, state and city properties.

## Object Expressions

Object expressions are a subset of the [MVEL](#) syntax. Pieces of the MVEL syntax which are supported include:

- Value tests
- Arrays, List and Maps
- Property Navigation
- Functions
- Strings and Numbers

Examples include:

```
object.property
object.function()
object[index]
object
'foo'
123
null
x + y
x == y
```

For a more in depth guide, please read the [MVEL documentation](#)

## MQL Roadmap

## MQL Roadmap

### 1.0

- Support for generating and querying XML
- Support for creating collections on new objects in select statements
- Better JAX-RS/Jersey integration
- Finalized syntax

## MQL Service Versioning

### Service Versioning

One of the hard problems of writing services is dealing with different versions of your service. Inevitably, you need to change the format of your data, yet somehow retain backward compatibility. In the XML world, many people have gone down the XSLT approach, but the complexity of it makes it very difficult. It also doesn't map well to JSON based RESTful services. This example will show you how to use MQL inside of Mule to easily version your JAX-RS web services.

Below is a table of two different data formats, version 1.0 and version 2.0. In version 2.0 we're changing the data format to make the address its own individual object.

Version 1.0	Version 2.0
<pre>{     "name": "Joe Schmoe",     "email": "joe@schmoe.com" ,     "address": "10 Foo",     "state": "NY",     "city": "New York" }</pre>	<pre>{     "name": "Joe Schmoe",     "email": "joe@schmoe.com" ,     "address" : {     "address1": "10 Foo",     "state": "NY",     "city": "New York" } }</pre>

The conversion can be done with two simple MQL expressions. From version 1.0 to 2.0:

```
from payload as u
select new(com.mulesoft.mql.example.User) {
    name = u.name,
    email = u.email,
    address = new(com.mulesoft.mql.example.Address) {
        address = u.address,
        city = u.city,
        state = u.state
    }
}
```

And from version 2.0 to 1.0:

```

from payload as u
select new {
    name = u.name,
    email = u.email,
    address = u.address.address,
    city = u.address.city,
    state = u.address.state
}

```

Let's create our JAX-RS resource for version 2.0 of our service:

```

import java.util.Collection;
import javax.ws.rs.*;

@Path("/")
@Produces("application/json")
@Consumes("application/json")
public class UserResource {
    // a Map of Users keyed by their email
    private UserManager userManager = new UserManager();

    @GET
    public Collection<User> getUsers() {
        return userManager.getUsers();
    }

    @POST
    public User addUser(User user) {
        userManager.addUser(user);
        return user;
    }
}

```

This simply returns a list of users and adds users when the client does a POST request. Version 2.0 of this service can then be registered as a Mule endpoint:

```

<flow name="v2">
    <inbound-endpoint address="http://localhost:9002/api/v2/users" exchange-pattern="request-response"
/>
    <jersey:resources>
        <component>
            <singleton-object class="com.mulesoft.mql.example.UserResource"/>
        </component>
    </jersey:resources>
</flow>

```

We can now create an endpoint for version 1.0 of our service as well based on the MQL transformations. Here is an example. First, we'll declare our MQL transformers in the configuration:

```

<!-- Converts the old User object to the new version -->
<mql:transform name="V1-to-V2" query="from payload as u ..." />

<!-- Converts the new User object to the old version -->
<mql:transform name="V2-to-V1" query="from payload as u ..." />

```

And now, we'll create a flow which performs these transformations on a /api/v1 url:

```

<flow name="v1">
    <inbound-endpoint address="http://localhost:9002/api/v1/users" exchange-pattern="request-response"
/>

    <!-- Transform from old version if there is a request payload -->
    <processor-chain>
        <expression-filter expression="message.getInboundProperty('http.method') == 'POST'
                                || message.getInboundProperty('http.method') == 'PUT'"
                            evaluator="groovy"/>
        <transformer ref="V1-to-V2"/>
    </processor-chain>

    <jersey:resources>
        <component>
            <singleton-object class="com.mulesoft.mql.example.UserResource"/>
        </component>
    </jersey:resources>

    <!-- Transform to old version -->
    <transformer ref="V2-to-V1"/>
</flow>

```

And presto, you have two versions of your service running with only one codebase!

## Running the Example

### Running the example

Next, we'll configure and start the examples application:

- Download Mule and extract the distribution
- Download MQL and extract the distribution
- Copy the examples/mql-examples-0.9.zip file from the MQL distribution to MULE\_HOME/apps
- Start Mule:

```
$ cd mule-standalone-3.1.2/bin
$ ./mule
```

- Execute a query against version 1.0 service:

```
$ curl http://localhost:9002/api/v1/users/
[{"address": "123 Main St", "email": "dan@mulesoft.com", "name": "Dan Diephouse", "state": "CA", "city": "San Francisco"}]
```

- Execute a query against version 2.0 service:

```
$ curl http://localhost:9002/api/v2/users/
[{"address": {"address": "123 Main St", "state": "CA", "country": "USA", "city": "San Francisco"}, "name": "Dan Diephouse", "email": "dan@mulesoft.com"}]
```

## MQL Spring Integration

### MQL Spring Integration

MQL can integrate transparently into your Spring application. You just need to give it your ApplicationContext when you're performing your query.

For example:

```
import org.mule.mql.Query;
import org.mule.mql.spring.SpringQueryContext;

// Your Spring ApplicationContext. See the ApplicationContextAware interface to see how to retrieve
this
ApplicationContext applicationContext = ....;

// Create a Query context which delegates to this
SpringQueryContext queryContext = new SpringQueryContext(applicationContext);

Collection result = Query.execute("from yourSpringBean.property ...."/>
```

## Using Expressions

### Using Expressions

Expressions allow you to extract information from the current message or determine how to handle the message. Expressions are very useful with routers and filters for defining routing logic and for filtering out unwanted messages.

Mule ESB provides several [default expression evaluators](#), allowing you to embed expression logic in a variety of expression languages, or you can [create your own evaluators](#) to support additional languages.

This page describes how to use expressions. For more details on how to configure expressions, see [Expressions Configuration Reference](#).

### Message Property Scopes

Starting with Mule 3 expressions operating on message properties also support an optional scope qualifier syntax. Read [Message Property Scopes](#) for more info.

### Using Expressions with Transformers

This section describes the transformers that support expressions. For more information on transformers, see [Using Transformers](#).

#### Expression Transformer

The expression transformer executes one or more expressions on the current message where the result of the expression(s) will become the payload of the current message.

For example, imagine you have a service component with a message signature that accepts three arguments:

```
public class ShippingService
{
    public ShippingConfirmation makeShippingRequest(Customer customer, Item[] items, DataHandler
supportingDocumentation)
    {
        //do stuff
    }
}
```

And the message being passed to you component looks like this:

```

public interface ShippingRequestMessage
{
    public Customer getCustomer();
    public Item[] getShippingItems();
    //etc
}

```

The `<expression-transformer>` can be used to extract the fields from the `ShippingRequestMessage` to invoke the `ShippingService`. Note that we can only get two of the arguments from the `ShippingRequestMessage`: `Customer` and `Item[ ]`. The supporting documentation, which could be something like a Microsoft Word or Excel document, is an attachment to the `ShippingRequestMessage`. Attachments can be associated with any message within Mule.

```

<expression-transformer>
    <return-argument evaluator="bean" expression="customer"/>
    <return-argument evaluator="bean" expression="shippingItems"/>
    <return-argument evaluator="attachment" expression="supportingDocs" required="false"/>
</expression-transformer>

```

Here we execute three separate expressions to obtain the three arguments required to invoke the `ShippingService.makeShippingRequest()` method. The first two expressions use the `bean` evaluator to extract objects from the message. The last argument uses the `attachment` evaluator to obtain a single attachment. Note that `supportDocuments` can be null, so we set `required="false"` on the return argument.

## Message Properties Transformer

The `<message-properties-transformer>` allows you to add, remove, or rename properties dynamically or statically on the current message. For example:

```

<message-properties-transformer>
    <add-message-property key="GUID" value="#[string:#[xpath:/msg/header/ID]-#[xpath:/msg/body/@ref]]"
/>
</message-properties-transformer>

```

The above expressions extract the `<ID>` element value and the `ref` attribute on the `<body>` element, setting the result as a message property named `GUID`.

## XSLT Transformer

The XSLT transformer processes the XML payload of the message through XSLT. Using expressions, you can inject information about the current message into the XSLT as a parameter. For example:

```

<mulexml:xslt-transformer name="xslt" xslFile="./conf/xsl/cd-listing.xsl">
    <mulexml:context-property key="title" value="#[header>ListTitle]"/>
    <mulexml:context-property key="rating" value="#[header>ListRating]"/>
</mulexml:xslt-transformer>

```

When executed, the headers `ListTitle` and `ListRating` from the current message are added to the XSLT context as parameters called `title` and `rating`, respectively. You can reference these parameters inside the XSLT using the `<xsl:param>` element:

```

<xsl:param name="title"/>
<xsl:param name="rating"/>

```

## Using Expression Filters

Expression filters can be used in content-based routing to assert statements on the current message and route the message accordingly. Expression filters work in the same way as other types of Mule filters and have the same expression attributes as listed above. The expression on

the filter must evaluate to true or false. For example:

```
<expression-filter evaluator="header" expression="my-header!=null"/>
```

As usual, you can use AND, OR, and NOT filters to combine expressions.

```
<and-filter>
    <expression-filter evaluator="header" expression="origin-country=USA"/>
    <expression-filter evaluator="groovy" expression="payload.purchase.amount > 10000"/>
</and-filter>
```

Note that expression filters support a sub-set of all expression evaluators, because filters should only evaluate against the current message. For example, there is no point in using a function expression on a filter. The supported expression evaluators are: bean, custom, exception-type, groovy, header, xpath, ognl, payload-type, regex, wildcard, and xpath. For more information on expression evaluators, see [Expressions Configuration Reference](#).

For more information on filters, see [Using Filters](#).

## Using Expression Routers

Expression routers use expressions to determine where to route a message. Usually, outbound routers will support expressions. This section describes each of the Mule routers that support expressions. For more information on routers, see [Using Message Routers](#).

### Expression Recipient List Router

The `<expression-recipient-list>` router will evaluate an expression on the current message to obtain a list of one or more recipients to send the current message. Following is an example of an XML message:

```
<message>
    <header>
        <routing-slip>
            <recipient>http://mycompany.com/service1</recipient>
            <recipient>http://mycompany.com/service2</recipient>
        </routing-slip>
    </header>
    <body>
        ...
    </body>
</message>
```

The following router configuration extracts recipients from this message. This type of routing is commonly referred to as content-based routing.

```
<outbound>
    <expression-recipient-list-router evaluator="xpath" expression=
"/message/header/routing-slip/recipient" />
</outbound>
```



#### Best Practice

This example uses physical addresses for endpoints in the message. In a real production scenario, you would use logical endpoint names that map to physical endpoint addresses. These can be configured in your Mule configuration file or in a centralized registry.

### Expression Splitter Router

The `<expression-splitter-router>` can be used to route different parts of the current message to different destinations. Let's say our current message is a `FruitBowl` that contains different fruit that should be delivered to different places.

```

FruitBowl fruitBowl = new FruitBowl();
fruitBowl.addFruit(new Orange());
fruitBowl.addFruit(new Apple());
fruitBowl.addFruit(new Banana());
fruitBowl.addFruit(new Banana());

```

Now we have a `FruitBowl` containing an apple, an orange, and two bananas. When Mule receives this object, we want to route the fruit to different locations: the `AppleService`, `BananaService`, and `OrangeService`.

```

<service name="Distributor">
    <inbound>
        <jms:inbound-endpoint queue="distributor.queue"/>
    </inbound>
    <outbound>
        <!-- FruitBowl.getFruit() List -->
        <expression-splitter-router evaluator="bean" expression="fruit">
            <vm:outbound-endpoint path="apple.service.queue">
                <payload-type-filter expectedType="org.mule.tck.testmodels.fruit.Apple"/>
            </vm:outbound-endpoint>
            <vm:outbound-endpoint path="banana.service.queue">
                <payload-type-filter expectedType="org.mule.tck.testmodels.fruit.Banana"/>
            </vm:outbound-endpoint>
            <vm:outbound-endpoint path="orange.service.queue">
                <payload-type-filter expectedType="org.mule.tck.testmodels.fruit.Orange"/>
            </vm:outbound-endpoint>
        </expression-splitter-router>
    </outbound>
</service>

```

Notice that each of our outbound endpoints has a filter defined. This allows the splitter router to validate that the right object is routed to the right service. In this example, the `AppleService` and `OrangeService` will receive one request (fruit object) each and the `BananaService` will receive two. If the filters were not defined, the splitter router would send each object to the next endpoint in the list in a round robin fashion.

To read more about configuring expressions, see [Expressions Configuration Reference](#).

Your Rating:

Results: 3 rates

## Creating Expression Evaluators

### Creating Expression Evaluators

In addition to the [standard expression evaluators](#) provided with Mule ESB, you can create your own evaluators. This page describes how to create a custom evaluator, as well as how to add expression support to a custom module.

#### Creating the Custom Evaluator

To create a custom evaluator, the first step is to implement the `ExpressionEvaluator` interface. This is a simple strategy interface:

```

public interface ExpressionEvaluator extends NamedObject
{
    Object evaluate(String expression, MuleMessage message);
}

```

Note that this interface implements `NamedObject`, which allows the evaluator to be named. This is the name you use for the `evaluator` attribute when using this evaluator in config.

The arguments on the `evaluate` method are self-explanatory. The `expression` argument is the expression to evaluate on the current message being passed in.

Lets take the header expression evaluator as a concrete example. It will assume that the expression will contain a name of a header to return. **Note that this is a simplified example without message property scopes support. Consult the evaluators full source code for a more advanced version supporting scopes in expressions.**

```
public class MessageHeaderExpressionEvaluator implements ExpressionEvaluator
{
    public static final String NAME = "myEval";

    public Object evaluate(String expression, MuleMessage message)
    {
        Object result = null;
        boolean required;

        //Is the header optional? the '*' denotes optional
        if (expression.endsWith("*"))
        {
            expression = expression.substring(expression.length() - 1);
            required = false;
        }
        else
        {
            required = true;
        }

        //Look up the property on the message
        result = message.getProperty(expression);

        if (result == null && required)
        {
            throw new RequiredValueException(CoreMessages.expressionEvaluatorReturnedNull(NAME,
expression));
        }
        return result;
    }

    public String getName()
    {
        return NAME;
    }

    public void setName(String name)
    {
        throw new UnsupportedOperationException("setName");
    }
}
```

Note that the name of the expression evaluator is fixed as "myEval" so the `setName` method throws an `UnsupportedOperationException`.

## Registering the Custom Evaluator

After creating your custom expression evaluator, you must register it with Mule. There are two ways of doing this, depending on how you are configuring your Mule instance.

### Configuring the Evaluator as a Bean

If you are using XML configuration, you can just configure your expression evaluator as a bean, and Mule will discover it.

```
<spring:beans>
    <spring:bean class="org.mule.expressions.MessageHeaderExpressionEvaluator" />
</spring:beans>
```

### Bootstrapping the Evaluator

If you want your expression evaluator to be loaded automatically by Mule when your module (JAR) is on the class path, you need to add a

registry-bootstrap.properties file to your JAR under the following directory:

```
/META-INF/services/org/mule/config
```

The contents of the registry-bootstrap.properties should look something like this:

```
object.1=org.mule.expression.MessageHeaderExpressionEvaluator
```

When Mule starts, it will discover this bootstrap file before loading any configuration and will install any objects listed in the file into the local registry. For more information, see [Bootstrapping the Registry](#).

## Using the Custom Evaluator

To use the custom evaluator, you use the `custom-evaluator` attribute as follows for a transformer:

```
<expression-transformer>
  <return-argument evaluator="custom" custom-evaluator="myEval" expression="foo"/>
</expression-transformer>
```

or as follows for a filter:

```
<expression-filter evaluator="custom" custom-evaluator="myEval" expression="foo"/>
```

When embedding the expression, you can use normal syntax:

```
#[myEval:foo]
```

## Adding Expression Support to Custom Modules

The `ExpressionManager` is responsible for maintaining a list of supported Expression Evaluators and resolving expressions at run-time. If you are adding support for expressions in your custom Mule extensions, you will need access to this object. This is currently a static class so all methods can be called statically, for example:

```
Object result = ExpressionManager.evaluate("#[xpath://foo/bar]", muleMessage);
```

As of Mule 2.2, you can get the `ExpressionManager` using:

```
Object result = muleContext.getExpressionManager().evaluate("#[xpath://foo/bar]", muleMessage);
```

Note that the `muleContext` is available by implementing `MuleContextAware`. If you are extending a Mule API abstract class (i.e. `AbstractConnector`) then always check that the base class doesn't already provide the `MuleContext`.

Your Rating: 

Results:  0 rates

## Mule Expression Language

### Mule Expression Language

There is a powerful expression language for querying Mule information at run-time. It provides a unified language for querying message properties, attachments payload, Mule context information such as the current service or endpoint, and access to the registry. The syntax is

`#`[mule: followed by message, context, or registry, followed by a period and then the object to query or an evaluator followed by the values in parentheses. Following is more information on each of these types.

### Message Properties

You can extract information from the current message. You set an evaluator such as headers or payload and then specify in parentheses the values to evaluate on that part of the message. You can also set root message properties like correlationId. For more information, see Expressions Configuration Reference.

For example:

- `#`[mule:message.headers(foo, bar)] - retrieves two headers foo and bar and returns a Map
- `#`[mule:message.attachments-list(attach1, attach2\*)] - retrieves two named attachments in a List. The asterisk on attach2 indicates that it is optional
- `#`[mule:message.headers(all)] - retrieves all headers and returns as a Map
- `#`[mule:message.payload(org.dom4j.Document)] - returns the payload and converts it to an org.dom4j.Document
- `#`[mule:message.correlationId] - returns the correlationId on the message
- `#`[mule:message.map-payload(foo)] - expects a Map payload object and retrieves the property foo from the map

### Context Properties

You can use `#`[mule:context to return information about the server itself, such as the server ID, or about the current request, such as the current service name. Following are the properties you can return from the Mule context:

Context Property	Description
serviceName	Returns the name of the service currently processing the message
modelName	Returns the name of the model that hosts the current service
inboundEndpoint	Returns the URI string of the endpoint that received the current message
serverId	The Mule instance server ID
clusterId	The Mule instance cluster ID
domainId	The Mule instance domain ID
workingDir	Mule's working directory
homeDir	Mule's home directory

For example:

- `#`[mule:context.serviceName]
- `#`[mule:context.modelName]
- `#`[mule:context.workingDir]

### Registry Objects

You can use `#`[mule:registry to return objects you have written to the registry. For example:

- `#`[mule:registry.apple] - returns an object called apple from the registry
- `#`[mule:registry.apple\*] - returns an object called apple from the registry but is optional
- `#`[mule:registry.apple.washed] - returns the property washed on an object called apple in the registry

### Example

Please put an example in code of how to use, is this used in expression transformer? What is the evaluator?

Your Rating: 

Results:  3 rates

## Message Property Scopes

### Message Property Scopes

[ Background ] [ API ] [ Configuration Changes ] [ Internal Behavior ]

## Background

Mule has the following scopes:

- **Inbound** - properties/headers coming from the client's request
- **Invocation** - used mostly internally by Mule for the duration of this service's call, not typically utilized nor meant for end-user
- **Outbound** - values deemed to be sent out from this service. They become either request properties for the next service, or response properties in case of a synchronous invocation
- **Session** - values which are passed from invocation to invocation (note that session scope handling has been much improved in Mule 3.x)

To better understand scopes think of e.g. left-to-right flow of the message as it passes through the inbound, hits the component and goes to the outbound. Mule properties may move between scopes, either implicitly ('magic' set of correlation properties handled internally by Mule), or explicitly, when mandated and configured by a user (using `message-properties-transformer`).

Mule 2.x has always had property scopes, although they were mostly used internally and not enforced in the user-facing code. E.g. a call in 2.x like:

```
message.getProperty("Content-Type")
```

resulted in the property lookup in every scope. Another side-effect was that property set ended up having more values than wanted, sometimes interfering with the flow (requiring a user to remove those from the message explicitly).

Mule 3.x introduced a much more stringent concept, which resulted in the changes outlined below.

## API

1. `message.getProperty()` has been deprecated in favor of scope-specific variants e.g. `message.getInboundProperty()`. See `MuleMessage` javadoc for more details. The deprecated call now looks only in the outbound scope, as that was the most common use case for it.
2. `message.getPropertyNames()` has been deprecated as returned a flattened set of all scope properties. Replaced with scope-specific methods, e.g. `message.getInboundPropertyNames()`
3. `message.setProperty()` has been deprecated in favor of scope-specific variants e.g. `message.getOutboundProperty()`. See `MuleMessage` javadoc for more details. The deprecated call now sets the properties only in the outbound scope, as that was the most common use case for it.
4. `message.get/setSessionProperty()` convenience methods have been introduced for working with the session-scoped properties.

## Configuration Changes

1. `message-properties-transformer` puts much more stress on the scope attribute now as a result of these changes. By default, an **outbound** scope is used, customize via the `scope` attribute on the transformer
2. `message-property-filter` now supports an optional scope attribute
3. Expression syntax has been enhanced to support scopes. The scope part is optional, and is **case-insensitive**. Default scope is **outbound**. General syntax is:

```
<evaluator>:<scope>:<expression>
```

For example:

```
header:INBOUND:foo
```

## Internal Behavior

1. Session handler serializes session contents and saves them in the **outbound** scope. The receiving end looks for and deserializes the session from the **inbound** scope.
2. Inbound properties are no longer propagated automatically. By default, none of the inbound user properties are copied to the outbound scope. A user may explicitly propagate a property when needed. Note how in this example the default outbound scope is used as a target one, this can be customized via the `scope` attribute of the transformer:

```

<outbound>
  <pass-through-router>
    <vm:outbound-endpoint path="middle" exchange-pattern="request-response">
      <message-properties-transformer scope="outbound">
        <!-- Propagate 'myFooProperty' from the inbound to outbound -->
        <add-message-property key="myFooProperty" value=
          "#[header:INBOUND:myFooProperty]"/>
      </message-properties-transformer>
    </vm:outbound-endpoint>
  </pass-through-router>
</outbound>

```

3. VM-to-VM calls behave as expected of other transports, namely outbound properties end up in the inbound scope for the receiving VM endpoint.

Your Rating:  Results:  4 rates

## Transaction Management

### Transaction Management

[ Single-resource (Local) Transactions ] [  Multi-resource Transactions ] [ XA Transactions ] [ Transaction Manager Lookup ] [ Transaction Coordination ]

Mule's transaction framework is agnostic to the underlying transaction manager. The transaction could be a JDBC transaction, XA transaction, or a JMS transaction or message acknowledgment. All transaction types can be handled the same way. Mule transactions are configured on synchronous endpoints, where an endpoint can be configured to start a new transaction or join an existing one. Transactions are configured on an endpoint using `<transaction>`, which maps to the `org.mule.transaction.MuleTransactionConfig` class. This element defines what action an endpoint should take when it receives an event and the transaction factory to use to create transactions.



If you have multiple inbound or outbound endpoints in a service and you specify a transaction for one of them, you must specify transactions for *all* of them. For example, if you have two outbound endpoints and you specify a transaction for the first one, you must also specify a transaction for the second one.

For an excellent article on distributed transactions using both XA and non-XA approaches, see <http://www.javaworld.com/javaworld/jw-01-2009/jw-01-spring-transactions.html>. The multi-resource transaction support described below maps to the Best Efforts 1PC pattern described in the article.

For more details on the elements you configure for transactions, see [Transactions Configuration Reference](#).

For information on using exception strategies to determine whether a transaction gets committed or rolled back in case of an error, see [Error Handling with Transactions](#).

### Single-resource (Local) Transactions

Single-resource transactions (also called "local transactions") are transactions that are provided by the underlying resource, such as JDBC transactions and JMS transactions. These kind of transactions can be used to receive and/or send messages using a single resource only.

An example configuration for a single-resource transaction might look like this:

```

<jms:endpoint name="In" queue="test.In" connector-ref="jmsConnector1" />
<jms:endpoint name="Out" queue="test.Out" connector-ref="jmsConnector1" />
...
<inbound>
  <inbound-endpoint ref="In">
    <jms:transaction action="ALWAYS_BEGIN" />
  </inbound>
...
<outbound>
  <pass-through-router>
    <outbound-endpoint ref="Out">
      <jms:transaction action="ALWAYS_JOIN" />
    </outbound-endpoint>
  </pass-through-router>
</outbound>

```

This configuration defines a global JMS endpoint that receives on a "test.In" queue and another global JMS endpoint that sends on a "test.Out" queue. The action attribute tells Mule what to do for each message. In this case, a new transaction will be created for every message received. The outbound endpoint will use the resource enlisted in the current transaction, if one is running. In this case, it will use the same JMS session that has been used to receive the event. When the message has been routed from the inbound endpoint to the outbound endpoint, the transaction will be committed or rolled back.

You can send multiple messages using the [recipient list router](#), which will send all messages in the same transaction.

You can set action values on the `<transaction>` element as follows:

- NONE - Never participate in a transaction.
- ALWAYS\_BEGIN - Always start a new transaction when receiving a message. If a previous transaction exists, it commits that transaction.
- BEGIN\_OR\_JOIN - If a transaction is already in progress when an event is received, join the transaction, otherwise start a new transaction.
- ALWAYS\_JOIN - Always expects a transaction to be in progress when an event is received. If there is no transaction, an exception is thrown.
- JOIN\_IF\_POSSIBLE - Will join the current transaction if one is available. Otherwise, no transaction is created.



## Multi-resource Transactions

As of version 2.2, if you are using Mule Enterprise Edition, you can use the `<multi-transaction>` element to enable a series of operations from multiple JMS resources to be grouped into a single virtual transaction. Multi-resource transactions work without the overhead of XA. The trade-off is that XA reliability guarantees aren't provided, and your services must be ready to handle duplicates. This is very similar to a 1.5 phase commit concept (for a discussion of different approaches, see the [JavaWorld article on distributed transactions](#)).

Multi-resource transactions are useful for creating transactional non-XA bridges. For example, if you want to bridge two different JMS connectors, each of which is running local transactions instead of XA transactions, you could configure the multi-resource transaction as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jms="http://www.mulesource.org/schema/mule/jms"
      xmlns:wmq="http://www.mulesource.org/schema/mule/ee/wmq"
      xmlns:ee="http://www.mulesource.org/schema/mule/ee/core"
      xmlns:test="http://www.mulesource.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.mulesource.org/schema/mule/test
          http://www.mulesource.org/schema/mule/test/3.1/mule-test.xsd
          http://www.mulesource.org/schema/mule/core
          http://www.mulesource.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesource.org/schema/mule/ee/core
          http://www.mulesource.org/schema/mule/ee/core/3.1/mule-ee.xsd
          http://www.mulesource.org/schema/mule/jms
          http://www.mulesource.org/schema/mule/ee/wmq
          http://www.mulesource.org/schema/mule/ee/wmq/3.1/mule-wmq-ee.xsd>

    <jms:activemq-connector name="jmsConnector"
        maxRedelivery="3"
        disableTemporaryReplyToDestinations="true" />

    <wmq:connector name="wmqConnector"
        hostName="winter"
        port="1414"
        disableReplyToHandler="true"
        disableTemporaryReplyToDestinations="true"
        queueManager="MY_QUEUE_MANAGER"
        targetClient="NONJMS_MQ"
        transportType="CLIENT_MQ_TCPIP"
        specification="1.1"
        numberOfConsumers="16"
        username=""
        password=""/>

    <model name="Multi-TX Test Model">
        <service name="TestService1">
            <inbound>
                <jms:inbound-endpoint queue="in">
                    <ee:multi-transaction action="ALWAYS_BEGIN" />
                </jms:inbound-endpoint>
            </inbound>
            <test:component/>
            <outbound>
                <pass-through-router enableCorrelation="NEVER" >
                    <wmq:outbound-endpoint queue="out">
                        <ee:multi-transaction action="ALWAYS_JOIN" />
                    </wmq:outbound-endpoint>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>

```

In this example, the local JMS transaction is started when the message is received on the "in" endpoint, and the local WMQ transaction is started when the message is sent out on the "out" endpoint. The last transaction (WMQ) is committed first, and then the previous transaction (JMS) is committed.

Note that when the inbound endpoint has a multi-resource transaction configured on it, any outbound endpoints must also be configured with multi-resource transaction support and the action set to "ALWAYS\_JOIN" to become part of the virtual transaction.

## XA Transactions

You can use XA transactions if you want to enlist multiple managed resources within the same transaction and require 100% reliability. The inbound endpoints are configured in the same manner as for single-resource transactions, but the connectors need to be configured to use XA-enabled resources.

If you run Mule outside an application server, you can use JBoss Transaction Manager to configure an embedded transaction manager.

Currently, only the following transports support XA transactions:

- VM Transport Reference
- JDBC Transport Reference
- JMS Transport Reference
- Mule WMQ Transport Reference (as of Mule Enterprise Edition 2.2)

The following example of an XA transaction configuration uses a single transaction to read from a JMS queue and write to a database.

```
<service name="JmsToJdbc">
  <inbound>
    <inbound-router>
      <jms:inbound-endpoint queue="my.queue" reuseSession="false"/>
        <xa-transaction action="ALWAYS_BEGIN" timeout="60000"/>
      </jms:inbound-endpoint>
    </inbound-router>
  </inbound>
  <outbound>
    <pass-through-router>
      <jdbc:outbound-endpoint address="writeTest" type="2">
        <xa-transaction action="ALWAYS_JOIN"/>
      </jdbc:outbound-endpoint>
    </pass-through-router>
  </outbound>
</service>
```

Because the inbound JMS endpoint has an XA transaction configured on it, any outbound endpoints must also be configured with XA transaction support to become part of the XA transaction. This requires that the transport type supports XA transactions. For this configuration to work, you will need to configure a JMS connector that uses a JMS XA Connection Factory and a JDBC connector that is configured to use an XA data source.

Note that although Java EE does not support nested transactions, XA transactions have a suspend/resume concept. Therefore, if a service is configured with an XA transaction set to ALWAYS\_BEGIN, and the message is forwarded to another service with an XA transaction set to ALWAYS\_BEGIN, the first transaction is suspended until the second transaction completes.

## XA Transaction Element and Attributes

The `xa-transaction` element is a child element of the `abstract-transaction` element. It inherits the `action` attribute from `abstract-transaction` and the `action` settings have the same meaning for `xa-transaction` as they do for `abstract-transaction`. However, `xa-transaction` does not inherit the `timeout` attribute, except as noted in the section below on setting polling frequency.

The `xa-transaction` element includes another attribute, `interactWithExternal`, which is a boolean type. When set to true, `interactWithExternal` causes Mule ESB to interact with transactions begun outside of Mule ESB. For instance, if an external transaction is active and `interactWithExternal` is set to true, then the BEGIN\_OR\_JOIN setting for `action` results in Mule ESB joining the existing transaction while the ALWAYS\_BEGIN action attribute setting causes an exception to be thrown. Note that the default value for the `interactWithExternal` attribute is false.

## Setting the Polling Frequency

When you configure an inbound JMS endpoint with XA transactions, the receiver polls every 100 ms. You can change the polling frequency by setting the `pollingFrequency` property as follows:

```
<jms:inbound-endpoint queue="my.queue" reuseSession="false">
  <xa-transaction action="ALWAYS_BEGIN" timeout="60000"/>
  <properties>
    <spring:entry key="pollingFrequency" value="5000"/>
  </properties>
</jms:inbound-endpoint>
```

This property is only applicable if you are using the `XaTransactedJmsMessageReceiver`, which is the default receiver on inbound JMS endpoints that use XA transactions. If you are using JBoss transactions, please read [here](#) for information on how to configure the `timeout` value.

## Transaction Manager Lookup

Mule uses `javax.transaction.TransactionManager` for managing transaction spanning multiple resources (XA). If you need the SUSPEND semantics for your transactions (which is what EJB's RequiresNew transaction attribute value does), you **must** use the transaction manager. Conversely, the more typical `javax.transaction.UserTransaction` is just a thin handle to a transaction manager with limited (though in most cases sufficient) functionality that does not let you suspend the current transaction.

Note: Depending on your application server vendor, the transaction manager might be available via JNDI or only through proprietary APIs.

The following table summarizes some common Java EE servers:

Application Server	Remote	Embedded	Common Location	Lookup class
JBoss	✗	✓	<code>java:/TransactionManager</code>	<code>org.mule.transaction.lookup.JBossTransactionManagerLookupFactory</code>
Weblogic	✓	✓	<code>javax.transaction.TransactionManager</code>	<code>org.mule.transaction.lookup.WeblogicTransactionManagerLookupFactory</code>
WebSphere	?	✓	<i>Proprietary API call</i>	<code>org.mule.transaction.lookup.WebsphereTransactionManagerLookupFactory</code>
Resin	✗	✓	<code>java:comp/TransactionManager</code>	<code>org.mule.transaction.lookup.Resin3TransactionManagerLookupFactory</code>
JRun	✗	✓	<code>java:/TransactionManager</code>	<code>org.mule.transaction.lookup.JRunTransactionManagerLookupFactory</code>
Other	?	✓	Specified via a <code>jndiName</code> property	<code>org.mule.transaction.lookup.GenericTransactionManagerLookupFactory</code>

For example, to use Weblogic's transaction manager, you would configure Mule as follows:

```
<transaction-manager factory="org.mule.transaction.lookup.WeblogicTransactionManagerLookupFactory" />
```

## Transaction Coordination

Transaction demarcation is set on endpoints. The actual management of transactions is handled by the [Mule Transaction Coordinator](#). Note that any transacted event flows will be synchronous. The Transaction Coordinator is a singleton manager that looks after all the transactions for a Mule instance and provides methods for binding and unbinding transaction and retrieving the current transaction state.

For example, to determine whether a transaction is an XA transaction, you could use `TransactionCoordination.getInstance().getTransaction().isXa()`.

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## Shared Transactions

### Transactions Can Now Be Shared

[ [Introducing Shared Transactions](#) ] [ [New Attribute interactWithExternal on <xa-transaction>](#) ] [ [Rules on Transaction Sharing](#) ] [ [More Information](#) ]

#### Introducing Shared Transactions

One of the strengths of the Mule ESB is its ability to share many kinds of resources with the rest of the software environment: libraries, Spring beans, transaction managers, and many more. Starting in Mule 2.2.6 and 3.0, there's another thing Mule can share: transactions.

#### New Attribute `interactWithExternal` on `<xa-transaction>`

In JTA, the Java Enterprise Edition API for transaction processing, transactions are tied to threads. When Mule is entered from user code it's possible that a transaction has already been started on the current thread, and Mule might want to join it rather than starting a new (nested) transaction. You can control this with a new, optional boolean attribute on the `<xa-transaction>` element called `interactWithExternal`. The default value is `false`, which make Mule act like it did in previous versions. The way that the two transaction-controlling attributes `action` and `interactWithExternal` combine is straightforward if you keep in mind what the two `interactWithExternal` settings mean:

- true – pay attention to transactions started outside of Mule

- false – ignore transactions started outside of Mule

## Rules on Transaction Sharing

The complete set of rules is below. For simplicity, we call a transaction started by Mule a "Mule Transaction", and a transaction started by user code a "non-Mule transaction".

- action = NONE
  - interactWithExternal=true
    - Throw an exception if any transaction is active.
  - interactWithExternal=false
    - Throw an exception if a Mule transaction is active..
- action = ALWAYS\_BEGIN
  - interactWithExternal=true
    - If no transaction is active, begin a new one.
    - Otherwise, throw an exception.
  - interactWithExternal=false
    - If no Mule transaction is active, begin a new one. If a non-Mule transaction was active, this new transaction will be nested inside it.
    - Otherwise, begin a new transaction.
- action = BEGIN\_OR\_JOIN
  - interactWithExternal=true
    - If any transaction is active is started, join it.
    - Otherwise, begin a new transaction.
  - interactWithExternal=false
    - If a Mule transaction is active, join it.
    - Otherwise, begin a new transaction. If a non-Mule transaction was active, this new transaction will be nested inside it.
- action = ALWAYS\_JOIN
  - interactWithExternal=true
    - If any transaction is active, join it.
    - Otherwise, throw an exception.
  - interactWithExternal=false
    - If a Mule transaction is active, join it.
    - Otherwise, throw an exception.
- action = JOIN\_IF\_POSSIBLE
  - interactWithExternal=true
    - If any transaction is active, join it.
  - interactWithExternal=false
    - If a Mule transaction is active, join it.

## More Information

This is documented at [Transaction Management](#), along with other information about managing transactions with Mule. In particular, the entire section on XA Transactions is extremely useful.

## Configuring Security

### Configuring Security

Mule ESB allows you to authenticate requests via endpoints using transport-specific or generic authentication methods. It also allows you to control method-level authorization on your service components. The Security Manager is responsible for authenticating requests based on one or more security providers. All security is pluggable via the [Mule security API](#), so you can easily plug in custom implementations.

For information on the elements you can configure for the Security Manager, see [Security Manager Configuration Reference](#). The following sections provide links to information on configuring different types of security managers.

### Spring Security 3.0

Spring Security is the next version of Acegi and provides a number of authentication and authorization providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO. The following topics will help you get started securing your services using Spring Security:

- Configuring the Spring Security Manager
- Component Authorization Using Spring Security
- Setting up LDAP Provider for Spring Security

### Acegi

NOTE: Acegi is replaced by Spring Security and deprecated. It is not supported in future versions of Mule.

Acegi provides a number of authentication and authorization providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO. The following topics will help you get started securing your services using Acegi:

- [Configuring the Acegi Security Manager](#)
- [Component Authorization Using Acegi](#)
- [Setting up LDAP Provider for Acegi](#)

## WS-Security and SAML

WS-Security is a standard protocol for applying security to Web services. It contains specifications on how integrity and confidentiality in a SOAP message can be enforced via XML signatures and binary security tokens such as X.509 certificates and Kerberos tickets as well as encryption headers. It ensures end-to-end security by working in the application layer as opposed to the transport layer. Mule provides the following resources for WS-Security:

- [WS-Security Example](#) The WS-Security example demonstrates the different possibilities available for incorporating WS-Security into your Mule application. This example is available in the enterprise edition of Mule as of version 2.2.3.
- [Enabling WS-Security](#) - Describes how to secure your CXF SOAP endpoints with WS-Security.
- [SAML Module](#) - Mule now supports the SAML standard for exchange of security information between systems. This module is available in the enterprise edition of Mule as of version 2.2.3

## Other Security Integration

Mule also supports the following security technologies:

- [Encryption Strategies](#) - Secure your messages by encrypting them.
- [PGP Security](#) - Secure your messages by encrypting them with PGP.
- [Jaas Security](#)

Your Rating:  Results:  1 rates

## Configuring the Spring Security Manager

### Configuring the Spring Security Manager

[ [Example](#) ] [ [Security Filters](#) ]

As of Mule 3.1, you can use Spring Security 3.0 as a Security Manager inside of Mule. You can use any of the library's security providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO. For more information on the elements you can configure for a Mule security manager, see [Security Manager Configuration Reference](#).

#### Example

The following example illustrates how to configure a single security provider on Mule, in this case an in-memory database of users. To configure the provider, we set up a `<user-service>` element and the `<authentication-manager>` to which Mule delegates.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:mule-ss="http://www.mulesoft.org/schema/mule/spring-security"
      xmlns:ss="http://www.springframework.org/schema/security"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/spring-security
          http://www.mulesoft.org/schema/mule/spring-security/3.1/mule-spring-security.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.0.xsd">

    <mule-ss:security-manager>
        <mule-ss:delegate-security-provider name="memory-provider" delegate-ref="authenticationManager" />
    </mule-ss:security-manager>

    <spring:beans>
        <ss:authentication-manager alias="authenticationManager">
            <ss:authentication-provider>
                <ss:user-service id="userService">
                    <ss:user name="ross" password="ross" authorities="ROLE_ADMIN" />
                    <ss:user name="anon" password="anon" authorities="ROLE_ANON" />
                </ss:user-service>
            </ss:authentication-provider>
        </ss:authentication-manager>
    </spring:beans>
    ...cut...
</mule>

```

## Security Filters

Security filters can be configured on an object to either authenticate inbound requests or attach credentials to outbound requests. For example, to configure an HTTP basic authorization filter on an HTTP endpoint, you would use the following endpoint security filter:

```

<inbound-endpoint address="http://localhost:4567">
    <mule-ss:http-security-filter realm="mule-realm"/>
</inbound-endpoint>

```

When a request is received, the authentication header will be read from the request and authenticated against all security providers on the Security Manager. If you only want to validate on certain ones, you can supply a comma-separated list of security provider names.

```

<inbound-endpoint address="http://localhost:4567">
    <mule-ss:http-security-filter realm="mule-realm" securityProviders="default,another"/>
</inbound-endpoint>

```

The `realm` is an optional attribute required by some servers. You only need to set this attribute if required by the server on the other end.

Your Rating: ★★★★★    Results: ★★★★★ 0 rates

## Configuring the Acegi Security Manager

## Configuring the Acegi Security Manager

[ Example Configuration ] [ Security Filters ]



### Deprecated Module

The Acegi module is deprecated. Please [upgrade](#) to the Spring Security module instead, which is a drop-in replacement. As of Mule 3.2 the Acegi module is removed from the distribution.

The Mule Acegi security manager implementation delegates to Acegi to provide authorization and authentication functions. You can use any of the Acegi security providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO. For more information on the elements you can configure for a Mule security manager, see [Security Manager Configuration Reference](#).

### Example Configuration

The following example illustrates how to configure a single security provider on Mule, in this case an in-memory DAO. Here we have a static DAO security provider that allows user credentials to be set in memory with two users: ross and anon.

```
<mule xmlns="http://www.mulesource.org/schema/mule/core/2.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:acegi="http://www.mulesource.org/schema/mule/acegi/2.2"
      ...cut...

      <spring:bean id="inMemoryDaoImpl" class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
          <spring:property name="userMap">
              <spring:value>
                  ross=ross,ROLE_ADMIN
                  anon=anon,ROLE_ANONYMOUS
              </spring:value>
          </spring:property>
      </spring:bean>

      <spring:bean id="daoAuthenticationProvider" class=
"org.acegisecurity.providers.dao.DaoAuthenticationProvider">
          <spring:property name="userDetailsService" ref="inMemoryDaoImpl"/>
      </spring:bean>

      <acegi:security-manager>
          <acegi:delegate-security-provider name="memory-dao" delegate-ref="daoAuthenticationProvider"/>
      </acegi:security-manager>
      ...cut...
  </mule>
```

### Security Filters

Security filters can be configured on an object to either authenticate inbound requests or attach credentials to outbound requests. For example, to configure an HTTP basic authorization filter on an HTTP endpoint, you would use the following endpoint security filter:

```
<inbound-endpoint address="http://localhost:4567">
    <acegi:http-security-filter realm="mule-realm" />
</inbound-endpoint>
```

When a request is received, the authentication header will be read from the request and authenticated against all security providers on the Security Manager. If you only want to validate on certain ones, you can supply a comma-separated list of security provider names.

```
<inbound-endpoint address="http://localhost:4567">
    <acegi:http-security-filter realm="mule-realm" securityProviders="default,another" />
</inbound-endpoint>
```

The `realm` is an optional attribute required by some servers. You only need to set this attribute if required by the server on the other end.

## Component Authorization Using Spring Security

### Component Authorization Using Spring Security

[ Securing Service Components ] [ Setting Security Properties on the Security Provider ]

This page describes how you can configure method-level authorization using Spring Security on your components so that users with different roles can only invoke certain service methods. Spring Security is available as of Mule 2.2.

#### Securing Service Components

To secure MethodInvocations, you must add a properly configured MethodSecurityInterceptor into the application context. The beans requiring security are chained into the interceptor. This chaining is accomplished using Spring's ProxyFactoryBean or BeanNameAutoProxyCreator. Alternatively, Spring Security provides a MethodDefinitionSourceAdvisor, which you can use with Spring's DefaultAdvisorAutoProxyCreator to automatically chain the security interceptor in front of any beans defined against the MethodSecurityInterceptor.

In addition to the daoAuthenticationProvider and inMemoryDaoImpl beans (see [Configuring Security](#)), the following beans must be configured:

- MethodSecurityInterceptor
- AuthenticationManager
- AccessDecisionManager
- AutoProxyCreator
- RoleVoter

#### *The MethodSecurityInterceptor*

The MethodSecurityInterceptor is configured with a reference to the following:

- AuthenticationManager
- AccessDecisionManager

Following is a security interceptor for intercepting calls made to the methods of a component myComponent, which defines two methods: delete and writeSomething. Roles are set on these methods as seen below in the property securityMetadataSource.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mule="http://www.mulesource.org/schema/mule/core"
       xmlns:mule-ss="http://www.mulesource.org/schema/mule/spring-security"
       ...cut...
       <bean id="myComponentSecurity" class=
"org.springframework.security.access.intercept.aopalliance.MethodSecurityInterceptor">
       <property name="authenticationManager" ref="authenticationManager"/>
       <property name="accessDecisionManager" ref="accessDecisionManager"/>
       <property name="securityMetadataSource">
           <value>
               com.foo.myComponent.delete=ROLE_ADMIN
               com.foo.myComponent.writeSomething=ROLE_ANONYMOUS
           </value>
       </property>
   </bean>
```

**Note:** Because of a limitation in Spring, you must refer to the component implementation and not the interfaces it implements when defining the security bean.

#### *The AuthenticationManager*

This bean is responsible for passing requests through a chain of AuthenticationProvider objects.

```

<bean id="authenticationManager" class="org.springframework.security.authentication.ProviderManager">
    <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider"/>
        </list>
    </property>
</bean>

```

### The AccessDecisionManager

This bean specifies that a user can access the protected methods if they have any one of the roles specified in the securityMetadataSource.

```

<bean id="accessDecisionManager" class='org.springframework.security.access.vote.AffirmativeBased'>
    <property name="decisionVoters">
        <list>
            <ref bean="roleVoter"/>
        </list>
    </property>
</bean>

```

### The AutoProxyCreator

This bean defines a proxy for the protected bean. When an application asks Spring for a myComponent bean, it will get this proxy instead.

```

<bean id="autoProxyCreator" class=
"org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="interceptorNames">
        <list>
            <value>myComponentSecurity</value>
        </list>
    </property>
    <property name="beanNames">
        <list>
            <value>myComponent</value>
        </list>
    </property>
    <property name='proxyTargetClass' value="true" />
</bean>

```

When using BeanNameAutoProxyCreator to create the required proxy for security, the configuration must contain the property proxyTargetClass set to true. Otherwise, the method passed to MethodSecurityInterceptor.invoke is the proxy's caller, not the proxy's target.

### The RoleVoter

The RoleVoter class will vote if any ConfigAttribute begins with ROLE\_. The RoleVoter is case sensitive on comparisons as well as the ROLE\_ prefix.

- It will vote to grant access if there is a GrantedAuthority, which returns a String representation (via the getAuthority() method) exactly equal to one or more ConfigAttribute objects starting with ROLE\_.
- If there is no exact match of any ConfigAttribute starting with ROLE\_, the RoleVoter will vote to deny access.
- If no ConfigAttribute begins with ROLE\_, the voter will abstain.

```

<bean id="roleVoter" class="org.springframework.security.access.vote.RoleVoter" />

```

### Setting Security Properties on the Security Provider

You can add any additional properties to the security provider in the securityProperties map. For example, this map can be used to change

Spring Security's default security strategy into one of the following:

- MODE\_THREADLOCAL: allows the authentication to be set on the current thread (this is the default strategy used by Spring Security)
- MODE\_INHERITABLETHREADLOCAL: allows authentication to be inherited from the parent thread
- MODE\_GLOBAL: allows the authentication to be set on all threads

## Securing Components in Asynchronous Systems

The use of Spring Security strategies is particularly useful for asynchronous systems, since we have to add a property on the security provider for the authentication to be set on more than one thread. In this case, we would use MODE\_GLOBAL as shown in the following example:

```
<mule-ss:security-manager>
    <mule-ss:delegate-security-provider name="memory-dao" delegate-ref="authenticationManager">
        <mule-ss:security-property name="securityMode" value="MODE_GLOBAL"/>
    </mule-ss:delegate-security-provider>
</mule-ss:security-manager>
```

Your Rating:  Results:  0 rates

## Component Authorization Using Acegi

### Component Authorization Using Acegi

[ Securing Service Components ] [ Setting Security Properties on the Security Provider ]



#### Deprecated Module

The Acegi module is deprecated. Please [upgrade](#) to the Spring Security module instead, which is a drop-in replacement. As of Mule 3.2 the Acegi module is removed from the distribution.

This page describes how you can configure method-level authorization on your components so that users with different roles can only invoke certain service methods.

### Securing Service Components

To secure MethodInvocations, developers must add a properly configured `MethodSecurityInterceptor` into the application context. The beans requiring security are chained into the interceptor. This chaining is accomplished using Spring's `ProxyFactoryBean` or `BeanNameAutoProxyCreator`. Alternatively, Acegi security provides a `MethodDefinitionSourceAdvisor`, which you can use with Spring's `DefaultAdvisorAutoProxyCreator` to automatically chain the security interceptor in front of any beans defined against the `MethodSecurityInterceptor`.

In addition to the `daoAuthenticationProvider` and `inMemoryDaoImpl` beans (see [Configuring Security](#)), the following beans must be configured:

- `MethodSecurityInterceptor`
- `AuthenticationManager`
- `AccessDecisionManager`
- `AutoProxyCreator`
- `RoleVoter`

### The `MethodSecurityInterceptor`

The `MethodSecurityInterceptor` is configured with a reference to an:

- `AuthenticationManager`
- `AccessDecisionManager`

Following is a security interceptor for intercepting calls made to the methods of a component that has an interface `myComponentIfc`, which defines two methods: `delete` and `writeSomething`. Roles are set on these methods as seen below in the property `objectDefinitionSource`.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:acegi="http://www.mulesoft.org/schema/mule/acegi"
      xmlns:https="http://www.mulesoft.org/schema/mule/https"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/acegi
          http://www.mulesoft.org/schema/mule/acegi/3.1/mule-acegi.xsd
          http://www.mulesoft.org/schema/mule/https
          http://www.mulesoft.org/schema/mule/https/3.1/mule-https.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">
    ...
    <bean id="myComponentSecurity" class=
"org.acegisecurity.intercept.method.aopalliance.MethodSecurityInterceptor">
        <property name="authenticationManager" ref="authenticationManager"/>
        <property name="accessDecisionManager" ref="accessDecisionManager"/>
        <property name="objectDefinitionSource">
            <value>
                com.foo.myComponentIfc.delete=ROLE_ADMIN
                com.foo.myComponentIfc.writeSomething=ROLE_ANONYMOUS
            </value>
        </property>
    </bean>
</mule>

```

### The AuthenticationManager

An AuthenticationManager is responsible for passing requests through a chain of AuthenticationProvider objects.

```

<bean id="authenticationManager" class='org.acegisecurity.providers.ProviderManager'>
    <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider"/>
        </list>
    </property>
</bean>

```

### The AccessDecisionManager

This bean specifies that a user can access the protected methods if they have any one of the roles specified in the objectDefinitionSource.

```

<bean id="accessDecisionManager" class='org.acegisecurity.vote.AffirmativeBased'>
    <property name="decisionVoters">
        <list>
            <ref bean="roleVoter"/>
        </list>
    </property>
</bean>

```

### The AutoProxyCreator

This bean defines a proxy for the protected bean. When an application asks Spring for a `myComponent` bean, it will get this proxy instead.

```

<bean id="autoProxyCreator" class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="interceptorNames">
        <list>
            <value>myComponentSecurity</value>
        </list>
    </property>
    <property name="beanNames">
        <list>
            <value>myComponent</value>
        </list>
    </property>
    <property name='proxyTargetClass' value="true" />
</bean>

```

When using BeanNameAutoProxyCreator to create the required proxy for security, the configuration must contain the property proxyTargetClass set to true. Otherwise, the method passed to MethodSecurityInterceptor.invoke is the proxy's caller, not the proxy's target.

### The RoleVoter

The RoleVoter class will vote if any ConfigAttribute begins with ROLE\_. The RoleVoter is case sensitive on comparisons as well as the ROLE\_ prefix.

- It will vote to grant access if there is a GrantedAuthority, which returns a String representation (via the getAuthority() method) exactly equal to one or more ConfigAttributes starting with ROLE\_.
- If there is no exact match of any ConfigAttribute starting with ROLE\_, the RoleVoter will vote to deny access.
- If no ConfigAttribute begins with ROLE\_, the voter will abstain.

```
<bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter"/>
```

### Setting Security Properties on the Security Provider

You can add any additional properties to the security provider in the securityProperties map. For example, this map can be used to change Acegi's default security strategy into one of the following:

- MODE\_THREADLOCAL, which allows the authentication to be set on the current thread (this is the default strategy used by Acegi)
- MODE\_INHERITABLETHREADLOCAL, which allows authentication to be inherited from the parent thread
- MODE\_GLOBAL, which allows the authentication to be set on all threads

### Securing Components in Asynchronous Systems

The use of Acegi's security strategies is particularly useful when using an asynchronous system, since we have to add a property on the security provider for the authentication to be set on more than one thread.

In this case, we would use MODE\_GLOBAL as seen in the example below.

```

<acegi:security-manager>
    <acegi:delegate-security-provider name="memory-dao" delegate-ref="daoAuthenticationProvider">
        <acegi:security-property name="securityMode" value="MODE_GLOBAL" />
    </acegi:delegate-security-provider>
</acegi:security-manager>

```

Your Rating: 

Results:  0 rates

## Setting up LDAP Provider for Spring Security

### Setting Up an LDAP Provider for Spring Security

Your Rating: Results:  1 rates

This page describes how you can configure a Spring Security LDAP provider, which can be used by Mule 2.2 or later as follows:

- As its security provider via [SpringProviderAdapter](#)
- To perform [component authorization](#)

For information on configuring an in-memory provider, see [Configuring Security](#).

## Declaring the Beans

You must set up two beans in Spring, an [DefaultSpringSecurityContextSource](#) and an [LdapAuthenticationProvider](#). The [DefaultSpringSecurityContextSource](#) is the access point for obtaining an LDAP context where the [LdapAuthenticationProvider](#) provides integration with the LDAP server. For example:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mule="http://www.mulesource.org/schema/mule/core"
       xmlns:mule-ss="http://www.mulesource.org/schema/mule/spring-security"
       ...cut...

    <bean id="contextSource"      class=
"org.springframework.security.ldap.DefaultSpringSecurityContextSource">
        <constructor-arg value="ldap://localhost:389/dc=com,dc=foobar"/>
        <property name="userDn"  value="cn=root,dc=com,dc=foobar"/>
        <property name="password" value="secret"/>
    </bean>

    <bean id="authenticationProvider" class=
"org.springframework.security.ldap.authentication.LdapAuthenticationProvider">
        <constructor-arg>
            <bean class="org.springframework.security.authentication.ldap.authenticator.BindAuthenticator">
                <constructor-arg ref="contextSource"/>
                <property name="userDnPatterns">
                    <list><value>uid={0},ou=people</value></list>
                </property>
            </bean>
        </constructor-arg>
        <constructor-arg>
            <bean class="org.springframework.security.ldap.populator.DefaultLdapAuthoritiesPopulator">
                <constructor-arg ref="contextSource"/>
                <constructor-arg value="ou=groups"/>
                <property name="groupRoleAttribute" value="ou"/>
            </bean>
        </constructor-arg>
    </bean>

```

## Configuring the Mule Security Provider

The [SpringSecurityProviderAdapter](#) delegates to an [AuthenticationProvider](#) such as the [LdapAuthenticationProvider](#).

```
<mule-ss:security-manager>
    <mule-ss:delegate-security-provider name="spring-security-ldap" delegate-ref=
"authenticationManager"/>
</mule-ss:security-manager>
```

With the above configuration, you can achieve endpoint-level security and other security features in Mule that require one or more security providers.

## Configuring the MethodSecurityInterceptor

The configuration for component authorization is similar to the one described in [Component Authorization Using Spring Security](#). A key point of

configuration is `securityMetadataSource`:

```
<property name="securityMetadataSource" value="org.mule.api.lifecycle.Callable.onCall=ROLE_MANAGERS"/>
```

The roles are looked up by the `DefaultLdapAuthoritiesPopulator`, which you configured in the previous section. By default, a role is prefixed with `ROLE_`, and its value is extracted and converted to uppercase from the LDAP attribute defined by the `groupRoleAttribute`.

## Setting up LDAP Provider for Acegi

### Setting Up an LDAP Provider for Acegi

[ Declaring the Beans ] [ Configuring the Mule Security Provider ] [ Configuring the MethodSecurityInterceptor ]



#### Deprecated Module

The Acegi module is deprecated. Please [upgrade](#) to the [Spring Security module](#) instead, which is a drop-in replacement. As of Mule 3.2 the Acegi module is removed from the distribution.

This page describes how you can configure an Acegi LDAP provider, which can be:

- Used by Mule as its security provider via [AcegiProviderAdapter](#)
- Used by Acegi/Spring to perform [Component Authorization](#)

For information on configuring an in-memory DAO provider, see [Configuring Security](#).

#### Declaring the Beans

You must set up two beans in Spring, an `InitialDirContextFactory` and an `LdapAuthenticationProvider`. The `InitialDirContextFactory` is the access point for obtaining an LDAP context where the `LdapAuthenticationProvider` provides integration with the LDAP server. For example:

```

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mule="http://www.mulesource.org/schema/mule/core"
    xmlns:acegi="http://www.mulesource.org/schema/mule/acegi"
    ...cut...

    <bean id="initialDirContextFactory" class="org.acegisecurity.ldap.DefaultInitialDirContextFactory">
        <constructor-arg value="ldap://localhost:389/dc=com,dc=foobar" />
        <property name="managerDn">
            <value>cn=root,dc=com,dc=foobar</value>
        </property>
        <property name="managerPassword">
            <value>secret</value>
        </property>
    </bean>

    <bean id="authenticationProvider" class=
"org.acegisecurity.providers.ldap.LdapAuthenticationProvider">
        <constructor-arg>
            <bean class="org.acegisecurity.providers.ldap.authenticator.BindAuthenticator">
                <constructor-arg>
                    <ref local="initialDirContextFactory" />
                </constructor-arg>
                <property name="userDnPatterns">
                    <list>
                        <value>uid={0},ou=people</value>
                    </list>
                </property>
            </bean>
        </constructor-arg>
        <constructor-arg>
            <bean class="org.acegisecurity.providers.ldap.populator.DefaultLdapAuthoritiesPopulator">
                <constructor-arg>
                    <ref local="initialDirContextFactory" />
                </constructor-arg>
                <constructor-arg>
                    <value>ou=groups</value>
                </constructor-arg>
                <property name="groupRoleAttribute">
                    <value>cn</value>
                </property>
                <property name="searchSubtree">
                    <value>true</value>
                </property>
                <property name="rolePrefix">
                    <value>ROLE_</value>
                </property>
                <property name="convertToUpperCase">
                    <value>true</value>
                </property>
            </bean>
        </constructor-arg>
    </bean>

```

## Configuring the Mule Security Provider

The AcegiProviderAdapter delegates to an AuthenticationProvider such as the LdapAuthenticationProvider.

```

<acegi:security-manager>
    <acegi:delegate-security-provider name="acegi-ldap" delegate-ref="authenticationProvider" />
</acegi:security-manager>

```

With the above configuration, you can achieve endpoint-level security and other security features in Mule that require one or more security providers.

## Configuring the MethodSecurityInterceptor

The configuration for component authorization is similar to the one described in [Component Authorization Using Acegi](#). A key point of configuration is `ObjectDefinitionSource`:

```
<property name="objectDefinitionSource" value="org.mule.api.lifecycle.Callable.onCall=ROLE_MANAGER"/>
```

The roles are looked up by the `DefaultLdapAuthoritiesPopulator`, which you configured in the previous section. By default, a role is prefixed with `ROLE_` and its value is extracted (and converted to uppercase) from the LDAP attribute defined by the `groupRoleAttribute`.

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## Upgrading from Acegi to Spring Security

### Upgrading from Acegi to Spring Security

[ [Adding the Namespaces](#) ] [ [Updating the Acegi Package Names](#) ] [ [Using an AuthenticationManager](#) ] [ [Simplifying the Configuration](#) ]

Spring Security is version 3.0 of the Acegi Security framework. Upgrading your Mule application to use Spring Security instead of Acegi involves the following steps:

1. Adding the necessary namespaces to your Mule configuration
2. Updating the Acegi package names
3. Updating your Mule configuration to use an `AuthenticationManager`
4. Simplification using new Spring Security elements

### Adding the Namespaces

Your Mule configuration file should have the following namespaces declared.

```
<mule xmlns="http://www.mulesource.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:spring=
      "http://www.springframework.org/schema/beans"
      xmlns:mule-ss="http://www.mulesource.org/schema/mule/spring-security"
      xmlns:ss="http://www.springframework.org/schema/security"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesource.org/schema/mule/core http://www.mulesource.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesource.org/schema/mule/spring-security
          http://www.mulesource.org/schema/mule/spring-security/3.0/mule-spring-security.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security-3.0.xsd">
...
</mule>
```

The `mule-ss` namespace is for the Mule Spring Security extensions. The `ss` namespace is for the Spring Security schema elements that are not Mule specific and allows you to use the less verbose XML that is part of Spring Security 2.0.

### Updating the Acegi Package Names

Except for the changed package names, the Spring Security API has remained largely compatible with Acegi. For example, assume you configured a `DaoAuthenticationProvider` like this one:

```
<bean class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">
    <property name="userDetailsService" ref="userService"/>
</bean>
```

To use Spring Security, you simply change the `acegisecurity` package to `springframework.security`:

```
<bean class="org.springframework.security.authentication.dao.DaoAuthenticationProvider">
    <property name="userDetailsService" ref="userService"/>
</bean>
```

Repeat this step for all your Acegi bean definitions. To find the correct spring framework package names, check the API documentation at <http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/>.

## Using an AuthenticationManager

The only major difference between Mule integration with Acegi and Spring Security is that the latter uses the `AuthenticationManager` to provider authentication functions, while the former tied in at the Acegi `AuthenticationProvider` level. With the Acegi provider, the authentication flow followed this progression:

```
AcegiProviderAdapter (Mule) -> AuthenticationProvider (Acegi)
```

With the new Spring Security adapter, it follows this progression:

```
SpringProviderAdapter (Mule) -> AuthenticationManager (Spring Security) -> AuthenticationProvider (Spring Security)
```

This allows the authentication manager to try multiple authentication providers to authenticate your messages.

Configuration of this approach requires a little more XML. For example, consider this original configuration:

```
<mule ...>
    <acegi:security-manager>
        <acegi:delegate-security-provider name="memory-dao" delegate-ref="daoAuthenticationProvider"/>
    </acegi:security-manager>

    <spring:bean id="inMemoryDaoImpl" class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">
        <spring:property name="userMap">
            <spring:value>
                ross=ross,ROLE_ADMIN
                anon=anon,ROLE_ANONYMOUS
            </spring:value>
        </spring:property>
    </spring:bean>

    <spring:bean id="daoAuthenticationProvider" class=
"org.acegisecurity.providers.dao.DaoAuthenticationProvider">
        <spring:property name="userDetailsService" ref="inMemoryDaoImpl"/>
    </spring:bean>

</mule>
```

To upgrade this configuration, you add an `AuthenticationManager`. This would result in the following:

```

<mule ...>
    <mule-ss:security-manager>
        <mule-ss:delegate-security-provider name="memory-dao" delegate-ref="authenticationManager" />
    </mule-ss:security-manager>

    <spring:bean id="inMemoryDaoImpl" class=
"org.springframework.security.core.userdetails.memory.InMemoryDaoImpl">
        <spring:property name="userMap">
            <spring:value>
                ross=ross,ROLE_ADMIN
                anon=anon,ROLE_ANONYMOUS
            </spring:value>
        </spring:property>
    </spring:bean>

    <spring:bean id="daoAuthenticationProvider" class=
"org.springframework.security.authentication.dao.DaoAuthenticationProvider">
        <spring:property name="userDetailsService" ref="inMemoryDaoImpl" />
    </spring:bean>

    <spring:bean id="authenticationManager" class=
"org.springframework.security.authentication.ProviderManager">
        <spring:property name="providers">
            <spring:list>
                <spring:ref bean="daoAuthenticationProvider" />
            </spring:list>
        </spring:property>
    </spring:bean>
</mule>

```

## Simplifying the Configuration

Spring Security 3.0 includes new XML syntax that can simplify configurations, especially in simple cases. For example, the previous example has an in-memory user database, a DAO authentication provider, and an authentication manager. This can be simplified to:

```

<mule ...>
    <mule-ss:security-manager>
        <mule-ss:delegate-security-provider name="memory-dao" delegate-ref="authenticationManager" />
    </mule-ss:security-manager>

    <spring:beans>
        <ss:authentication-manager alias="authenticationManager">
            <ss:authentication-provider>
                <ss:user-service id="userService">
                    <ss:user name="ross" password="ross" authorities="ROLE_ADMIN" />
                    <ss:user name="anon" password="anon" authorities="ROLE_ANON" />
                </ss:user-service>
            </ss:authentication-provider>
        </ss:authentication-manager>
    </spring:beans>
</mule>

```

The `<authentication-manager>` element defines the name of our `AuthenticationManager` bean. We then create a single `AuthenticationProvider` with the `<authentication-provider>` and `<user-service>` elements. This `<user-service>` is the same as our `InMemoryDaoImpl` above.

For more information on how to configure Acegi, see the following Spring documentation:

- [Spring Security Documentation](#)
- [Spring Security Javadoc](#)
- [Spring Security XML Schema reference](#)

Your Rating: 

Results:  0 rates

# Encryption Strategies

## Encryption Strategies

The Security Manager can be configured with one or more encryption strategies that can then be used by encryption transformers, security filters, or secure transports such as [SSL](#) or [HTTPS](#). These encryption strategies can greatly simplify configuration for secure messaging as they can be shared across components.

Following is an example of a password-based encryption strategy (PBE) that provides password-based encryption using JCE. Users must specify a password and optionally a salt and iteration count as well. The default algorithm is PBEWithMD5AndDES, but users can specify any valid algorithm supported by JCE.

```
<security-manager>
    <password-encryption-strategy name="PBE" password="mule" />
</security-manager>
```

This strategy can then be referenced by other components in the system such as filters or transformers.

```
<decrypt-transformer name="EncryptedToByteArray" strategy-ref="PBE" />

<service name="Svc1">
    <inbound>
        <inbound-endpoint address="vm://test">
            <encryption-security-filter strategy-ref="PBE" />
        </inbound-endpoint>
    </inbound>
    ...cut...

    <service name="Svc2">
        ...cut...
        <outbound>
            <pass-through-router>
                <outbound-endpoint address="vm://output" transformer-refs="EncryptedToByteArray" />
            </pass-through-router>
        </outbound>
    </service>
</service>
```

Your Rating: 

Results:  0 rates

# PGP Security

## PGP Security

[ Requirements ] [ Encrypting and Decrypting ] [ Configuration Reference ]

This extension adds PGP security on endpoint communication. With PGP you can achieve end-to-end security communication with signed and encrypted messages between parties.

### Requirements

#### Policy Files

If you are running JDK 1.4+ that comes with the Sun JCE by default, you must install the Unlimited Strength Jurisdiction Policy files, which can be downloaded from the following URL (note that they are listed entirely at the bottom of the page, in the Other Downloads section):

JDK 1.4  
JDK 5  
JDK 6

These files must be installed in `$JAVA_HOME$/jre/lib/security`

According to Sun, the default distribution of the JCE allows "strong, but limited strength cryptography." This means that you cannot use RSA keys

bigger than 2048 bits and no symmetric ciphers that use more than 128 bits. ElGamal is not allowed at all, thus DH/DSS cannot be used for encryption.



### Useful PGP Links

- How PGP works (intro documentation)
- GnuPG (freeware implementation)
- enigmail (extension for Thunderbird)

## Encrypting and Decrypting

To encrypt and decrypt messages you need to configure the following elements:

- A security manager: responsible of holding a security provider, which contains the key rings, and the encryption strategy to be used. This allows for the encryption of all messages using the same key or to facilitate the use of different key rings.
- A key manager: which is responsible for reading the key rings.
- A credential accessor: which determines the key ring and key manager to be used to encrypt/decrypt the message being processed.

A full example is shown below:

```
<spring:beans>
    <spring:bean id="pgpKeyManager" class="org.mule.module.pgp.PGPKeyRingImpl" init-method="initialise">

        <spring:property name="publicKeyRingFileName" value="pubring.gpg"/>
        <spring:property name="secretKeyRingFileName" value="secreng.gpg"/>
        <spring:property name="secretAliasId" value="${public.KeyId.LongValue}"/>
            <spring:property name="secretPassphrase" value="${secret.Passphrase}"/>
    </spring:bean>

    <spring:bean id="credentialAccessor" class="com.somecompany.apps.AppCredentialAccessor">
        <spring:property name="credentials" value="John Smith (TestingKey)
<![CDATA[john.smith@somecompany.com]]>"/>
    </spring:bean>
</spring:beans>

<pgp:security-manager>
    <pgp:security-provider name="pgpSecurityProvider" keyManager-ref="pgpKeyManager" />
        <pgp:keybased-encryption-strategy
            name="keyBasedEncryptionStrategy"
            keyManager-ref="pgpKeyManager"
            credentialsAccessor-ref="credentialAccessor"/>
    </pgp:security-provider>
</pgp:security-manager>
```

The pgpKeyManager (in the spring:beans tag) is the one responsible for reading the rings. You have to set all the parameters: public and secret rings, the alias id (the long value in the ring) and the secret passphrase. In the same section, you can see the credentials accessor which needs to implement the CredentialsAccessor interface basically returning the key id based on the message (MuleEvent). Finally the pgp:security-manager glues both beans.

You are ready to encrypt and decrypt messages in your flows. The following two flows show how to use the encrypt-transformer and decrypt-transformer to encrypt and decrypt files.

```

<flow name="processEncryptFiles">
<file:inbound-endpoint connector-ref="inputEncrypt"
path="file:///temp/fileInput" moveToDirectory="file:///temp/fileInputBackup"
moveToPattern="#[header:originalfilename].backup" transformer-refs="file2Bytes" />

<encrypt-transformer name="pgpEncrypt"
strategy-ref="keyBasedEncryptionStrategy" />

<file:outbound-endpoint connector-ref="output"
path="file:///temp/fileOutput" outputPattern="#{function:timestamp}-#[header:originalfilename]" />
</flow>

<flow name="processDecryptFiles">
<file:inbound-endpoint connector-ref="inputDecrypt"
path="file:///temp/fileOutput" moveToDirectory="file:///temp/fileOutputEncrypted"
moveToPattern="#[header:originalfilename].backup" transformer-refs="file2Bytes" />

<decrypt-transformer name="pgpDecrypt"
strategy-ref="keyBasedEncryptionStrategy" />

<file:outbound-endpoint connector-ref="output"
path="file:///temp/fileOutputDecrypted" outputPattern=
"#{function:timestamp}-#[header:originalfilename]" />
</flow>

```

In the following section we explain how to configure each element.

### **Configuring the Security Manager**

To configure the Security Manager you need to reference your key manager and your encryption strategy. The Key manager is simple a reference to your key manager ring.

### **Configuring the Key Manager**

To configure your key manager you have to create a spring bean as shown before. You will need to set the public and secret ring files, the alias id and the secret passphrase. As Mule uses the bouncy castle library to encrypt/decrypt messages we recommend to obtain the alias id (as a long value) using this library. If Mule does not find your id in the ring it will throw an exception and it will list all the available ids in your ring.

### **Configuring a Credential Accessor**

To configure your credential accessor you need to define a class which determines your key id. For instance the following class (used in the example) returns always the same fixed string thus all the messages will be encrypted/decrypted using the same key id. If you need to use different key ids then return different strings according to the MuleEvent received as a parameter.

```

public class FakeCredentialAccessor implements CredentialsAccessor
{
    private String credentials = "John Smith (TestingKey) <john.smith@somecompany.com>";

    public FakeCredentialAccessor()
    {
    }

    public FakeCredentialAccessor(String string)
    {
        this.credentials = string;
    }

    public String getCredentials()
    {
        return credentials;
    }

    public void setCredentials(String credentials)
    {
        this.credentials = credentials;
    }

    public Object getCredentials(MuleEvent event)
    {
        return this.credentials;
    }

    public void setCredentials(MuleEvent event, Object credentials)
    {
        // dummy
    }
}

```

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

## PGP Module

This extension adds PGP security on endpoint communication. With PGP you can achieve end-to-end security communication with signed and encrypted messages between parties.

### **Security manager**

Attributes of <security-manager...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

Child Elements of <security-manager...>

Name	Cardinality	Description
security-provider	0..1	Security provider for PGP-related functionality.
keybased-encryption-strategy	0..1	The key-based PGP encryption strategy to use.

### **Security provider**

Security provider for PGP-related functionality.

Attributes of <security-provider...>

Name	Type	Required	Default	Description
keyManager-ref	string	yes		Reference to the key manager to use.

Child Elements of <security-provider...>

Name	Cardinality	Description
------	-------------	-------------

### **Keybased encryption strategy**

The key-based PGP encryption strategy to use.

Attributes of <keybased-encryption-strategy...>

Name	Type	Required	Default	Description
keyManager-ref	string	yes		Reference to the key manager to use.
credentialsAccessor-ref	string	no		Reference to the credentials accessor to use.
checkKeyExpiry	boolean	no		Check key expiration.

Child Elements of <keybased-encryption-strategy...>

Name	Cardinality	Description
------	-------------	-------------

### **Security filter**

Filters messages based on PGP encryption.

Attributes of <security-filter...>

Name	Type	Required	Default	Description
strategyName	string	yes		The name of the PGP encryption strategy to use.
signRequired	string	yes		Whether signing is required.
keyManager-ref	string	yes		Reference to the key manager to use.
credentialsAccessor-ref	string	yes		Reference to the credentials accessor to use.

Child Elements of <security-filter...>

Name	Cardinality	Description
------	-------------	-------------

Your Rating:  Results:  0 rates

## **Jaas Security**

### **Jaas Security**

[ Using the Jaas Configuration File ] [ Passing the Credentials Directly to the Provider ] [ Passing a Non-default Login Module ] [ Configuring the Security Filter on an Endpoint ]

The `JaasSimpleAuthenticationProvider` is a security provider that provides a way to interact with the Jaas Authentication Service.

The security provider for Jaas can be configured in a couple of different ways. It allows you to configure Jaas either by passing to the provider a Jaas configuration file or by passing the required attributes directly to the `JaasSimpleAuthenticationProvider`. These two configuration methods are described below.

### **Using the Jaas Configuration File**

Usually, JAAS authentication is performed in a pluggable fashion, so applications can remain independent from underlying authentication technologies.

```

jaasTest{
    org.mule.module.jaas.loginmodule.DefaultLoginModule required
    credentials="anon:anon;Marie.Rizzo:dragon;"
};

```

The above example was saved in a file called `jaas.conf`. This file contains just one entry called `com.ss.jaasTest`, which is where the application we want to protect can be found. The entry specifies the login module that will be used to authenticate the user. As a login module, you can either use Mule's `DefaultLoginModule`, one of the login modules that come with Sun, or else create your own. In this case, we have opted for Mule's `DefaultLoginModule`.

The `required` flag that follows the login module specifies that the login module must succeed for the authentication to be considered successful. Additional flags are:

**Required** - The login module is required to succeed. If it succeeds or fails, authentication still continues to proceed down the login module list.

**Requisite** - The login module is required to succeed. If it succeeds, authentication continues down the login module list. If it fails, control immediately returns to the application.

**Sufficient** - The login module is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the login module list). If it fails, authentication continues down the login module list.

**Optional** - The login module is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the login module list.

The entry also specifies the credentials, in which we put a string of authorized users together with their passwords. The credentials are put here only when the `DefaultLoginModule` is going to be used, as the method in which the user names and passwords are obtained may vary from one login module to another.

The format of the credentials string must adhere to the following format if the `DefaultLoginModule` is going to be used:

```
<username>:<password>;
```

### Configuring the Provider in the Mule Configuration File

```

<mule xmlns="http://www.mulesource.org/schema/mule/core/2.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jaas="http://www.mulesource.org/schema/mule/jaas/2.2"
      ...cut...

      <jaas:security-manager>
          <jaas:security-provider name="jaasSecurityProvider" loginContextName="jaasTest" loginConfig=
"jaas.conf"/>
      </jaas:security-manager>

```

Note that in the above, the `loginContextName` contains the same name of the entry as in the Jaas configuration file. This name will be used for creating the login context as well as to find the complete URL of the `jaas.conf` file.

### Passing the Credentials Directly to the Provider

The second option for the configuration of the `JaasSimpleAuthenticationProvider` is to pass the configuration details that would otherwise be found in the Jaas configuration file directly to the provider.

```

<jaas:security-manager>
    <jaas:security-provider name="jaasSecurityProvider" loginContextName="jaasTest" credentials=
"anon:anon;Marie.Rizzo:dragon;"/>
</jaas:security-manager>

```

In the above configuration, note that we removed the property `loginConfig` and don't need to pass any Jaas configuration file. Instead, we simply pass the credentials to the provider (using the same format as specified above). Since no login module is specified, the `DefaultLoginModule` is used.

### Passing a Non-default Login Module

The third option is to enter your own login module.

```

<jaas:security-manager>
    <jaas:security-provider name="jaasSecurityProvider" loginContextName="jaasTest" loginModule=
    "com.sun.security.auth.module.NTLoginModule"/>
</jaas:security-manager>

```

In the above configuration, we have added the `loginModule` property, which allows you to specify the login module you want to use to authenticate the user. Since the `NTLoginModule` does not require you to input a list of accepted usernames and passwords, the property for the credentials was removed.

## Configuring the Security Filter on an Endpoint

You can use `JaasSecurityFilter` as a security filter, as follows:

```

<inbound>
    <inbound-endpoint address="vm://test">
        <jaas:jaas-security-filter/>
    </inbound-endpoint>
</inbound>

```

Your Rating:  Results:  0 rates

## SAML Module



### SAML Module

[ Using the SAML Module ] [ Choosing a SAML Profile ] [ Example ]

As of version 2.2.3, Mule enterprise offers support for the **Security Assertion Markup Language (SAML)**, which is a standard for exchange of security information between federated systems. For more information on SAML, see <http://saml.xml.org/wiki/saml-wiki-knowledgebase>.

Current support in Mule is limited to SAML 1.1 and CXF web services only. Future versions of Mule will support the use of SAML with other transports. The supported SAML module is only available in the enterprise edition of Mule, although an unsupported version is available on the [MuleForge](#).

### Using the SAML Module

This section describes how to configure the SAML module in your Mule configuration.

#### Adding the SAML Module JAR

To use the SAML module, the `mule-module-saml` JAR file must be in a location on the classpath of your application.

#### Configuring the Security Manager

To use the features of the SAML module, add the SAML module namespace to your Mule configuration file as follows:

```

<mule xmlns:saml="http://www.mulesource.org/schema/mule/saml/2.2"
      xsi:schemaLocation="http://www.mulesource.org/schema/mule/saml/2.2
      http://www.mulesource.org/schema/mule/saml/2.2/mule-saml.xsd">
    <!-- Rest of your mule configuration -->
</mule>

```

Next, you configure the SAML security manager as shown below. The following example starts off with the definition of the SAML security manager and its accompanying security provider. The security provider specifies the default security realm to use by security filters if none is specified. This is especially useful in case you have only one security realm.

```

<saml:security-manager>
    <saml:saml-security-provider name="samlSecurityProvider" default-realm="senderVouches">
        <saml:keystore-provider name="default-key-provider"
            key-store-file="classpath:saml.ks"
            key-store-type="JKS"
            key-store-password="changeit"/>
        <saml:sender-vouches-realm name="senderVouches" sign-key-alias="mulesaml"
            sign-key-password="changeit" key-provider-ref="default-key-provider" resign-assertions="true"/>
        <saml:holder-of-key-realm name="holderOfKey" key-provider-ref="default-key-provider" />
    </saml:saml-security-provider>
</saml:security-manager>

```

Within the security provider, you define a key provider, which reads keys and certificates from a standard Java keystore file. You configure this file using the normal Spring options to define resources. In this case, the keystore is read from the classpath.

In this example, two security realms are defined. One uses the sender vouches SAML scheme and is also the default realm. The other is a holder of key realm. Both use the same key provider as defined above. For more information on these realms, see [Choosing a SAML Profile](#) below.

### **Configuring Security on an Endpoint**

Once you've defined a security manager, you can configure security filters on CXF endpoints as shown in the examples below. The first example does not specify a security realm, so the default realm is used. Both filters specify the same certificate that is used to verify the SAML assertions as issued by the assertion provider.

```

<saml:cxf-security-filter certificate-alias="mulesaml" />

<saml:cxf-security-filter certificate-alias="mulesaml" security-realm="non-default" />

```

Additionally, you must create specific configurations for the various transports to instruct them to support SAML. For example, CXF has to be instructed to configure WSS4j for usage of SAML as WS-Security profile.

### **Choosing a SAML Profile**

SAML defines two different profiles: Sender-vouches (SV) and Holder-of-key (HOK).

- The Sender Vouches profile means that the sender of a message is authorized to act for one of its users towards another system. In this case, the sender of the message vouches its correctness. If both systems trust each other, this profile is appropriate.
- Holder-of-key means that the user himself is authorized to perform the actions. In this case, the owner (holder) of the key is acting. If your target system trusts the token issuer (and therefore the user) you'll use Holder-of-key.

For a more detailed description of these profiles and the use cases when they're appropriate, see the article in the SAP documentation [here](#).

### **Example**

Since SAML is used for single sign-on, authentication of the user is assumed to have already occurred, and the SAML token simply contains one or more **subjects**, which provide some information understood by other systems. In this case we will configure our service to require a SAML subject of AllowGreetingServices. To our inbound endpoint we add a `SAMLVerifyInterceptor` with a callback, which will check for the correct SAML subject:

```

<spring:bean class="org.mule.module.saml.cxf.SAMLVerifyInterceptor">
    <spring:property name="callback">
        <spring:bean class="org.mule.example.security.VerifyAuthorization">
            <spring:property name="subject" value="AllowGreetingServices" />
        </spring:bean>
    </spring:property>
</spring:bean>

```

```

public class VerifyAuthorization implements SAMLVerifyCallback
{
    private String subject;

    public SAMLAuthenticationAdapter verify(SAMLAuthenticationAdapter
samlAuthentication) throws SecurityException
    {
        SAMLS subject samlSubject = samlAuthentication.getSubject();
        if (!samlSubject.getNameIdentifier().getName().equals(subject))
        {
            throw new UnauthorizedException(...cut...

```

When the expected SAML token is added to the WS-Security header of the message, it looks like this:

```

<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
AssertionID="_40082eadbf045476e26a107e4f37861d"
IssueInstant="2009-11-13T02:26:06.569Z" Issuer="self"
MajorVersion="1" MinorVersion="1">
    <AuthenticationStatement AuthenticationInstant="2009-11-13T02:26:06.569Z"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
        <Subject>
            <NameIdentifier>AllowGreetingServices</NameIdentifier>
            <SubjectConfirmation>
                <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
            </ConfirmationMethod>
            </SubjectConfirmation>
        </Subject>
    </AuthenticationStatement>
</Assertion>

```

To verify that the received SAML token is authentic, SAML offers two different modes of trust: **Sender Vouches** and **Holder of Key**. In this case, we are using Sender Vouches, which means that the sender of the message must be trusted (e.g., via a digital signature). In Holder of Key mode, the sender of the message does not matter, but the SAML token subject must contain a key from a trusted source (e.g., an X.509 certificate from Verisign).

Your Rating:  Results:  0 rates

## Error Handling

### Error Handling

[ [Exception Strategies](#) ] [ [Using the Exception-based Router](#) ] [ [Using the Exception Type Filter](#) ] [ [Error Handling with Transactions](#) ]

Mule ESB provides several ways to handle errors. You can set exception strategies for flows, models, or individual services. You can use the exception router to specify where the message goes when an error occurs. You can also use the exception type filter for fine-grained control. This page describes the various error-handling techniques. For information on setting retry policies, which control how a connector behaves when its connection fails, see [Configuring Reconnection Strategies](#).

### Exception Strategies

Exception strategies are used to handle exception conditions when an error occurs during the processing of a message. You can set exception strategies in the following places:

- On a **flow** to set a common exception strategy for all message processors within that flow.
- On a **model** to set a common exception strategy for all services within that model.
- On a **service** to set an exception strategy for any exception within that service.

Exception strategies are used to handle component exceptions. These are exceptions that occur during the processing of a message through

Mule and include all exceptions other than system exceptions such as initializing/disconnecting connections. Typically, you customize the exception handler to control how component exceptions are logged and routed.

There is also a system exception strategy. You can consider this a fallback strategy to catch exceptions that occur at startup, connection failures, or other exceptions not associated with a flow or service, and therefore cannot use a component exception strategy. Typically, you can't do much more than log system exceptions. Therefore, you should not need to customize the system exception strategy.

For details on configuring exception strategies, see [Exception Strategy Configuration Reference](#).

## Specifying Exception Strategies

The default exception strategy is `org.mule.exception.DefaultServiceExceptionStrategy`, which you configure with the `<default-service-exception-strategy>` element.

To configure it for a flow, so that it applies to all message processors in that flow, add the element before the message processors:

```
<flow name="CreditCheck">
  <default-exception-strategy>
    <vm:outbound-endpoint path="systemErrorHandler" />
  </default-exception-strategy>
  ...
</flow>
```

To configure it at the model level, so that it applies to all services in that model, add the element before the services:

```
<model name="CreditCheck">
  <default-exception-strategy>
    <vm:outbound-endpoint path="systemErrorHandler" />
  </default-exception-strategy>
  <service> ... </service>
  <service> ... </service>
</model>
```

To configure it on a service, so that it applies to all components within that service, add it at the end of the service:

```
<model name="CreditCheck">
  <service>
    ...
    <default-exception-strategy>
      <vm:outbound-endpoint path="systemErrorHandler" />
    </default-exception-strategy>
  </service>
</model>
```

You set an endpoint on an exception strategy to forward the message that failed to a destination such as an error queue.

To implement your own strategy, your class can extend `org.mule.exception.AbstractExceptionListener`, but a recommended approach is to extend `org.mule.exception.DefaultServiceExceptionStrategy` and just overload the `defaultHandler()` method. You can set bean properties on your custom exception strategy in the same way as other Mule-configured objects using a `<properties>` element.

It is up to the `defaultHandler()` method to do all necessary processing to contain the exception, so an exception should never be thrown from an exception strategy. The exception strategy must manage fatal errors. For example, if an error queue is being used but the dispatch fails, you might want to stop the current component and fire a server notification to alert a system monitor and write the event to file. Additionally, if you want to control the payload that is returned from the exception strategy to the caller (such as returning the original message or the error message), call `AbstractExceptionListener.setReturnMessage(message)` from within your exception strategy.

If you want to change the way exceptions are logged, override the `logException()` method from `org.mule.exception.AbstractExceptionListener`.

## Using the Exception-based Router

When an exception occurs, the exception-based router `org.mule.routing.outbound.ExceptionBasedRouter` determines where the message goes. You can have multiple endpoints specified on the exception-based router, so that if the first endpoint fails with a `FatalConnectionException`, the next endpoint is tried, and then the next. If all endpoints fail, an `org.mule.api.routing.RoutingException` is thrown. Note that the

exception-based router will override the endpoint mode to synchronous while looking for a successful send, and it will resort to using the endpoint's mode for the last item in the list.

Following is an example of configuring the exception-based router:

```
<outbound>
  <exception-based-router>
    <tcp:endpoint host="10.192.111.10" port="10001" />
    <tcp:endpoint host="10.192.111.11" port="10001" />
    <tcp:endpoint host="10.192.111.12" port="10001" />
  </exception-based-router>
</outbound>
```

For more information on routers, see [Using Message Routers](#).

## Using the Exception Type Filter

You can use the exception type filter to gain fine-grained control over messages that produce errors. For example, assume you want a synchronous flow where the message is sent to a validation service, and then if it fails validation, the message and its exception are forwarded to another service AND the message and its exception are returned to the caller. You could achieve this using a chaining router and the `<exception-type-filter>` as follows:

```
<chaining-router>
  <vm:outbound-endpoint path="ValidationService" synchronous="true"/>
  <vm:outbound-endpoint path="ValidationException" synchronous="true">
    <exception-type-filter expectedType="java.lang.Exception"/>
  </vm:outbound-endpoint>
</chaining-router>
```

For more information on filters, see [Using Filters](#).

## Error Handling with Transactions

If you are using [transactions](#), you can use the `<commit-transaction>` and `<rollback-transaction>` elements to specify when a transaction gets committed or rolled back based on the name of the exception that is caught. You can set a comma-separated list of wildcard patterns that will be matched against the fully qualified classname of the current exception. For example, the following code would roll back the transaction if the exception classname begins with `com.ibm.mq`. and would commit all other transactions:

```
<default-exception-strategy>
  <commit-transaction exception-pattern="*"/>
  <rollback-transaction exception-pattern="com.ibm.mq.*"/>
  <vm:outbound-endpoint path="handleError"/>
</default-exception-strategy>
```

With the following example, you will rollback the transaction if the exception thrown is instance of `java.lang.IllegalArgumentException`.

```
<default-service-exception-strategy>
  <commit-transaction exception-pattern="*"/>
  <rollback-transaction exception-pattern="java.lang.IllegalArgumentException+"/>
  <vm:outbound-endpoint path="handleError"/>
</default-service-exception-strategy>
```

By default, all transactions are rolled back.

Your Rating: 

Results:  3 rates

# Using Web Services

## Using Web Services

Mule is designed to facilitate the use of Web services in the following ways:

- Web services can be hosted or consumed
- Transport decoupled from protocol, that is, you can send / receive over JMS, email, etc.
- JAX-WS or Simple services
- Web services can be proxied or modified without much code

[CXF Module Reference](#) - Describes how to build and consume web services with CXF.

[Using the Mule Client](#) - Describes the Mule Client, which can be used as a SOAP client.

[Proxying Web Services](#) - Describes how to use Mule as a web service gateway/proxy in various scenarios.

[Supported Web Service Standards](#) - Web service standards supported by Mule and CXF.

[Web Service Wrapper](#) - Describes how to use the `WebServiceWrapperComponent` class to send the result of a web service call to another endpoint.

[RESTpack](#) - Provides support for building RESTful services inside Mule. You can also use the REST service wrapper (a specialized Mule service component) in the [HTTP transport](#) to proxy REST-style requests to external REST/HTTP services. This wrapper component acts as a REST client and is appropriate if you do not have strict REST requirements.

[Echo Example](#) - A simple example of exposing a Mule service as a web service using Axis.

[Using .NET Web Services with Mule](#) - Tips for using .NET web services with Mule.

Your Rating:  Results:  0 rates

## Proxying Web Services

### Proxying Web Services

Mule can act as a Web Service gateway/proxy. Gateways can perform several useful functions:

- Routing to the appropriate backend service (whether remote or local)
- Message transformations, such as converting from old versions of the message format
- Protocol bridging, such as HTTP to JMS
- Validation
- Security enforcement
- WS-Policy enforcement

Mule provides several utilities that help you do this:

- Protocol bridging - allows you to forward requests from one endpoint to another. This is generally the best option for proxying Web Services.
- WSProxyService - allows you to service WSDLs locally while proxying remote web services.
- Proxying Web Services - perform WS-Security or WS-Policy actions, route based on information such as the operation or SOAP Action, and easily work with just the payload by taking advantage of CXF's web service capabilities.

The following sections provide more information on these utilities.

### Protocol Bridging

The simplest type of Web Service proxy just involves forwarding a request from one endpoint to another via service [bridging](#). You can forward the data streams directly, or you can process and transform the XML. If you are doing content-based routing, this is often the best option, as it will add less overhead than a full CXF proxy (which is only needed in certain cases).

Following is a simple configuration example that forwards a request from one HTTP endpoint to another:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd">

    <flow name="HttpProxyService">
        <http:inbound-endpoint address="http://localhost:8888" exchange-pattern="request-response"/>
        <http:outbound-endpoint address="http://www.webservicex.net#[header:INBOUND:http.request]"
                               exchange-pattern="request-response"/>
    </flow>
</mule>

```

This type of bridge can be combined with filters for easy message routing. For fast XPath routing of messages, you can use the [SXC Module Reference](#).

Alternatively, you can use [Bridge Pattern](#).

## WSProxyService

The WSProxyService allows you to serve WSDLs locally while proxying remote web services. This is handy when you have an alternate WSDL you want to service, or if you don't want WSDL requests to be routed with all the other SOAP message requests. Any request that comes in with a "?wsdl" attached to the HTTP URL will be redirected, and the specified WSDL will be served instead.

To configure this for your service, you must first define a WSProxyService bean with your WSDL:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core/2.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http/2.2"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core/2.2 http://www.mulesoft.org/schema/mule/core/2.2/mule.xsd
          http://www.mulesoft.org/schema/mule/http/2.2
          http://www.mulesoft.org/schema/mule/http/2.2/mule-http.xsd">

    <spring:bean name="WSProxyService" class="org.mule.module.cxf.WSProxyService">
        <spring:property name="wsdlFile" value="localWsdl.wsdl"/>
    </spring:bean>

```

Next, define your service bridge to use this component:

```

<service name="HttpWebServiceBridge">
  <inbound>
    <inbound-endpoint address="http://localhost:8080/my/service" synchronous="true"/>
  </inbound>
  <component>
    <spring-object bean="WSProxyService" />
  </component>
  <outbound>
    <chaining-router>
      <outbound-endpoint address="http://acme.come/remote/web/service" synchronous="true"/>
    </chaining-router>
  </outbound>
</service>

```

Now any request to "http://localhost:8080/my/service?wsdl" will cause your WSDL to be served.

## Using CXF Proxy

While many times you can proxy web services without using CXF, there are several cases where you might want to use CXF proxies:

- To work directly with the SOAP body, such as adding XML directly to it
- To take advantage of the CXF web service standards support to use WS-Security or WS-Addressing
- To enforce WS-Policy assertions
- To easily service a WSDL associated with your service
- To transform a SOAP body

See the [Proxifying Web Services with CXF](#) for more details.

Your Rating:  Results:  3 rates

## Using .NET Web Services with Mule

### Using .NET Web Services with Mule

Following are tips for using Mule to communicate with .NET web services.

#### Authentication and Other Security Features

There are three ways to secure a web service:

- Via an HTTP web server
- Via authentication tokens in the SOAP header
- Via WS-Security

#### *Via an HTTP Web Server*

If you are running Mule on a Web App, you can configure the web server to use security by setting security configuration in `web.xml` and in the server's configuration file.

Alternatively, to secure a web service running on Mule (where Mule is the server), you can set the `HttpBasicAuthenticationFilter` on the web service component. Any call made to the web service would have to pass through the filter that delegates the authentication to Acegi.

Another alternative would be to use HTTPS where certificates are used for authentication.

For more information see [Configuring Security](#) and [Using HTTPS with CXF](#).

#### **Using Authentication Tokens in SOAP Headers**

You can send authentication tokens through SOAP headers as long as there is an authentication provider established that is able to understand the headers and perform the authentication.

#### **Using WS-Security**

If you are using CXF, you can configure a client and service to use WS-Security. For details, see [Enabling WS-Security](#).

### **Passing Authentication Information to a Web Service**

There are three methods for passing authentication information to a web service configured on Mule:

- Pass them in the URL, such as `http://name:password@localhost:8080/MyService`
- Set the authentication items as properties when using the Mule client
- Create headers containing the authentication items and send them as properties when using the Mule client

Your Rating: 

Results:  1 rates

## **Web Service Wrapper**

### **Using the Web Service Wrapper**

The `WebServiceWrapperComponent` class allows you to send the result of a web service call to another endpoint. The web service call is performed within `WebServiceWrapperComponent`, providing the following advantages:

- You can set any type of router on inbound and outbound.
- Unlike the chaining router, it can send more than one HTTP request at a time
- The URL for the web service call can be changed at run-time by sending the URL with the message

### **Configuring the Web Service Wrapper**

To use the web service wrapper, you specify the `<wrapper-component>` configuration element. The following table describes the attributes you can set for this element. These attributes are described in more detail in the examples that follow.

Attribute	Description	Required?
address	Specifies the URL of the web service to call	Yes, unless <code>addressFromMessage</code> is set to true
addressFromMessage (default is false)	Specifies that the URL of the web service will be obtained from the message itself	Not required if <code>address</code> is set
wrapperProperties	The SOAP document style, expressed as a map of two properties: <code>style</code> , which can be set to RPC (the default), Document, Message, or Wrapped, and <code>use</code> , which can be Encoded or Literal.	No
<code>&lt;soap-method&gt;</code>	A SOAP method to call (see <a href="#">Configuring SOAP Methods</a> below)	No

The web service wrapper is generic and can be used with any type of web service stack supported by Mule, including Axis and CXF. The examples below show synchronous use cases, but the web service wrapper can also support an asynchronous use case like the [loan broker example](#).

### **Example Configuration Using the CXF Transport**

Consider the following example. The web service wrapper is configured as a Mule component, accepting messages from a VM endpoint. A call must be made to a web service on the URL `cxfrs://localhost:65081/services/TestUMO?method=onReceive` and the result must be sent to the outbound endpoint `vm://testout`.

The inbound and outbound endpoints are configured in the usual way. The address is set as an attribute on the component, specifying the web service URL that you want to call.

```

<cxn:connector name="cxn" defaultFrontend="simple" />
<model name="Sample">
    <service name="WebServiceSample">
        <inbound>
            <vm:inbound-endpoint path="testin" />
        </inbound>
        <cxn:wrapper-component address="http://localhost:65081/services/TestUMO?method=onReceive" />
        <outbound>
            <pass-through-router>
                <outbound-endpoint address="vm://testout" />
            </pass-through-router>
        </outbound>
    </service>
</model>

```

### **Example Configuration Using the addressFromMessage Property**

The "address" property is ideal to use when the web service URL for the web service is known at configuration time. However, if this URL is either not known or else needs to be changed at run-time, the "address" property can be omitted and the "addressFromMessage" property can be set to true. The following example shows this configuration:

```

<service name = "WebServiceSample2">
    <inbound>
        <vm:inbound-endpoint path = "testin2"/>
    </inbound>
    <cxn:wrapper-component addressFromMessage = "true"/>
</service>

```

The URL must be set on the message with the property name "ws.service.url".

### **Configuring SOAP Methods**

CXF endpoints are fairly easy to configure, whereas Axis needs some further configuration to set SOAP methods. You can set a SOAP method using the <soap-method> element as shown below:

```

<service name = "WebServiceSample3">
    <inbound>
        <vm:inbound-endpoint path = "queue.in" connector-ref = "VMQueue"/>
    </inbound>
    <axis:wrapper-component address = "http://localhost:62088/axis/Calculator?method=add" style =
    "WRAPPED" use = "LITERAL">
        <axis:soap-method method = "qname{add:http://muleumo.org/Calc}">
            <axis:soap-parameter parameter = "Number1" type = "int" mode = "IN"/>
            <axis:soap-parameter parameter = "Number2" type = "int" mode = "IN"/>
            <axis:soap-return type = "int"/>
        </axis:soap-method>
    </axis:wrapper-component>
    <outbound>
        <pass-through-router>
            <vm:outbound-endpoint path = "queue.out" connector-ref = "VMQueue"/>
        </pass-through-router>
    </outbound>
</service>

```

Your Rating: 

Results:  0 rates

## **Advanced Usage of Mule ESB 3**

# Advanced Usage of Mule ESB 3

This section describes advanced techniques of using Mule.

- Object Scopes
- Storing Objects in the Registry
- Using Mule with Spring
- Configuring Properties
- Mule Agents
- About Configuration Builders
- Tuning Performance
- Configuring Reconnection Strategies
- Streaming
- Bootstrapping the Registry
- Configuring Queues
- Internationalizing Strings
- Using the Mule Client
- Mule Object Stores
- Flow Processing Strategies

Your Rating: 

Results:  1 rates

## Object Scopes

### Object Scopes

[ Determining How the Mule Container Deals with Objects ] [ Three Scopes ] [ Rules About Object Scopes ] [ Examples ]

#### Determining How the Mule Container Deals with Objects

Scope (also referred to as cardinality) describes how the objects are created and managed in the Mule container.

#### Three Scopes

Three object scopes are defined in Mule:

- Singleton - only one object is created for all requests
- Prototype - a new object is created for every request or every time the object is requested from the registry
- Pooled - Only applies to component objects, but these are stored in a pool that guarantees that only one thread will access an object at any one time.

#### Rules About Object Scopes

Singleton objects must be thread-safe, since multiple threads will be accessing the same object. Therefore, any member variables need to be guarded when writing to ensure only one thread at a time changes the data.

Objects that are given prototype scope are created for each request on the object, so the object does not need to be thread-safe, since there will only ever be one thread accessing it. However, the object must be stateless, since any member variables will only exist for the lifetime of the request.

Pooled objects are thread safe since Mule will guarantee that only one thread will access the object at a time. Pooled objects can't easily maintain state on the object itself since there will be multiple instances created. The advantage of pooled over prototype is when the object may be expensive to create and creating a new instance for every message it receives will slow down the application.

When configuring Mule through XML Spring is used to translate the XML into objects that define how the Mule instance will run. All top-level objects in Mule are singletons including Service, Model, Connector, Endpoint, Agent, Security Manager, Transaction Manager. The only object that has prototype scope is Transformer (This is because transformers can be used in different contexts and potentially in different threads at the same time).

Components (POJO objects used by your service) can either be singleton, prototype or pooled.

#### Examples

##### Pooled

```
<pooled-component class="com.foo.Bar" />
```

## Singleton

```
<component>
  <singleton-object class="com.foo.Bar" />
</component>
```

## Prototype

```
<component>
  <prototype-object class="com.foo.Bar" />
</component>

<!-- or short form -->

<component class="com.foo.Bar" />
```

## Spring scopes

Also you can reference a spring as a component where the scope of the object is defined by Spring -

```
<component>
  <spring-object ref="myBean" />
</component>
```

Your Rating: 

Results:  1 rates

## Storing Objects in the Registry

### Storing Objects in the Registry

If you need to store runtime data that is available across the application, you can store the data as objects in the [Registry](#). You can get a handle to the Registry from anywhere that you have access to the [MuleContext](#), as in most Mule ESB entities. For example, you could store the object as follows:

```
muleContext.getRegistry().registerObject("foo", new MyFoo());
```

You could then update the object from somewhere else:

```
Foo foo = (Foo) muleContext.getRegistry().lookupObject("foo");
foo.setBarVal(25);
// Replace the previous object
muleContext.getRegistry().registerObject("foo", foo);
```

This approach is useful for storing an object that is needed by multiple components across your application.

Your Rating: 

Results:  0 rates

# Using Mule with Spring

## Using Mule with Spring

Mule ESB and Spring can integrate on different levels. You can choose as much or little Mule in your Spring application or Spring in your Mule application. The following pages describe in detail how you can use Mule and Spring together.

- [About the XML Configuration File](#)  
The default way to configure Mule is via a Spring 2.0 XML file.
- [Spring Application Contexts](#)  
Describes different options for how Mule creates and manages Spring Application Contexts.
- [Using Spring Beans as Service Components](#)  
How to configure Spring beans as service components in Mule.
- [Sending and Receiving Mule Events in Spring](#)  
A really easy way for your beans to send and receive Mule events via the Spring Application Context without any code changes.
- [Spring Remoting](#)  
An example of accessing Mule from an external application using Spring remoting.

Additionally, there are two Spring-related projects in the [Developer Sandbox](#).

Your Rating:  Results:  0 rates

## Sending and Receiving Mule Events in Spring

### Sending and Receiving Mule Events in Spring

You can configure Spring beans to publish events to Mule and configure Spring event listeners to receive Mule events. This page describes how to set up the configuration.

#### Spring Events Overview

Spring provides a simple mechanism for sending and receiving events between beans. To receive an event, a bean implements [ApplicationListener](#), which has a single method:

#### ApplicationListener.java

```
public void onEvent(ApplicationEvent event);
```

To publish events to listeners, you call the [publishEvent\(\)](#) method on the [ApplicationContext](#). This will publish the same event to every listener in the context. You can also plug in custom event handlers to the application context.

#### Mule Events in Spring

To start receiving Mule events, you create a bean based on [MuleEventMulticaster](#) in your Mule configuration file. This class is an [Application Event Multicaster](#) that enables Spring beans to send and receive Mule events. You also add one or more endpoints on which to receive events:

```

xmlns:spring="http://www.springframework.org/schema/beans"
...
<spring:beans>
    <spring:bean id="applicationEventMulticaster" class=
"org.mule.module.spring.events.MuleEventMulticaster">
        <spring:property name="subscriptions">
            <spring:list>
                <spring:value>jms://my.queue</value>
                <spring:value>pop3://ross:secret@mail.muleumo.org</value>
            </spring:list>
        </spring:property>
    </spring:bean>
</spring:beans>

```

With this configuration, any emails received for `ross@muleumo.org` or any JMS messages sent to `my.queue` will be received by all Spring event listeners. Note that the `MuleEventMulticaster` does not interfere with normal Spring event behavior. If a non-Mule `applicationEvent` is sent via the `ApplicationContext`, all beans registered to receive events will still get the event.

The inbound endpoints can be any valid [Mule Endpoint](#), so you can receive JMS messages, SOAP requests, files, HTTP and servlet requests, TCP, multicast, and more.

### ***Adding Bean Subscriptions***

You can have beans subscribe only to relevant events. The `MuleSubscriptionEventListener` includes two methods for getting and setting an array of endpoints on which the bean will receive events.

**TestSubscriptionBean.java**

```

package org.mule.module.spring.events;
public class TestSubscriptionEventBean extends TestMuleEventBean implements
MuleSubscriptionEventListener
{
    private String[] subscriptions;
    public void setSubscriptions(String[] subscriptions)
    {
        this.subscriptions = subscriptions;
    }
    public String[] getSubscriptions()
    {
        return subscriptions;
    }
}

```

You configure this bean like any other bean:

```

xmlns:spring="http://www.springframework.org/schema/beans"
...
<spring:beans>
    <spring:bean id="subscriptionBean" class="org.mule.module.spring.events.TestSubscriptionEventBean">
        <spring:property name="subscriptions">
            <spring:list>
                <spring:value>vm://event.*</value>
            </spring:list>
        </spring:property>
    </spring:bean>
</spring:beans>

```

### ***Publishing Events to Mule***

Publishing events is just as easy as receiving them. You use the standard `publishEvent()` method on the application context. Your bean can get a reference to the application context by implementing `ApplicationContextAware` or by querying `MuleApplicationEvent`.

```

//Create a new MuleEvent.
Object message = new String("This is a test message");
MuleApplicationEvent muleEvent = new MuleApplicationEvent(
    message, "jms://processed.queue");

//Call publishEvent on the application context, and Mule does the rest
applicationContext.publishEvent(muleEvent);

```

For more information on publishing events, see the [Error Handler Example](#).

Your Rating:  Results:  0 rates

## Spring Application Contexts

### Spring Application Contexts

[ [Single Application Context](#) ] [ [Multiple Application Contexts](#) ] [ [Using an Existing Application Context](#) ] [ [Using an Existing Application Context as Parent](#) ]

This page describes the options available for controlling how Mule creates and manages Spring application contexts for your application.

#### Single Application Context

By default, Mule will combine all resource files into a single `ApplicationContext`, whether they are "pure" Spring files or Mule configuration files. For example, the following code will create a single application context consisting of the objects in `spring-beans.xml` plus the objects in `mule-config.xml`:

##### Single Application Context

```

MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext();

ConfigurationBuilder builder = new SpringXmlConfigurationBuilder("spring-beans.xml, mule-config.xml");
builder.configure(muleContext);

muleContext.start();

```

Or, in a more abbreviated form:

```

MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext(
    new SpringXmlConfigurationBuilder("spring-beans.xml, mule-config.xml"
));
muleContext.start();

```

#### Multiple Application Contexts

You can instruct Mule to create a separate application context for each Mule configuration file. The following code will create two application contexts, one for each configuration resource:

## Multiple Application Contexts

```
MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext();

ConfigurationBuilder builder1 = new SpringXmlConfigurationBuilder("spring-beans.xml");
builder1.configure(muleContext);

ConfigurationBuilder builder2 = new SpringXmlConfigurationBuilder("mule-config.xml");
builder2.configure(muleContext);

muleContext.start();
```

## Using an Existing Application Context

If you already have an application context, you can instruct Mule to use it as follows:

### Using an Existing Application Context

```
ApplicationContext myAppContext = getMyAppContextFromSomewhereElse();

ConfigurationBuilder builder1 = new SpringConfigurationBuilder(myAppContext);
ConfigurationBuilder builder2 = new SpringXmlConfigurationBuilder("mule-config.xml");

List bList = new ArrayList();
bList.add(builder1);
bList.add(builder2);

MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext(bList);

muleContext.start();
```

## Using an Existing Application Context as Parent

You can designate an existing application context as a [parent context](#) for Mule, so that the Mule configuration can refer to and/or override beans in the application context:

### Using an Existing Application Context as Parent

```
ApplicationContext myAppContext = getMyAppContextFromSomewhereElse();

ConfigurationBuilder builder = new SpringXmlConfigurationBuilder("mule-config.xml");
builder.setParentContext(myAppContext);

MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext(builder);
muleContext.start();
```

The `MuleXmlBuilderContextListener` class checks to see if an application context (`WebApplicationContext`) has already been created by Spring, and if there is one, Mule uses it as the parent automatically.

Your Rating:  Results:  1 rates

## Using Spring Beans as Service Components

### Using Spring Beans as Service Components

[ [Defining the Beans](#) ] [ [Configuring the Beans](#) ] [ [Configuring the Component](#) ] [ [Using JNDI and EJB Session Beans](#) ]

You can construct service components from Spring beans that you define in a separate Spring context file or right in the Mule ESB configuration file. This page provides an example using two beans; a `RestaurantWaiter` bean that receives and logs orders and then passes them to the `KitchenService` bean, which receives the orders.

## Defining the Beans

The Java code for the beans look like this:

```
public class RestaurantWaiter
{
    private KitchenService kitchen = null;

    public void takeOrder(Order order) {
        //log order

        //notify kitchen
        this.kitchen.submitOrder(order);
    }

    public void setKitchenService(KitchenService kitchen) {
        this.kitchen = kitchen;
    }

    public KitchenService getKitchenService() {
        return kitchen;
    }
}
```

## Configuring the Beans

First, you configure the beans in your Spring application context:

```
<beans>
    <bean id="restaurantWaiter" scope="prototype" class="com.foo.RestaurantWaiter">
        <property name="kitchenService">
            <ref local="kitchenService"/>
        </property>
    </bean>

    <bean id="kitchenService" class="com.foo.KitchenService"/>
</beans>
```

We now have beans called restaurantWaiter and kitchenService that will be created by Spring. Notice the restaurantWaiter bean scope is set to prototype (by default, all beans in Spring are singletons unless specified otherwise). This is important if you want to pool component instances--telling Spring not to create a singleton ensures that each pooled instance will be a unique instance. If you want a singleton instance of your component, you would use Spring's default singleton scope.

If you want to configure the beans right in your Mule configuration file instead of in a separate Spring context file, you could specify them like this:

```
xmlns:spring="http://www.springframework.org/schema/beans"
...
<spring:beans>
    <spring:bean id="restaurantWaiter" scope="prototype" class="com.foo.RestaurantWaiter">
        <spring:property name="kitchenService">
            <spring:ref local="kitchenService"/>
        </spring:property>
    </spring:bean>
    <spring:bean id="kitchenService" class="com.foo.KitchenService"/>
</spring:beans>
```

## Configuring the Component

After you have configured the beans, you can create your reference to restaurantWaiter in the component. For example, the following configuration creates a component that will enable restaurantWaiter to receive events from VM. This example assumes the beans are in a separate file, so if you configured them right in the Mule configuration file, you do not need the `<spring:import>` tag.

```

xmlns:vm="http://www.mulesource.org/schema/mule/vm/2.0"
xmlns:spring="http://www.springframework.org/schema/beans"
...
<spring:import resource="conf/applicationContext.xml"/>
...
<service name="Restaurant Waiter">
  <inbound>
    <vm:inbound-endpoint path="order.queue"/>
  </inbound>
  <pooled-component>
    <spring-object bean="restaurantWaiter"/>
  </pooled-component>
</service>

```

When the Mule server starts, each of the `<service>` elements are loaded, and the bean you specified in the `<spring-object>` tag is created. When an event is received on `vm://orders.queue`, an Order object is passed to the `takeOrder()` method on the RestaurantWaiter, which then logs the order and passes it to the KitchenService.

For more information on component configuration, see [Configuring Components](#). For more information on the elements you use to configure components, see [Component Configuration Reference](#).

## Using JNDI and EJB Session Beans

If you define JNDI and EJB session beans in Spring using the generic `<bean>` element, you configure them exactly as any other Spring bean in Mule. However, if you use the `<jee>` elements to define them in Spring (`<jee:jndi-lookup>`, `<jee:local-slsb>`, and `<jee:remote-slsb>`), you must include the jee namespace and schema locations in your Mule configuration file as follows:

```

xmlns:jee="http://www.springframework.org/schema/jee"
xsi:schemaLocation="
  ...
  http://www.springframework.org/schema/jee
  http://www.springframework.org/schema/jee/spring-jee-2.5.xsd"
...
<jee:remote-slsb id="creditAgencyEJB" jndi-name="local/CreditAgency"
business-interface="org.mule.example.loanbroker.credit.CreditAgency">
  <jee:environment>
    java.naming.factory.initial=org.openejb.client.LocalInitialContextFactory
    java.naming.provider.url=rmi://localhost:1099 openejb.base=${openejb.base}
    openejb.configuration=${openejb.configuration} logging.conf=${logging.conf}
    openejb.nobanner=${openejb.nobanner}
  </jee:environment>
</jee:remote-slsb>
...
<mule:service name="CreditAgencyService">
  <mule:inbound>
    <mule:inbound-endpoint ref="CreditAgency" />
  </mule:inbound>
  <mule:component>
    <mule:spring-object bean="creditAgencyEJB" />
  </mule:component>
</mule:service>
...

```

For more information, see [Enterprise Java Beans \(EJB\) integration](#) and the `jee` schema reference on the Spring site.

Your Rating:  Results:  4 rates

## Configuring Properties

### Configuring Properties

This page describes configuring properties, such as property placeholders and system properties.

## Property Placeholders

You can use Ant-style property placeholders in your Mule ESB configuration. For example:

```
<smtp:outbound-endpoint user="${smtp.username}" password="${smtp.password}"/>
```

The values for these placeholders can be made available in a variety of ways, as described in the sections below.

## Global Properties

You can use the `<global-property>` element to set a placeholder value from within your Mule configuration, such as from within another Mule configuration file:

```
<global-property name="smtp.username" value="JSmith"/>
<global-property name="smtp.password" value="ChangeMe"/>
```

## Properties Files

To load properties from a file, you can use the standard Spring element

`<context:property-placeholder>`:

```
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd"

<context:property-placeholder location="smtp.properties"/>
```

where the contents of `smtp.properties` is:

```
smtp.username=JSmith
smtp.password=ChangeMe
```

To load multiple properties files, separate them with commas:

```
<context:property-placeholder location="email.properties,http.properties,system.properties"
placeholderPrefix="${"}/>
```

## Message Properties

You can use placeholders to perform logic on message properties such as the header. For example, if you wanted to evaluate the content-type portion of the message header, you would specify it as `##[header:INBOUND:Content-Type]`. Typically, you use message property placeholders with expressions. For more information, see [Using Expressions](#).

## System Properties

The placeholder value can come from a JDK system property. If you start Mule from the command line, you would specify the properties as follows:

```
mule -M-Dsmtp.username=JSmith -M-Dsmtp.password=ChangeMe
```

or edit the system properties in `conf/wrapper.conf` if you are deploying Mule as a webapp. When running Mule in a container, as of Mule 2.2.2 you can also specify the server ID in the `web.xml` file as follows:

```
<context-param>
  <param-name>mule.serverId</param-name>
  <param-value>MyServer</param-value>
</context-param>
```

If you start Mule programmatically, you would specify the properties as follows before creating and starting the Mule context:

```
System.getProperties().put("smtp.username", "JSmith");
System.getProperties().put("smtp.password", "ChangeMe");
```

There are also several system properties that are immutable after startup. To set these, you customize the `MuleConfiguration` using the `set` method for the property (such as `setId` for the system ID), create a `MuleContextBuilder`, load the configuration to the builder, and then create the context from the builder.

For example:

```
SpringXmlConfigurationBuilder configBuilder = new SpringXmlConfigurationBuilder("my-config.xml");
DefaultMuleConfiguration muleConfig = new DefaultMuleConfiguration();
muleConfig.setId("MY_SERVER_ID");
MuleContextBuilder contextBuilder = new DefaultMuleContextBuilder();
contextBuilder.setMuleConfiguration(muleConfig);
MuleContextFactory contextFactory = new DefaultMuleContextFactory();
MuleContext muleContext = contextFactory.createMuleContext(configBuilder, contextBuilder);
muleContext.start();
```

For information on the `set` methods you can use to set system properties, see [org.mule.config.DefaultMuleConfiguration](#). For information on configuration builders, see [About Configuration Builders](#).

## Environment Variables

There is no standard way in Java to access environment variables. However, this [link](#) has some options you might find useful.

Your Rating:  Results:  0 rates

## Mule Agents

### Using Mule Agents

[ [Configuring an Agent](#) ] [ [Creating Custom Agents](#) ]

An agent is a service that is associated with or used by Mule ESB but is not a Mule-managed component. Agents have the same lifecycle as the Mule instance they are registered with, so you can initialize and destroy resources when the Mule instance starts or is disposed.

Mule provides [several agents for JMX support](#), including notifications and remote management. You can also [create custom agents](#) to plug any functionality into Mule, such as running functionality as a background process or embedding a server in Mule.

### Configuring an Agent

Agents are defined in the Management module. To use an agent, specify the `management` namespace and schema, and then specify the properties for the agents you want to use. For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:management="http://www.mulesoft.org/schema/mule/management"
      xsi:schemaLocation=""
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
      http://www.mulesoft.org/schema/mule/management
      http://www.mulesoft.org/schema/mule/management/3.0/mule-management.xsd">
    <management:jmx-default-config port="1098" registerMx4jAdapter="true" />

    <management:log4j-notifications logName="myMuleLog" logConfigFile="mule-log.txt"/>

    <management:chainsaw-notifications chainsawPort="8080" chainsawHost="127.0.0.1" />

    <management:publish-notifications endpointAddress="vm://myService" />

```

For a list of agents provided with Mule and how to configure them, see [JMX Management](#). You can also create a custom agent as described below.

## Creating Custom Agents

To create your own agent, your agent class must implement `org.mule.api.agent.Agent`. You then configure your agent using the `<custom-agent>` element, which takes two attributes: `name` specifies a unique name for this agent, and `class` specifies the class where it's defined. If your agent requires that you pass in properties, you can specify them as name/value pairs. Note that this element is now in the core Mule namespace.

For example:

```

<custom-agent name="test-custom-agent" class="org.mule.tck.testmodels.mule.TestAgent">
  <spring:property name="frobbbit" value="woggle"/>
<custom-agent>

```

Your Rating: 

Results:  0 rates

## JMX Management

[ [Using the Default JMX Support Agent](#) ] [ [Jmx Default Config](#) ] [ [Configuring the JMX Agent](#) ] [ [Jmx Server](#) ] [ [Remote Management](#) ] [ [JMX Notifications Agent](#) ] [ [Endpoint Notifications Publisher Agent](#) ] [ [Log4J Agent](#) ] [ [Log4J Notifications Agent](#) ] [ [Chainsaw Notifications Agent](#) ] [ [MX4J Adapter](#) ] [ [Jmx Mx4j Adaptor](#) ] [ [YourKit Profiler](#) ]

**This topic relates to the most recent version of Mule ESB**

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

**Java Management Extensions (JMX)** is a simple and standard way to manage applications, devices, services, and other resources. JMX is dynamic, so you can use it to monitor and manage resources as they are created, installed, and implemented. You can also use JMX to monitor and manage the Java Virtual Machine (JVM).

Each resource is instrumented by one or more Managed Beans, or MBeans. All MBeans are registered in an MBean Server. The JMX server agent consists of an Mbean Server and a set of services for handling Mbeans.

There are several [agents](#) provided with Mule ESB for JMX support. The easiest way to configure JMX is to use the default JMX support agent.

### Using the Default JMX Support Agent

You can configure several JMX agents simultaneously using the `<jmx-default-config>` element. When set, this element registers the following agents:

- JMX agent
- RMI registry agent (if necessary) on rmi://localhost:1099
- Remote JMX access on service:jmx:rmi:///jndi/rmi://localhost:1099/server
- (Optional) Log4J JMX agent, which exposes the configuration of the Log4J instance used by Mule for JMX management
- JMX notification agent used to receive server notifications using JMX notifications
- (Optional) MX4J adapter, which provides web-based JMX management, statistics, and configuration viewing of a Mule instance

This element includes the following properties:

cache: Unexpected program error: java.lang.NullPointerException

## Jmx Default Config

### Attributes of <jmx-default-config...>

Name	Type	Required	Default	Description
registerMx4jAdapter	boolean	no		Whether to enable the MX4J adaptor.
registerLog4j	boolean	no		Whether to enable the Log4j agent.
host	string	no		The host to bind to. Normally, override this only for multi-NIC servers (default is localhost).
port	port number	no		The port on which the RMI registry will run. This is also used for remote JMX management. Default is 1099.

### Child Elements of <jmx-default-config...>

Name	Cardinality	Description
credentials	0..1	A map of username/password properties for remote JMX access. The configuration option delegates to the JmxAgent.

For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:management="http://www.mulesoft.org/schema/mule/management"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.mulesoft.org/schema/mule/core
    http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
    http://www.mulesoft.org/schema/mule/management
    http://www.mulesoft.org/schema/mule/management/3.0/mule-management.xsd">
    <management:jmx-default-config port="1098" registerMx4jAdapter="true">
        <management:credentials>
            <spring:entry key="jsmith" value="foo"/>
            <spring:entry key="dthomas" value="bar"/>
            <spring:entry key="clee" value="pwd"/>
        </management:credentials>
    </management:jmx-default-config>
</mule>
```

**Note:** you only specify the port if you don't want to use the default of 1099.

The default agent does a lot of useful plumbing for JMX but at the expense of defaulting many parameters. If you need to customize some subsystems, you could either:

- Subclass `DefaultJmxSupportAgent` and override the corresponding `createXXX()` factory methods.
- Disassemble the services provided by this support agent into separate agents and configure them individually.

## Configuring the JMX Agent

The JMX agent enables the configuration of a local or remote JMX connection to Mule and registers Mule services with the MBean server. You

can then use JMX to view the configuration state of the Mule Manager, stop and start the Mule instance, stop and start services, stop/start/resume service components, and query event processing and endpoint routing statistics on individual services or the whole server instance.

You configure the JMX agent using the `<jmx-server>` element. You can set the following properties on the agent.

cache: Unexpected program error: java.lang.NullPointerException

## Jmx Server

### Attributes of `<jmx-server...>`

Name	Type	Required	Default	Description
server-ref	string	no		The mBean server to use.
locateServer	boolean	no	true	Whether the agent should try locating an MBeanServer instance before creating one.
createServer	boolean	no	false	Whether the agent should create an MBean server if one couldn't be found or locateServer was set to false.
createRmiRegistry	boolean	no	true	Whether the agent should try locating an RmiRegistry instance before creating one. Unless there is a RmiRegistry explicitly created on the port defined by the connector-server URI, this must be set to true which is the default
enableStatistics	boolean	no	true	Whether statistics reporting is enabled for the Mule instance.

### Child Elements of `<jmx-server...>`

Name	Cardinality	Description
connector-server	0..1	Configures the remote JMX connector server by specifying the URL and whether to rebind.
credentials	0..1	A map of username/password entries used to authenticate remote JMX access. If not specified, remote access is not restricted.

For example:

```
<management:jmx-server >
  <management:connector-server url="service:jmx:rmi:///jndi/rmi://localhost:1099/server" rebind=
  "false" />
  <management:credentials>
    <spring:entry key="jsmith" value="foo" />
    <spring:entry key="dthomas" value="bar" />
  </management:credentials>
</management:jmx-server>
```

Note that the JMX domain for the Mule server is taken from the Mule server ID. To set the server ID, you set the `-M-Dmule.serverId=YOUR_MULE_SERVER_ID` system property at the command line, or set it programmatically by calling `org.mule.config.DefaultMuleConfiguration.setId()`. You can also set it in your `web.xml` file as follows:

```
<context-param>
  <param-name>mule.serverId</param-name>
  <param-value>MyServer</param-value>
</context-param>
```

## Remote Management

You can configure the Mule JMX subsystem for remote management with third-party tools like MC4J. Mule provides an RMI registry agent, which binds to an existing RMI registry or creates a new one on a defined URI.

You configure the RMI registry agent using the `<rmi-server>` element. This element has two attributes: `serverUri`, which you set to the URI of the RMI server (the default is `rmi://localhost:1099`), and `createRegistry`, which you set to true if you want to create a new registry instead of binding to an existing one.

For example:

```
<management:rmi-server serverUri="rmi://myServer.com:1099" createRegistry="true" />
```

## JMX Notifications Agent

The `<jmx-notifications>` element configures the JMX notifications agent, which sends JMX server notifications. This element takes the following attributes:

Attribute	Description
ignoreManagerNotifications	Whether to ignore notifications for state changes on the Mule manager such as initializing, starting, and stopping.
ignoreModelNotifications	Whether to ignore notifications for state changes on models such as models initializing, starting, and stopping or components being registered or unregistered.
ignoreComponentNotifications	Whether to ignore notifications for state changes on components such as when a component is started, stopped, paused, or resumed.
ignoreConnectionNotifications	Whether to ignore notifications when a connector attempts to connect to its underlying resource. Notifications are fired when a connection is made, released, or the connection attempt fails.
ignoreSecurityNotifications	Whether to ignore notifications about security.
ignoreManagementNotifications	Whether to ignore notifications for when a request is denied security access.
ignoreCustomNotifications	Whether to ignore notifications fired by objects to custom notification listeners.
ignoreAdminNotifications	Whether to ignore administrative notifications about requests being received by the Mule Admin agent. These are usually triggered by MuleClient calls using the RemoteDispatcher, which proxies calls to a remote server.
ignoreMessageNotifications	Whether to ignore message notifications. These notifications are fired when an event is sent or received in the system. They are very good for tracing, but they create a performance impact, so they should only be used during testing.

For example:

```
<management:jmx-notifications ignoreAdminNotifications="true" ignoreMessageNotifications="true" />
```

## Endpoint Notifications Publisher Agent

This agent routes server notifications to a specified endpoint URI. You configure it using the `<publish-notifications>` element and specify the endpoint using the `endpointAddress` attribute. For example:

```
<management:publish-notifications endpointAddress="vm://myService" />
```

## Log4J Agent

The `Log4j` agent exposes the configuration of the Log4J instance used by Mule for JMX management. You enable the Log4J agent using the `<jmx-log4j>` element. It does not take any additional properties.

For example:

```
<management:jmx-log4j/>
```

## Log4J Notifications Agent

The Log4J notifications agent logs server notifications to a file or console using Log4J. You configure this agent using the `<log4j-notifications>` element. It takes the same attributes as the JMX notifications agent plus two additional attributes: `logName`, a name used to identify this log, and `logConfigFile`, the name of the file where you want to output the log messages.

The Log4J notifications agent also takes the <level-mapping> child element, which takes one or more pairs of severity/eventId attributes. The severity attribute specifies the severity level of the notifications you want to log for the corresponding event ID. The severity level can be DEBUG, INFO, WARN, ERROR, or FATAL. The eventId attribute specifies the type of event to log. The event ID is the notification type plus the action, such as ModelNotification.stop.

For example:

```
<management:log4j-notifications logName="myMuleLog" logConfigFile="mule-log.txt">
  <management:level-mapping eventId="ModelNotification.stop" severity="WARN" />
</management:log4j-notifications>
```

## **Chainsaw Notifications Agent**

The Chainsaw notifications agent logs server notifications to a **Chainsaw console**. You configure this agent using the <chainsaw-notifications> element. It takes the same attributes as the JMX notifications agent plus two additional attributes: chainsawHost and {chainsawPort}, which specify the host name and port of the Chainsaw console.

The Chainsaw notifications agent also takes the <level-mapping> child element, which takes one or more pairs of severity/eventId attributes. The severity attribute specifies the severity level of the notifications you want to send to the Chainsaw console for the corresponding event ID. The severity level can be DEBUG, INFO, WARN, ERROR, or FATAL. The eventId attribute specifies the type of event to send to the Chainsaw console. The event ID is the notification type plus the action, such as ModelNotification.stop.

For example:

```
<management:chainsaw-notifications chainsawHost="localhost" chainsawPort="20202">
  <management:level-mapping eventId="ModelNotification.stop" severity="WARN" />
</management:chainsaw-notifications>
```

## **MX4J Adapter**

**MX4J** is an open source implementation of the JMX technology. The MX4J agent for Mule configures an MX4J HTTP adapter to provide JMX management, statistics, and configuration viewing of a Mule instance. You configure the MX4J agent using the <jmx-mx4j-adaptor> element.

cache: Unexpected program error: java.lang.NullPointerException

### **Jmx Mx4j Adaptor**

#### **Attributes of <jmx-mx4j-adaptor...>**

Name	Type	Required	Default	Description
jmxAdaptorUrl	string	no		The URL of the JMX web console. The default is <a href="http://localhost:9999">http://localhost:9999</a> .
login	string	no		The login name for accessing the JMX web console.
password	string	no		The password for accessing the JMX web console.
authenticationMethod	none/basic/digest	no	basic	The type of authentication to perform when the login and password are set: basic (the default), digest, or none.
cacheXsl	string	no	true	Indicates whether to cache the transformation objects, which speeds-up the process. It is usually set to true, but you can set it to false for easier testing.
xslFilePath	string	no		Specifies the path of the XSL files used to customize the adaptor's stylesheet. If you specify a directory, it assumes that XSL files are located in the directory. If you specify a .jar or .zip file, it assumes that the files are located inside. Specifying a file system is especially useful for testing.
pathInJar	string	no		If the xslFilePath is a JAR file, specifies the directory in the JAR where the XSL files are located.

#### **Child Elements of <jmx-mx4j-adaptor...>**

Name	Cardinality	Description
socketFactoryProperties	0..1	A map containing properties for SSL server socket factory configuration. If this element contains at least one property, the agent will switch to HTTPS connections. These properties will be delegated as is to the agent's HTTP/S adaptor. For a list of available properties, see the <a href="#">MX4J API documentation</a> .

For example:

```
<management:jmx-mx4j-adaptor jmxAdaptorUrl="https://myjmxserver.com:9999">
  <management:socketFactoryProperties>
    <spring:entry key="keystore" value="/path/to/keystore" />
    <spring:entry key="storepass" value="storepwd" />
  </management:socketFactoryProperties>
</management:jmx-mx4j-adaptor>
```

For security's sake, the management console is accessible from the localhost only. To loosen this restriction, change "localhost" to "0.0.0.0", which allows access from any computer on the LAN. For more information, see the [MX4J documentation](#).

## MX4J Security

You can protect the JMX web console with a user name and password. If the `login` property has been specified, the authentication scheme is applied.

In addition to protecting the console, you can protect the in-transit data using SSL. If the `socketFactoryProperties` element contains at least one property, the agent switches to HTTPS connections. If this element is omitted from the configuration, the agent will always use HTTP, even if you specify https:// in the `jmxAdaptorUrl` property.

## Viewing Statistics

Mule traps many different statistics about the running state of a server and number of events processed. You can view the Mule statistics report in the JMX Management Console by pointing your browser to <http://localhost:9999/> and then clicking on any JMX domain name (except for JMImplementation), or go to the Statistics tab and query the JMX domain for statistics from there.

Component Name	GreeterUMO	ChitChatUMO	muleManagerComponent
Component Pool Max Size	5/5	5/5	5/5
Component Pool Size	0	0	0
Thread Pool Size	8	8	8
Current Queue Size	0	0	0
Max Queue Size	0	0	0
Avg Queue Size	0	0	0
Sync Events Received	50	50	0
Async Events Received	0	0	0
Total Events Received	50	50	0
Sync Events Sent	50	50	0
Async Events Sent	0	0	0
ReplyTo Events Sent	0	0	0
Total Events Sent	50	50	0
Executed Events	50	50	0
Execution Messages	0	0	0
Fatal Messages	0	0	0
Min Execution Time	16	0	0
Max Execution Time	62	47	0
Avg Execution Time	3	2	0
Total Execution Time	170	124	0
<b>In Router Statistics</b>			
Total Received	50	50	0
Total Routed	50	50	0
Not Routed	0	0	0
Caught Events	0	0	0
<b>By Provider</b>			
	httpEndpoint: 50	endpoint.vm.chitchatter: 50	
<b>Out Router Statistics</b>			
Total Received	50	0	0
Total Routed	50	0	0
Not Routed	0	0	0
Caught Events	0	0	0
<b>By Provider</b>			
	endpoint.vm.chitchatter: 50		
Sample Period	1394628	1394906	1394750

## YourKit Profiler

This agent exposes the YourKit profiler to JMX to provide CPU and memory profiling. To use this agent, you must configure the `<yourkit-profiler>` element as shown below, and you must install and run the Profiler as described in [Profiling Mule](#).

```
<management:yourkit-profiler />
```

Your Rating:  Results:  0 rates

## About Configuration Builders

### About Configuration Builders

[ [SpringXmlConfigurationBuilder](#) ] [ [ScriptConfigurationBuilder](#) ] [ [Custom Configuration Builders](#) ] [ [Specifying the Configuration Builder](#) ]

The configuration builder is responsible for creating the configuration that's used at run time from the configuration files you provide. Mule ESB provides two standard configuration builders, or you can create your own.

#### SpringXmlConfigurationBuilder

The default configuration builder used to configure Mule is the SpringXmlConfigurationBuilder. This configuration builder uses Spring to configure Mule based on one or more XML files leveraging custom Mule namespaces. For more information, see [About the XML Configuration File](#).

#### ScriptConfigurationBuilder

This configuration builder allows a JSR-223 compliant script engine such as Groovy or Jython to configure Mule. For more information, see [Scripting Module Reference](#).

#### Custom Configuration Builders

You can easily create your own custom configuration builder by implementing the `ConfigurationBuilder` interface, which has a method `configure`. Typically, you call `configure(MuleContext muleContext.getRegistry())` to get access to Mule's internal registry, which contains the configuration information, and to register services and other elements.

In most cases, you will want to inherit from the class `AbstractResourceConfigurationBuilder`, which will make any configuration files specified on the command line available in an instance variable called `configResources`.

#### Specifying the Configuration Builder

`AutoConfigurationBuilder` is the default configuration builder. If you want to use another configuration builder, you can specify it in the deployment descriptor of an application.

You can also specify the configuration builder as a parameter to the `MuleContextFactory` when starting Mule programmatically:

```
MuleContext context = new DefaultMuleContextFactory().createMuleContext(new
ScriptConfigurationBuilder("mule-config.groovy"));
context.start();
```

Your Rating:  Results:  0 rates

## Tuning Performance

### Tuning Performance

[ [About Threads in Mule](#) ] [ [About Thread Pools and Threading Profiles](#) ] [ [About Pooling Profiles](#) ] [ [Calculating Threads](#) ] [ [Additional Performance Tuning Tips](#) ] [ [Threading Profile Configuration Reference](#) ]

A Mule ESB application is a collaboration of a set of services. Messages are processed by services in three stages:

1. Connector receiving stage
2. Service component processing stage
3. Connector dispatching stage

Tuning performance in Mule involves analyzing and improving these three stages for each service. You can start by applying the same tuning approach to all services and then further customize the tuning for each service as needed.

## About Threads in Mule

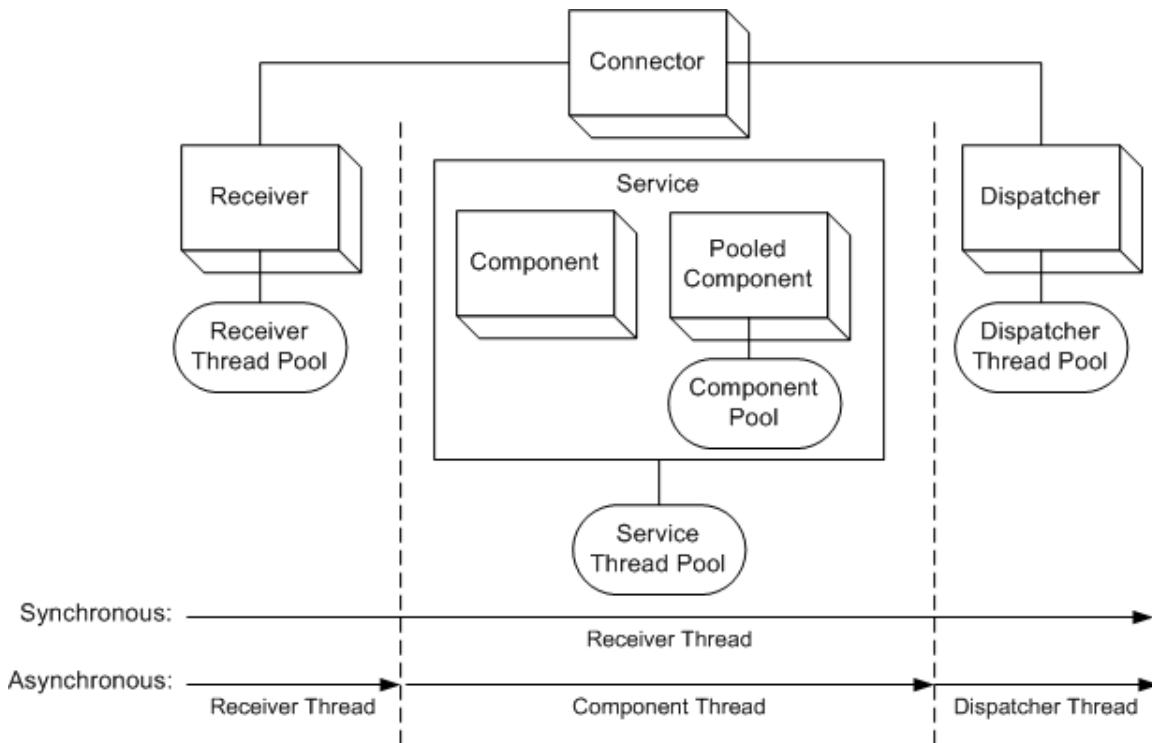
Each request that comes into Mule is processed on its own thread. A connector's receiver has a *thread pool* with a certain number of threads available to process requests on the inbound endpoints that use that connector.

Keep in mind that Mule can send messages asynchronously or synchronously. By default, messages are asynchronous, which is the "fire-and-forget" style where a message is sent with no response. If you need a response back, you must configure the Mule service as synchronous. When a service is configured as asynchronous, a connector receiver and dispatcher are used, whereas a synchronous service uses only the connector receiver.

If you are using **synchronous** processing, the same thread will be used to carry the message all the way through Mule.

If you are doing **asynchronous** processing, the receiver thread is used only to carry the message to the component, at which point the message is transferred to a *component thread*, and the receiver thread is released back into the receiver thread pool so it can carry another message. After the component has finished processing an asynchronous message, it is transferred to a *dispatcher thread* and is sent on its way. Therefore, the receiver, component, and dispatcher all have separate threads that are in use during **asynchronous** processing, whereas only the receiver thread is in use for **synchronous** processing.

The following diagram illustrates these threads.



## About Thread Pools and Threading Profiles

A *thread pool* is a collection of available threads. There is a separate thread pool for each receiver, service (shared by all the components in that service), and dispatcher.

The *threading profile* specifies how the thread pools behave in Mule. You specify a separate threading profile for each receiver thread pool, service thread pool, and dispatcher thread pool. The most important setting of each is `maxThreadsActive`, which specifies how many threads are in the thread pool. The [Calculating Threads](#) section below describes methodologies for determining what this setting should be, whereas the [Threading Profile Configuration Reference](#) section describes the options you can set for each threading profile.

## About Pooling Profiles

Unlike singleton components, pooled components (see [{{PooledJavaComponent}}](#)) each have a *component pool*, which contains multiple instances of the component to handle simultaneous incoming requests. A service's *pooling profile* configures its component pool. The most important setting is `maxActive`, which specifies the maximum number of instances of the component that Mule will create to handle simultaneous requests. Note that this number should be the same as the `maxThreadsActive` setting on the receiver thread pool, so that you have enough component instances available to handle the threads. You can use Mule HQ to monitor your component pools and see the maximum number of components you've used from the pool to help you tune the number of components and threads.

For more information on configuring the pooling profile, see [Pooling Profile Configuration Reference](#) below.

## Calculating Threads

To calculate the number of threads to set, you must take the following factors into consideration.

- **Concurrent User Requests:** In general, the number of concurrent user requests is the total number of requests to be processed simultaneously at any given time by the Mule server. For a **service**, concurrent user requests is the number of requests the service's inbound endpoint can process simultaneously. Concurrent user requests at the **connector** level is the total concurrent requests of all services that share the same connector. Typically, you get the total concurrent user requests from the business requirements.
- **Processing Time:** Processing time is the average time Mule takes to process a user request from the time a connector receiver starts the execution until it finishes and then sends the response to the outbound endpoint by connector dispatcher or back to the caller by the connector receiver after a round trip. Typically, you determine the processing time from the unit tests.
- **Response Time:** If a service runs in **synchronous** mode, the response time is the actual amount of time the end user is required to wait for the response to come back. If the service runs in **asynchronous** mode, it is the total time from when a request arrives in Mule until it is dispatched out of the service by the outbound dispatcher. In a thread pooling environment, when a request arrives, there is no guarantee that a thread will be immediately available. In this case, the request is put into an internal thread pool *work queue* to wait for the next available thread. Therefore, the response time is a function of the following:  
Response time = average of thread pool waiting time in work queue + average of processing time  
Your business requirements will dictate the actual response time required from the application.
- **Timeout Time:** If your business requirements dictate a maximum time to wait for a response before timing out, it will be an important factor in your calculations below.

After you have determined these requirements, you can calculate the adjustments you need to make to `maxThreadsActive` and `maxBufferSize` for the service and the receiver thread pool. In general, the formula is:

Concurrent user requests = `maxThreadsActive` + `maxBufferSize`

where `maxThreadsActive` is the number of threads that run concurrently and `maxBufferSize` is the number of requests that can wait in the queue for threads to be released.

## Calculating the Service Threads

Even if you will be performing synchronous messaging only, you must calculate the service threads so that you can correctly calculate the receiver threads. This section describes how to calculate the service threads.

Your business requirements dictate how many threads each service must be able to process concurrently. For example, one service might need to be able to process 50 requests at a time, while another might need to process 40 at a time. Typically, you use this requirement to set the `maxThreadsActive` attribute on the service (`maxThreadsActive="40"`).

If you have requirements for timeout settings for synchronous processing, you must do some additional calculations for each service.

1. Run synchronous test cases to determine the response time.
2. Subtract the response time from the timeout time dictated by your business requirements. This is your *maximum wait time* (maximum wait time = timeout time - response time).
3. Divide the maximum wait time by the response time to get the number of *batches* that will be run sequentially to complete all concurrent requests within the maximum wait time ( $batches = \text{maximum wait time} / \text{response time}$ ). Requests wait in the queue until the first batch is finished, and then the first batch's threads are released and used by the next batch.
4. Divide the concurrent user requests by the number of batches to get the thread size for the service's `maxThreadsActive` setting (that is, `maxThreadsActive = concurrent user requests / processing batches`). This is the total number of threads that can be run simultaneously for this service.
5. Set `maxBufferSize` to the concurrent user requests minus the `maxThreadsActive` setting (that is, `maxBufferSize = concurrent user requests - maxThreadsActive`). This is the number of requests that can wait in the queue for threads to become available.

For example, assume a service must have the ability to process 200 concurrent user requests, your timeout setting is 10 seconds, and the response time is 2 seconds, making your maximum wait time 8 seconds (10 seconds timeout minus 2 seconds response time). Divide the maximum wait time (8 seconds) by the response time (2 seconds) to get the number of batches (4). Finally, divide the concurrent user requests requirement (200 requests) by the batches (4) to get the `maxThreadsActive` setting (50) for the service. Subtract this number (50) from the concurrent user requests (200) to get your `maxBufferSize` (150).

In summary, the formulas for synchronous processing with timeout restrictions are:

- Maximum wait time = timeout time - response time
- Batches = maximum wait time / response time
- maxThreadsActive = concurrent user requests / batches
- maxBufferSize = concurrent user requests - maxThreadsActive

## Calculating the Receiver Threads

A connector's receiver is shared by all services that specify the same connector on their inbound endpoint. The previous section described how to calculate the `maxThreadsActive` attribute for each service. To calculate the `maxThreadsActive` setting for the receiver, that is, how many threads you should assign to a connector's receiver thread pool, sum the `maxThreadsActive` setting for each service that uses that connector on their inbound endpoints:

```
maxThreadsActive = (service 1 maxThreadsActive, service 2 maxThreadsActive...service n maxThreadsActive)
```

For example, if you have three components whose inbound endpoints use the VM connector, and your business requirements dictate that two of the services should handle 50 requests at a time and the third service should handle 40 requests at a time, set `maxThreadsActive` to 140 in the receiver threading profile for the VM connector.

## Calculating the Dispatcher Threads

Dispatcher threads are used only for asynchronous processing. Typically, set `maxThreadsActive` for the dispatcher to the sum of `maxThreadsActive` values for all services that use that dispatcher.

## Other Considerations

You can trade off queue sizes and maximum pool sizes. Using large queues and small pools minimizes CPU usage, OS resources, and context-switching overhead, but it can lead to artificially low throughput. If tasks frequently block (for example, if they are I/O bound), a system may be able to schedule time for more threads than you otherwise allow. Use of small queues generally requires larger pool sizes, which keeps CPUs busier but may encounter unacceptable scheduling overhead, which also decreases throughput.

## Additional Performance Tuning Tips

- In the `log4j.properties` file in your `conf` directory, set up logging to a file instead of the console, which will bypass the wrapper logging and speed up performance. To do this, create a new file appender (`org.apache.log4j.FileAppender`), specify the file and optionally the layout and other settings, and then change "console" to the file appender. For example:

```
log4j.rootCategory=INFO, mulelogfile

log4j.appender.mulelogfile=org.apache.log4j.FileAppender
log4j.appender.mulelogfile.layout=org.apache.log4j.PatternLayout
log4j.appender.mulelogfile.layout.ConversionPattern=%-22d{dd/MMM/yyyy HH:mm:ss} - %m%n
log4j.appender.mulelogfile.file=custommule.log
```

- (Coming in Mule 2.2.2) If you have a very large number of services in the same Mule instance, if you have components that take more than a couple seconds to process, or if you are processing very large payloads or are using slower transports, you should increase the `shutdownTimeout` attribute (see [Global Settings Configuration Reference](#)) to enable graceful shutdown.
- If polling is enabled for a connector, one thread will be in use by polling, so you should increment your `maxThreadsActive` setting by one. Polling is available on connectors such as File, FTP, and STDIO that extend `AbstractPollingMessageReceiver`.
- If you are using VM to pass a message between components, you can typically reduce the total number of threads because VM is so fast.
- If you are processing very heavy loads, or if your endpoints have different simultaneous request requirements (for example, one endpoint requires the ability to process 20 simultaneous requests but another endpoint using the same connector requires 50), you might want to split up the connector so that you have one connector per endpoint.



Thread pools in Mule use the JDK 1.4-compatible `util.concurrent` backport library, so all the variables defined on `org.mule.config.ThreadingProfile` are synonymous with `ThreadPoolExecutor`.

## Threading Profile Configuration Reference

Following are the elements you configure for threading profiles. You can create a threading profile at the following levels:

- Configuration level (`<configuration>`)
- Connector level (`<connector>`)
- Service level (`<service>`)

The rest of this section describes the elements and attributes you can set at each of these levels.

## Configuration Level

---

The `<default-threading-profile>`, `<default-receiver-threading-profile>`, `<default-dispatcher-threading-profile>`, and `<default-service-threading-profile>` elements can be set in the `<configuration>` element to set default threading profiles for all connectors and services. Following are details on each of these elements.

cache: Unexpected program error: java.lang.NullPointerException

### Default Threading Profile

The default threading profile, used by components and by endpoints for dispatching and receiving if no more specific configuration is given.

#### Attributes of `<default-threading-profile...>`

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

### Default Receiver Threading Profile

The default receiving threading profile, which modifies the default-threading-profile values and is used by endpoints for receiving messages. This can also be configured on connectors, in which case the connector configuration is used instead of this default.

#### Attributes of `<default-receiver-threading-profile...>`

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.

maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Default Service Threading Profile

The default service threading profile, which modifies the default-threading-profile and is used by services for processing messages. This can also be configured on models or services, in which case these configurations will be used instead of this default.

### Attributes of <default-service-threading-profile...>

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.

doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Default Dispatcher Threading Profile

The default dispatching threading profile, which modifies the default-threading-profile values and is used by endpoints for dispatching messages. This can also be configured on connectors, in which case the connector configuration is used instead of this default.

### Attributes of <default-dispatcher-threading-profile...>

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Connector Level

---

The <receiver-threading-profile> and <dispatcher-threading-profile> elements can be set in the <connector> element to configure the threading profiles for that connector. Following are details on each of these elements.

cache: Unexpected program error: java.lang.NullPointerException

### Receiver Threading Profile

The threading profile to use when a connector receives messages.

### Attributes of <receiver-threading-profile...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Dispatcher Threading Profile

The threading profile to use when a connector dispatches messages.

### Attributes of <dispatcher-threading-profile...>

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).

threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Service Level

---

The `<threading-profile>` element can be set in the `<service>` element to configure the threading profile for all components in that service. Following are details on this element.

cache: Unexpected program error: java.lang.NullPointerException

### Threading Profile

The threading profile to use for the service.

#### Attributes of `<threading-profile...>`

Name	Type	Required	Default	Description
maxThreadsActive	integer	no		The maximum number of threads that will be used.
maxThreadsIdle	integer	no		The maximum number of idle or inactive threads that can be in the pool before they are destroyed.
threadTTL	integer	no		Determines how long an inactive thread is kept in the pool before being discarded.
poolExhaustedAction	WAIT/DISCARD/DISCARD_OLDEST/ABORT/RUN	no		When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks. Possible values are: WAIT (wait until a thread becomes available; don't use this value if the minimum number of threads is zero, in which case a thread may never become available), DISCARD (throw away the current request and return), DISCARD_OLDEST (throw away the oldest request and return), ABORT (throw a RuntimeException), and RUN (the default; the thread making the execute request runs the task itself, which helps guard against lockup).
threadWaitTimeout	integer	no		How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, it will wait indefinitely.
doThreading	boolean	no	true	Whether threading should be used (default is true).
maxBufferSize	integer	no		Determines how many requests are queued when the pool is at maximum usage capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.

## Pooling Profile Configuration Reference

Each pooled component has its own pooling profile. You configure the pooling profile using the `<pooling-profile>` element on the `<pooled-component>` element.

cache: Unexpected program error: java.lang.NullPointerException

## Pooling profile

### Attributes of <pooling-profile...>

Name	Type	Required	Default
maxActive	string	no	
maxIdle	string	no	
initialisationPolicy	INITIALISE_NONE/INITIALISE_ONE/INITIALISE_ALL	no	INITIALISE_ONE
exhaustedAction	WHEN_EXHAUSTED_GROW/WHEN_EXHAUSTED_WAIT/WHEN_EXHAUSTED_FAIL	no	WHEN_EXHAUSTED
maxWait	string	no	

### Child Elements of <pooling-profile...>

Name	Cardinality	Description
------	-------------	-------------

Your Rating:  5 stars

Results:  1 rates

## Configuring Reconnection Strategies

### Configuring Reconnection Strategies

[ Configuring Reconnection Strategies with the mule-ee.xsd schema  ] [ Reconnect ] [ Reconnect forever ] [ Reconnect custom strategy ] [ Reconnect notifier ] [ Reconnect custom notifier ] [ Creating a Custom Reconnection Strategy ] [ Configuring Reconnection Strategies with the Spring Schema ]



Reconnection Strategies in Mule 2.x were called "Retry Policies", because the underlying classes are a generic framework for retrying the same operation over and over again. However, this name led to a lot of confusion because users were expecting a message to get re-delivered or a message exchange to be re-executed. Therefore in Mule 3.x, we have gone back to the name "Reconnection Strategies" to avoid any misunderstandings.

Reconnection Strategies configure how a connector behaves when its connection fails. Different policies can be used to control how a reconnection is made based on type of exception, number and/or frequency of reconnection attempts, notifications, and more.

For example, assume you are using the FTP transport and have a polling receiver set to poll every 1000 ms, and the connection is down. The polling receiver will try to connect (and fail) once every 1000 ms, using up resources and filling up your log files. It will continue to fail indefinitely until you manually stop the process. Additionally, it leaves the system in an unstable state, because the connector is theoretically "connected" even though it's constantly failing. With a reconnection strategy, you can better control the behavior when a connection fails, such as setting the connector to re-attempt the connection once every 15 minutes and to give up after 30 attempts. You can also send an automatic notification to your IT administrator when the reconnection strategy goes into effect. You can even define a strategy that only re-attempts during business hours, which is useful if your server is frequently shut down for maintenance at night, for example.

When using the FTP transport, and you are using synchronous inbound and outbound endpoints, all inbound messages would fail if the connection went down and you weren't using reconnection strategies. With reconnection strategies, you will lose the first message that fails, since FTP is not transactional, but once a reconnection strategy goes into effect, no further messages will be accepted by the inbound endpoint until the connection is re-established.

If you are using the Enterprise Edition of Mule ESB, there are several standard reconnection strategies available that you can configure using the mule-ee.xsd schema. If you are using the Community Edition of Mule ESB, you must [create your own strategies](#) and [configure them using standard Spring syntax](#) rather than the mule-ee.xsd schema.

### Configuring Reconnection Strategies with the mule-ee.xsd schema

This section describes the reconnection strategies you can configure directly in your XML by including the mule-ee.xsd schema. You import the schema as follows:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:ee="http://www.mulesoft.org/schema/mule/ee/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/ee/core
          http://www.mulesoft.org/schema/mule/ee/core/3.1/mule-ee.xsd">
```

cache: Unexpected program error: java.lang.NullPointerException

### Reconnect

A reconnection strategy that allows the user to configure how many times a reconnection should be attempted and how long to wait between attempts.

#### Attributes of <reconnect...>

Name	Type	Required	Default	Description
blocking	boolean	no	true	If false, the reconnection strategy will run in a separate, non-blocking thread
asynchronous	boolean	no		@deprecated Use <blocking> instead. Whether the retry policy should run in a separate, non-blocking thread
frequency	long	no	2000	How often (in ms) to reconnect
count	integer	no	2	How many reconnection attempts to make

#### Child Elements of <reconnect....>

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<jms:activemq-connector name="AMQConnector">
  <ee:reconnect count="5" frequency="1000" />
</jms:activemq-connector>
```

cache: Unexpected program error: java.lang.NullPointerException

### Reconnect forever

A reconnection strategy that retries an infinite number of times at the specified frequency.

#### Attributes of <reconnect-forever....>

Name	Type	Required	Default	Description
blocking	boolean	no	true	If false, the reconnection strategy will run in a separate, non-blocking thread
asynchronous	boolean	no		@deprecated Use <blocking> instead. Whether the retry policy should run in a separate, non-blocking thread
frequency	long	no	2000	How often (in ms) to reconnect

#### Child Elements of <reconnect-forever....>

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<jms:activemq-connector name="AMQConnector">
  <ee:reconnect-forever frequency="5000" />
</jms:activemq-connector>
```

cache: Unexpected program error: java.lang.NullPointerException

### Reconnect custom strategy

A user-defined reconnection strategy.

#### Attributes of <reconnect-custom-strategy....>

Name	Type	Required	Default	Description
blocking	boolean	no	true	If false, the reconnection strategy will run in a separate, non-blocking thread
asynchronous	boolean	no		@deprecated Use <blocking> instead. Whether the retry policy should run in a separate, non-blocking thread

class	class name	yes		A class that implements the RetryPolicyTemplate interface.
-------	------------	-----	--	--

### Child Elements of <reconnect-custom-strategy...>

Name	Cardinality	Description
spring:property	0..*	

For example:

```
<jms:activemq-connector name="AMQConnector">
    <ee:reconnect-custom-strategy class="org.mule.retry.test.TestRetryPolicyTemplate">
        <spring:property name="fooBar" value="true"/>
        <spring:property name="revolutions" value="500"/>
    </ee:reconnect-custom-strategy>
</jms:activemq-connector>
```

### Non-Blocking Reconnection

By default, a reconnection strategy will block until it is able to connect/reconnect. Enabling non-blocking reconnection means the application does not need to wait for all endpoints to connect before it can start up, and if a connection is lost, the reconnection will happen in a separate thread from the application thread. Note that such behavior may or may not be desirable depending on your application.

Any reconnection strategy can be made non-blocking by simply setting the attribute `blocking="false"`. For example:

```
<jms:activemq-connector name="AMQConnector">
    <ee:reconnect frequency="3000" blocking="false" />
</jms:activemq-connector>
```

If not specified, the `blocking` attribute defaults to "true".



In Mule 2.x, the attribute `asynchronous` was used for this purpose. The new attribute `blocking` is the inverse of `asynchronous` so a Mule 2.x config which had `asynchronous="true"` should be changed to `blocking="false"` for Mule 3.x.

### Transactions

If [transactions](#) are properly configured, any messages being routed by Mule at the time a reconnection strategy goes into effect will not be dropped. Instead, the transaction will roll back and only commit once the transport finally reconnects successfully via the reconnection strategy.

### Reconnect Notifiers

A reconnect notifier is called for each reconnection attempt and is also configurable. You can create a custom reconnect notifier that implements the [org.mule.api.retry.RetryNotifier](#) interface.

cache: Unexpected program error: java.lang.NullPointerException

### Reconnect notifier

Fires a `ConnectionNotification` upon each reconnection attempt.

### Attributes of <reconnect-notifier...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <reconnect-notifier...>

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<jms:activemq-connector name="AMQConnector">
    <ee:reconnect>
        <ee:reconnect-notifier/>
    </ee:reconnect>
</jms:activemq-connector>
```

cache: Unexpected program error: java.lang.NullPointerException

## Reconnect custom notifier

A user-defined reconnection notifier.

### Attributes of <reconnect-custom-notifier...>

Name	Type	Required	Default	Description
class	class name	yes		A class that implements the RetryNotifier interface.

### Child Elements of <reconnect-custom-notifier...>

Name	Cardinality	Description
spring:property	0..*	

For example:

```
<jms:activemq-connector name="AMQConnector">
    <ee:reconnect>
        <ee:reconnect-custom-notifier class="org.mule.retry.test.TestRetryNotifier">
            <spring:property name="color" value="red"/>
        </ee:reconnect-custom-notifier>
    </ee:reconnect>
</jms:activemq-connector>
```

## Configuring Separate Connectors for Inbound and Outbound

The reconnection strategy for a connector is used for both inbound and outbound connections. If you require a different behavior for inbound and outbound connections, you can achieve this by configuring two connectors with the different strategies and reference them from the inbound and outbound endpoints, respectively.

## Default Reconnection Strategy

The default reconnection strategy is used for any connector that does not have reconnection explicitly configured. You can set the default strategy using the <configuration> element:

```
<configuration>
    <ee:reconnect count="3"/>
</configuration>
```

## Creating a Custom Reconnection Strategy

To create a custom reconnection strategy, you implement the interface `RetryPolicy`, where the method `PolicyStatus applyPolicy(Throwable cause)` takes some action based on the type of exception and returns `PolicyStatus` to indicate whether the policy

has been exhausted or should continue to retry. You also create a `RetryPolicyTemplate`, which is what you actually configure on the connector. The template should generally inherit from `AbstractPolicyTemplate` and have the method `RetryPolicy createRetryInstance()` return an instance of your custom `RetryPolicy`. At runtime, a new instance of the `RetryPolicy` will be created each time the policy goes into effect, thereby resetting any state information it may contain, such as counters.

For example:

```
package com.acme.retry;

public class AstronomicalRetryPolicyTemplate extends AbstractPolicyTemplate
{
    int totalPlanets;

    public RetryPolicy createRetryInstance()
    {
        return new AstronomicalRetryPolicy(totalPlanets);
    }

    protected static class AstronomicalRetryPolicy implements RetryPolicy
    {
        int totalPlanets;

        public AstronomicalRetryPolicy(int totalPlanets) { this.totalPlanets = totalPlanets; }

        public PolicyStatus applyPolicy(Throwable cause)
        {
            if (AstronomyUtils.getPlanetsAligned() == totalPlanets)
            {
                return PolicyStatus.policyExhausted(cause);
            }
            else
            {
                Thread.sleep(5000);
                return PolicyStatus.policyOk();
            }
        }
    }
}

public int getTotalPlanets() { return totalPlanets; }
public void setTotalPlanets(int totalPlanets) { this.totalPlanets = totalPlanets; }
```

## Configuring Reconnection Strategies with the Spring Schema

Because the reconnection elements are only available in the Mule Enterprise Edition schema, Mule Community users must use standard Spring syntax for configuring a custom reconnection strategy. For example:

```
<jms:activemq-connector name="AMQConnector">
    <spring:property name="retryPolicyTemplate">
        <spring:bean class="com.acme.retry.AstronomicalRetryPolicyTemplate">
            <spring:property name="totalPlanets" value="8"/>
        </spring:bean>
    </spring:property>
</jms:activemq-connector>
```

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## Streaming

### Streaming

Streaming enables efficient processing of large data objects such as files, documents, and records by streaming the data through Mule ESB rather than reading the whole thing into memory. Streaming provides the following advantages:

- Allows services to consume very large messages in an efficient way
- Message payloads are not read into memory
- Simple routing rules based on message metadata are still possible
- You can combine streaming and non-streaming endpoints

## Streaming Transports and Modules

The following transports and modules support streaming:

- CXF
- File (set the `streaming` attribute to false to access the File object directly)
- FTP
- HTTP and HTTPS
- Jetty and Jetty SSL
- Servlet
- TCP
- UDP
- VM

## Streaming Transformers and Filters

Many transformers and filters can read input streams, process the contents, and send them on. However, most of these do not process the stream in real time; instead, they read the stream, load it into memory, process it, and then send it on. Therefore, transformers and filters can become a bottleneck in your application if you regularly stream large files.

The following transformers and filters do support true streaming and process the data as streams without loading them into memory first:

- XSLT Transformer
- XmlToXMLStreamReader Transformer
- DomToOutputHandler transformer (if the incoming XML format is a SAXSource or XMLStreamReader)
- SXC filter

Your Rating: 

Results:  0 rates

## Bootstrapping the Registry

### Bootstrapping the Registry

Mule ESB provides a mechanism for adding objects to the Mule registry as soon as a module or transport is loaded. This is useful for registering shared transformers or expression evaluators.

All objects you add to the registry using this mechanism must have a default constructor. They can implement injection interfaces such as `org.mule.MuleContextAware` and lifecycle interfaces such as `org.mule.api.lifecycle.Initialisable`.

### Specifying the Objects to Bootstrap

To load an object into the Mule registry using the bootstrap mechanism, you specify the object in the properties file `registry-bootstrap.properties`, which you then store in the META-INF directory for the transport or module. For example:

```
src/main/resources/META-INF/services/org/mule/config/registry-bootstrap.properties
```

Each entry in the `registry-bootstrap.properties` file is a simple key/value pair that defines the object:

```
registry-bootstrap.properties
```

```
myobject=org.foo.MyObject
```

If this file is in a module or transport's META-INF directory, Mule will register an instance of `org.foo.MyObject` with a key of 'myobject' into the

local registry when that module or transport is loaded.

If you want to ensure that the object gets a unique ID in the local registry, you can use `object.n` for the key, where `n` is a sequential number:

#### registry-bootstrap.properties

```
object.1=org.foo.MyObject  
object.2=org.bar.MyObject
```

## Adding Transformers

When adding transformers, you can also define the `returnClass` and name of the transformer as parameters:

#### registry-bootstrap.properties

```
transformer.1=org.mule.transport.jms.transformers.JMSMessageToObject,returnClass=byte[]  
transformer.2=org.mule.transport.jms.transformers.JMSMessageToObject,returnClass=java.lang.String,  
name=JMSMessageToString  
transformer.3=org.mule.transport.jms.transformers.JMSMessageToObject,returnClass=java.util.Hashtable)
```

Note that the key used for transformers must be `transformer.n` where `n` is a sequential number.

If the transformer name is not specified, Mule will automatically generate the name as `JMSMessageToXXX` where `XXX` is the return class name, such as `JMSMessageToString`. If no `returnClass` is specified, the default value in the transformer will be used.

Your Rating:  Results:  0 rates

## Configuring Queues

### Configuring Queues

[ [SEDA Flow and Service Queues](#) ] [ [Transport Queues](#) ] [ [Queue Configuration](#) ] [ [Persistence Strategies](#) ] [ [\[Mule 3.2\] SEDA Queue Persistence](#) ]

Mule uses queues to enable asynchronous message processing. This page describes how Mule implements queues and how you configure them. Note that users of the Enterprise Edition of Mule ESB can use the management console to monitor queues and gain historical insight into the numbers of messages they have contained. For details, see [Using the Mule Management Console](#).



For details on the use of SEDA queues in flows, see [Flow Processing Strategies](#).

### SEDA Flow and Service Queues

When a message comes in to a SEDA service or a flow that uses the queued-asynchronous processing strategy, it is stored in the queue until it can be processed by a thread from the thread pool maintained by the queue.

The queue profile specifies how the queue behaves. Typically, you do not need to configure the queue profile for performance, since the default configuration is usually sufficient, that is, the queue does not turn out to be the bottleneck. (Performance is usually limited by the service component or one of the endpoints). For other reasons, you still might want to specify a maximum queue size, or to enable persistence on the queue, which is disabled by default. A persistent queue resides in some form of non-volatile storage (most commonly a disk-based file system) so that messages are not lost when Mule exits.

When a non-persistent queue is in use and the flow or service that uses it is halted either because a `stop()` call has been issued against that service or Mule has begun shutting down for whatever reason, the service processes all messages in the queue before executing the stop order.

By contrast, when a persistent queue is in use and a `stop()` command comes in, Mule immediately stops accepting new messages. It then completes processing of all messages currently being processed before it shuts down. Mule does not begin the processing of any messages still waiting in the queue, because copies of these messages are in non-volatile storage (for example, a disk file) – the messages can be processed after Mule restarts and the SEDA service is once again available.



If a non-persistent queue is in use, and the SEDA service is paused when `stop()` is called, all messages pending processing in the queue are lost.

You can ensure that all message are saved when Mule exits by doing the following:

- Implement persistent SEDA queues.
- Specify transports that support persistence for all of Mule's internal message channels.

To further insure that no messages are lost, you can design your Mule deployment so that all transactions employ synchronous endpoints. By definition synchronous endpoints do not use queues. Note that synchronous implementations, in general, cannot match the performance of queued solutions.

## Transport Queues

The VM Transport facilitates asynchronous message delivery by using queues internal to Mule. The [VM Transport Reference](#) explains how to enable or disable queuing. It also covers the ways you can configure a VM queue.

## Queue Configuration

See [SEDA Queue Persistence](#) for details.

## Persistence Strategies

**[Mule 3.2]** The persistence strategies approach described in this section has been replaced in Mule 3.2. See [SEDA Queue Persistence](#) for further information.

By default, Mule use two persistence strategies:

- `MemoryPersistenceStrategy`, which is a volatile, in-memory persistence strategy.
- `FilePersistenceStrategy`, which uses a file store to persist messages to a (non-volatile) disk, and therefore maintains messages even if Mule is restarted.

The following table lists which persistence strategy is used when you set the `persistent` attribute in the queue-profile:

When you set <code>persistent</code> to ...	This is the result:
<code>true</code>	<code>FilePersistenceStrategy</code>
<code>false</code>	<code>MemoryPersistenceStrategy</code>

Currently, you cannot configure any other persistence strategies as part of the typed Mule XML configuration. However, if you need a different persistence strategy, such as to persist to a database rather than a disk, you can override the defaults in `mule-default-config.xml`. You achieve this by redefining the `_muleQueueManager` bean in your own configuration file. Such custom persistence strategies must implement `QueuePersistenceStrategy`.

## [Mule 3.2] SEDA Queue Persistence

Storage for Mule's internal SEDA queues is configured in a new way. The `QueuePersistenceStrategy` interface and its implementations have been replaced. Mule's internal SEDA queues now use an `ObjectStore` for persisting messages. This means that the persistence strategies described in [Persistence Strategies](#) no longer apply in Mule 3.2.

In addition, the attributes of `<queue-profile>` have changed. The `maxOutstandingMessage` attribute remains. However, the `persistent` attribute has been replaced by a child element, which identifies the object store for the SEDA queue. The queue store defaults to the in-memory store. For a cluster, the queue store is a shared data grid.

In most cases, the default object store performs well for SEDA queue message persistence. In these cases, no configuration is necessary. However, you can specify which object store to use when persisting messages for individual services by specifying an appropriate child element in the `<queue-profile>` element.

See [Mule Object Stores](#) for additional details.

Your Rating:

Results: 0 rates

## Internationalizing Strings

## Internationalizing Strings

[ Internationalized Messages ] [ Exceptions ] [ Using Custom Message Bundles ] [ Creating Message Instances from your Code ]

Mule ESB supports internationalization of exception messages and any other type of string message. Mule has support for the following Languages:

- English
- Japanese

### Internationalized Messages

Mule uses the Java [ResourceBundle](#) class to load messages from properties files on the classpath based on the current system's locale. Mule provides a full set of messages in English and [Japanese](#) only, but there may be additional languages provided in the future.

Mule's internationalized messages are represented by the [org.mule.config.i18n.Message](#) class. Instances are constructed with a message ID and zero or more message parameters. You can see a list of core messages that Mule provides in [META-INF/service/org/mule/i18n/core-messages.properties](#).

You never create instances of [Message](#) directly. Instead, you use subclasses of [MessageFactory](#). The messages for Mule's core project are accessible through the [org.mule.config.i18n.CoreMessages](#) class.

Each of Mule's modules and transports has such a messages class. Its name is equal to the module with *Messages* appended. For example, for the JMS transport you will use [JmsMessages](#) to retrieve messages.

The dedicated messages class per module/transport has the following advantages:

- Encapsulation of the message code
- Client code is not cluttered with [Message](#) constructors
- Client code has typesafe access to its messages
- Client code is not cluttered with formatting of message parameters. Instead, you handle this in the module-specific messages class

### Exceptions

[MuleException](#) is the base class of all Mule checked exceptions and can only be constructed using internationalized messages. To create a message for an exception, you use [MuleExtension](#) as follows:

```
MuleException e = new MuleException(CoreMessages.failedToGetPooledObject());
throw e;
```

### Using Custom Message Bundles

When writing Mule extensions or applications that will use the Mule internationalization class, you can supply custom message bundles containing messages specific to your extension or application. You create a resource bundle as follows:

```
1=Error message one
2=Error message with 2 parameters; param {0} and param {1}
...
```

where the number is the message ID and the actual message comes after. Note that message parameters are specified using '{0}' notation, which is standard when using the Java [MessageFormat](#) class.

The file should be named [x-messages.properties](#) where *x* is the identifying name for this bundle. You must place this file either in your JAR file under [META-INF/services/org/mule/i18n/x-messages.properties](#) or any other location on the classpath.

To access the messages of your own resource bundle, you create a subclass of [MessageFactory](#) as follows:

```

public class MyMessages extends MessageFactory
{
    // getBundlePath puts together the correct path
    (META-INF/services/org/mule/i18n/my-messages.properties)
    private static final String BUNDLE_PATH = getBundlePath("my");

    public static Message errorMessageOne()
    {
        return createMessage(BUNDLE_PATH, 1);
    }

    public static Message anotherErrorMessage(Object param1, Object param2)
    {
        createMessage(BUNDLE_PATH, 2, param1, param2);
    }
}

```

To load a message from this bundle, pass in the resource bundle name as follows:

```

Message m = MyMessages.anotherErrorMessage("one", "two");
System.out.println(m.toString());

```

This loads the message with ID 2 from `x-messages.properties`, formats the message with the parameters "one" and "two", and prints out the message to `System.out` as follows:

```
Error message with 2 parameters; param one and param two
```

## Creating Message Instances from your Code

If you need Message instances from your custom code (e.g., from a custom transformer), you create them as follows:

```

Message myMessage = MessageFactory.createStaticMessage("Oops");

```

Your Rating:  Results:  0 rates

## Using the Mule Client

### Using the Mule Client

[ [Using Send and Dispatch](#) ] [ [Configuring the Mule Client](#) ] [ [MuleClient as a Web Services Client](#) ] [ [Performing an Event Request Call](#) ] [ [Associating Properties with the Message](#) ] [ [When Not to Use the Mule Client](#) ] [ [Handling Message Collections](#) ] [ [Future Results](#) ] [ [Using the Remote Dispatcher](#) ] [ [Sending Messages to Components Directly](#) ]

In most Mule ESB applications, messages are triggered by an external occurrence such as a message being received on a queue or a file being copied to a directory. However, if you want to send and receive messages programmatically, you can create a Mule client.

MuleClient is a simple interface for Java clients to send and receive messages from a Mule server and other applications. You can use the Mule client for:

- Sending and receiving messages to and from a local or remote Mule server
- Communicating with other applications using any Mule transport
- Making requests to a Mule server behind a firewall using the RemoteDispatcher

The Mule client can be used as a web services client to make SOAP requests using popular SOAP implementations such as Apache CXF. It can also send messages directly to a service component and bypass the transports layer completely, which is useful for testing your service components or when triggering an event from a script or JSP page.

The Mule client can be used with any of the Mule transports, making it a universal client for many types of transports such as JDBC, JMS, FILE, POP3, XMPP, HTTP, etc.

The following sections describe how to use the MuleClient in various scenarios.

## Using Send and Dispatch

The Mule client allows the user to send and receive messages programmatically. For example, to send a JMS message to any application or Mule component listening on `my.queue`, you can use the `dispatch()` method. The `dispatch()` method provides the ability to "fire and forget" messages over an endpoint.

```
MuleClient client = new MuleClient(muleContext);
client.dispatch("jms://my.queue", "Message Payload", null);
```

To make a regular synchronous call to a service and receive a result, you can use the `send()` method:

```
MuleClient client = new MuleClient(muleContext);
MuleMessage result = client.send("tcp://localhost:3456", "Message Payload", null);
```

The client `send()` and `dispatch()` methods expect the following arguments:

1. The Mule URL endpoint: any valid [Mule Endpoint](#) used to determine the transport, endpoint, and other information about delivery of the message. This can also be an endpoint name stored in configuration.
2. The message payload: any object carried by the message.
3. Properties: any properties or meta-data to associate with the message being sent



### Don't use physical endpoints in your code

For clarity, the examples on this page use a physical endpoint URI, such as `jms://myQueue`. However, it is much better practice to define all your endpoints inside a Mule configuration file using the `<endpoint>` element and then reference those endpoint names in your code.

## Configuring the Mule Client

If you are using the Mule client in the same classloader (e.g., a Web App or Mule stand-alone), the client will have access to the server configuration. For example, if you had some endpoints defined in your server configuration file:

```
<http:endpoint host="192.168.0.1" port="80" path="/services" name="serviceEndpoint"/>
```

This endpoint will be accessible by the Mule client:

```
MuleClient client = new MuleClient(muleContext);
client.dispatch("serviceEndpoint", dataObject, null);
```

Essentially, the Mule client will have the same configuration information as the Mule server, since they will both have access to the same registry.

If you are running the Mule client in stand-alone mode, you can still configure it using its own Mule XML configuration file(s). You pass in these files when the client is created:

```
MuleClient client = new MuleClient("http-client-config.xml, shared-client-config.xml");
client.getMuleContext().start();
```

Note that you must start the local Mule context used by the client. You can also create your own Mule context and pass it into the client:

```

//Create a MuleContextFactory
MuleContextFactory muleContextFactory = new DefaultMuleContextFactory();

//create the configuration builder and optionally pass in one or more of these
ConfigurationBuilder builder =
    new SpringXmlConfigurationBuilder("http-client-config.xml, shared-client-config.xml");
//The actual context builder to use
MuleContextBuilder contextBuilder = new DefaultMuleContextBuilder();

//Create the context
MuleContext context = muleContextFactory.createMuleContext(builder, contextBuilder);

//Start the context
context.start();

//Create the client with the context
MuleClient client = new MuleClient(context);

```

## MuleClient as a Web Services Client

The Mule client can be used as a web services client to make SOAP requests using popular SOAP implementations such as [Apache CXF](#) or Axis.

```

MuleClient client = new MuleClient();

//Arguments for the addPerson WS method
String[] args = new String[]{"Ross", "Mason"};

//Call the web service
client.dispatch("axis:http://localhost:38004/PersonService?method=addPerson", args, null);

//Call another method to look up the newly added person
MuleMessage result = client.send
    ("axis:http://localhost:38004/PersonService?method=getPerson", "Ross", null);

//A person object is returned, and all type mapping is handled for you
Person person = (Person)result.getPayload();

System.out.println(person);

```

For SOAP support, you can use the Mule [CXF Module Reference](#) or [Axis Transport](#). For more information about Mule Axis support, see [Axis Web Services and Mule](#).

## Performing an Event Request Call

Making a request to an endpoint is useful when using a transport that has a store of events that you want to request rather than have a listener on the resource.

To make a request for a message, use the `request()` method:

```

MuleClient client = new MuleClient(muleContext);
MuleMessage result = client.request("pop3://ross:blah@mail.my.org", 5000);

```

This code will attempt to receive a message from a mailbox called `ross` on `mail.my.org` and will return after five seconds if no message was received. Calling `request()` works for all Mule supported transports, but it is more usual to make event request calls where there is a store to be queried such as a queue, file directory, or some other repository.

## Associating Properties with the Message

The previous examples set the properties argument to `null`. Properties can be arbitrary, such as to pass around custom metadata with your messages, or they can be transport-specific. The following example demonstrates an asynchronous request/response using JMS and the JMS-specific `JMSReplyTo` property. When the `JMSReplyTo` is set, it is stated in the JMS spec that a receiver of the message should send back

any results to the destination defined in the `JMSReplyTo` header. Mule does this for you.

```
//create the client instance
MuleClient client = new MuleClient(muleContext);

//create properties to associate with the message
Map props = new HashMap();

//Set the JMSReplyTo property, which is where the response message will be sent
props.put("JMSReplyTo", "replyTo.queue");

//dispatch the message asynchronously
client.dispatch("jms://test.queue", "Test Client Dispatch message", props);

//Receive the return message on the replyTo.queue
MuleMessage message = client.request("jms://replyTo.queue", 5000);

//This is the message sent back from the first component to process our message
System.out.println(message.getPayload());
```

## When Not to Use the Mule Client

It's generally not good practice to make calls using the Mule client from your service objects or within extensions to Mule such as routers or transformers.

When you need to dispatch or request events in Mule, you should use the current `org.mule.api.MuleEventContext` and call the `send/dispatch/request` methods on the context instead.

To gain access to the `MuleEventContext` inside your services, you can implement the `org.mule.api.lifecycle.Callable` interface.

If you need to make an event request from a transformer, filter, or interceptor, you should reconsider your design strategy for that event flow.

## Handling Message Collections

Some outbound routers such as the `List Message Splitter`, `Multicaster`, and `Recipient List` may return more than one result message in the following cases:

- There is more than one endpoint configured on the router
- More than one of the endpoints has the `synchronous=true` attribute set

To handle situations where multiple results occur, Mule has introduced a new message type `org.mule.api.MuleMessageCollection`. This type of message contains all message results in the order they were received. Note that `org.mule.api.MuleMessageCollection` extends `org.mule.api.MuleMessage`, so the interface is similar. If there are multiple results, the `MuleMessage.getPayload()` method returns a `java.util.List` containing the payloads of each of the returned messages.

When using the Mule client, you can cast the message return type to get access to all `MuleMessage` objects.

```
MuleClient client = new MuleClient(muleContext);
MuleMessage result = client.send("myEndpoint", "some data", null);

if (result instanceof MuleMessageCollection)
{
    MuleMessageCollection resultsCollection = (MuleMessageCollection) result;
    System.out.println("Number of messages: " + resultsCollection.size());
    MuleMessage[] messages = resultsCollection.getMessagesAsArray();
}
```

## Future Results

The Mule client allows you to make synchronous calls without blocking by using the `sendAsync()` method, which returns a `FutureMessageResult` that can be queried later.

```

MuleClient client = new MuleClient();
FutureMessageResult result = client.sendAsync("http://localhost:8881",
"Message Payload", null);
//Do some more stuff here

Object payload = result.getMessage().getPayload();

```

The FutureMessageResult returned is a placeholder for the real result message when the call returns. By using a future result, you can continue with other tasks while the remote call executes. Calling getMessage() will block until the call returns. Optionally, you can specify a timeout of how long to wait. You can also check if the call has returned using result.isReady().

## Using the Remote Dispatcher

The Mule client can connect to, send, and receive messages from a remote Mule server through a firewall using a remote dispatcher. This should only be used when the remote service being invoked does not expose an endpoint accessible by the Mule client. Note that there is performance overhead when using the remote dispatcher, because all requests and responses are serialized, sent to the server, and deserialized before the real invocation is made from within the firewall.

To use the remote dispatcher, you enable it on the server instance by configuring the remote dispatcher agent. You can ensure that the server can handle both asynchronous and synchronous calls by setting the synchronous attribute to true. You can also set the responseTimeout setting, although often it is better to control it at the MuleClient call level, as each call might have a different timeout requirement.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:client="http://www.mulesoft.org/schema/mule/client"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/client
          http://www.mulesoft.org/schema/mule/client/3.0/mule-client.xsd
          http://www.mulesoft.org/schema/mule/core/3.0
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">
    ...
    <client:remote-dispatcher-agent>
      <client:remote-endpoint address="http://localhost:81" exchange-pattern="request-response"
      responseTimeout="10000"/>
    </client:remote-dispatcher-agent>
    ...
</mule>

```

On the client side, you can now communicate with the remote server via the remote dispatcher agent. For example:

```

// start an empty context for client side
MuleClient client = new MuleClient(true);
RemoteDispatcher dispatcher = client.getRemoteDispatcher("http://localhost:81");

MuleMessage result = dispatcher.sendToRemoteComponent("StockManager", "give me the price of XXX", null
);

StockQuote sq = (StockQuote) result.getPayload();

```

The Mule client executes the StockManager component on a remote Mule server, returning the result to the client. Mule handles all the call marshalling. The first null argument is an optional string of comma-separated transformers to use on the result message. The second null argument contains properties associated with the request.

If you do not want to wait for the result to be returned from the remote server, you can use the sendAsyncToRemoteComponent() method, which returns a FutureMessageResult:

```

// start an empty context for client side
MuleClient client = new MuleClient(true);
RemoteDispatcher dispatcher = client.getRemoteDispatcher("tcp://localhost:60504");
FutureMessageResult result = dispatcher.sendAsyncToRemoteComponent("StockManager", null, "give me the
price of XXX", null);

//do some other stuff

StockQuote sq = (StockQuote) result.getMessage(1000).getPayload();

```

## Specifying the Wire Format

You can specify the wire format to use for dispatching messages by configuring one of the following:

- <xml-wire-format>: uses the XML-Object transformers
- <serialization-wire-format>: uses the ByteArray-Serializable transformers
- <custom-wire-format>: set the class attribute to the class file of the transformer you want to use.



### About Serialization

The Mule Client uses java Serialization. Make certain that all objects in the message implement serializable.

If you do not set the wire format, the serialization format is used. For more information on transformers, see [Using Transformers](#).

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:client="http://www.mulesoft.org/schema/mule/client/3.0"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/client
          http://www.mulesoft.org/schema/mule/client/3.0/mule-client.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">
    ...
    <client:remote-dispatcher-agent>
      <client:remote-endpoint address="http://localhost:81" exchange-pattern="request-response"
      responseTimeout="10000"/>
      <client:xml-wire-format/>
    </client:remote-dispatcher-agent>
    ...
</mule>

```

## Sending Messages to Components Directly

The Mule client provides a convenient way to send a message directly to a component without needing to use a transport when the Mule server is running in the same classloader as the client. This approach can be very useful in testing as well as triggering messages from a JSP page or JavaScript. For example, to dispatch a message directly to your stock quote component called StockManager, you would do the following:

```

MuleClient client = new MuleClient(muleContext);
MuleMessage result = client.sendDirect("StockManager", null, "give me the price of XXX", null);

StockQuote sq = (StockQuote) result.getPayload();

```

Note that the call is `sendDirect`, which tells the Mule client to go directly to the component and not through a transport. You can specify a comma-separated list of transformers to use in the second argument of this call.

Your Rating:

Results: 0 rates

# Mule Object Stores

## [Mule 3.2] Mule Object Stores

[ Object Stores Available in Mule ]

A **object store** is an abstraction for storing objects in Mule. By using an object store, Mule is decoupled from any specific store implementation, allowing you to choose or switch the implementation you want.

Mule uses object stores in:

- services. For SEDA queues, between an inbound and a service component.
- VM transports. In one-way endpoints only. When using one-way endpoints, messages are delivered to the corresponding inbound endpoint via a queue.
- Various filters, routers, and other message processors that need to be persistent.
- [Mule 3.2] Flows. Through the queued-asynchronous processing-strategy. See [Flow Processing Strategies](#) for more information about the queued-asynchronous processing-strategy.
- [Mule 3.2] Clusters. Mule uses a clustered object store for clusters. The clustered object store is in a shared memory grid

## Object Stores Available in Mule

Mule provides two types of default object stores:

- Default in-memory store. This is where all non-persistent data is stored by default. Mule creates a default in-memory store in memory. However, for a cluster, Mule creates the default in-memory store in the shared memory grid.
- Default persistent store. This is where all persistent data is stored by default. Mule creates a default persistent store in the file system. However, for a cluster, Mule creates the default persistent store in the shared memory grid.

In most cases, the default object store will meet your needs. In these cases, no configuration is necessary. However, you can specify a queue store for message persistence (a queue store is an object store for queues). For a service's SEDA queue, you specify the queue store to use by specifying an appropriate child element in the `<queue-profile>` element. The available queue stores, and their respective elements, are listed in the following table:

queue store	description	element
queue-store	References the global queue store.	<code>&lt;queue-store&gt;</code>
default in-memory-queue-store	The default in-memory store.	<code>&lt;default-in-memory-queue-store&gt;</code>
default persistent-queue-store	A file-based store. For a cluster, it is the default in-memory store.	<code>&lt;default-persistent-queue-store&gt;</code>
simple in-memory queue-store	Always an in-memory store.	<code>&lt;simple-in-memory queue-store&gt;</code>
file queue store.	Always a file-based store.	<code>&lt;file-queue-store&gt;</code>

For example, you can request a file-based store for a service's SEDA queue as follows:

```
<service name="serviceExplicitObjectStore">
  ...
  <queue-profile>
    <file-queue-store/>
  </queue-profile>
</service>
```

You can also explicitly configure a custom object store to use for a service's SEDA queues by specifying the `<custom-queue-store>` child element in the `<queue-profile>` element, as shown in the following example:

```

<service name="serviceCustomObjectStore">
  ...
  <queue-profile>
    <custom-queue-store class="..."/>
  </queue-profile>
</service>

```

And you can specify a queue store for a flow, that is, if the flow uses the queued-asynchronous processing strategy. You specify the queue store in an appropriate child element in the global processing strategy for the flow. Here's an example:

```

<queued-asynchronous-processing-strategy name="customStoreQueueStrategy">
  <file-queue-store/>
</queued-asynchronous-processing-strategy>

<flow name="asynchronousFlow" processingStrategy="customStoreQueueStrategy">
  <vm:inbound-endpoint path="input" exchange-pattern="one-way"/>
  <vm:outbound-endpoint path="output" exchange-pattern="one-way"/>
</flow>

```

Your Rating:  Results:  0 rates

## Flow Processing Strategies

### [Mule 3.2] Flow Processing Strategies

[ [About the Synchronous Flow Processing Strategy](#) ] [ [About the Queued-Asynchronous Flow Processing Strategy](#) ] [ [How Mule Selects a Flow Processing Strategy](#) ] [ [Changing the Processing Strategy for Your Flow](#) ] [ [Forcing a Specific Flow Processing Strategy](#) ] [ [Fine-Tuning a Processing Strategy](#) ] [ [Creating a Processing Strategy](#) ] [ [Reusing Processing Strategies](#) ]

A flow processing strategy determines how Mule implements message processing for a given flow. There are several decisions that have to be made:

- Should the message be processed synchronously (on a single thread) or asynchronously (writing the message to a queue for another thread to run)?
- If asynchronously, what are the properties of the queue?
- Again, if asynchronously, what are the properties of the pool of threads used that process messages found in the queue?

Mule features two main flow processing strategies, each one optimal for certain flows. This page introduces both of these processing strategies and details the specific flow characteristics to which each is suited. It then covers the criteria Mule uses to determine the optimal processing strategy for each flow. Next, this page explains the circumstances under which the user can override Mule's initial choice of processing strategy. Finally, this page covers the parameters a user can change or fine-tune in the processing strategy used for a given flow.

### About the Synchronous Flow Processing Strategy

The synchronous approach is used to processes each message on the same thread that initially received the message. After the flow receives a message, all processing, including the processing of the response, is done in that same thread. The one exception is the processing for sections explicitly marked with the `async` element. The synchronous strategy is ideally suited to flows where:

- The sender of the message expects a response. This is known as a "request-response" exchange pattern.
- The flow needs to meet the requirements of transactional processing. In other words, all the steps in the flow are considered a single unit, which must succeed entirely or fail entirely. Additionally, appropriate parties (such as the sender of the message or the administrator of the business process encapsulated by the flow) must be notified of the result. This means that a transactional flow must not hand off processing to other threads, where errors can occur after the transaction is completed.
- The flow's inbound endpoint must be notified of all errors that occur during the processing of the message. This situation is discussed further in Reliability Patterns.

### About the Queued-Asynchronous Flow Processing Strategy

The queued-asynchronous approach uses a queue to decouple the flow's receiver from the other steps in the flow. This means that once the receiver places a message it has just accepted into the queue, it can immediately accept another message. Furthermore, each message waiting in the queue can be assigned a different thread from the pool of threads managed by the queue, and all assigned threads can execute

simultaneously. Such parallel processing is ideal for situations where the receiver can, at peak times, accept messages significantly faster than the rest of the flow can process those messages.



The specific type of queue implemented for the queued-asynchronous flow processing strategy is known as a SEDA queue.

Under the queued-asynchronous processing strategy, the receiver does not have to wait before accepting the next message, and the processing speed for the rest of the steps in the flow is effectively multiplied, because multiple messages are being processed at the same time.

However, the increased throughput facilitated by the asynchronous approach comes at the cost of transactional reliability. Also, the queued-asynchronous approach, which uses two threads to process each message, is not suitable for request-response exchange patterns, which need to be performed entirely on a single thread.

## How Mule Selects a Flow Processing Strategy

Each flow varies in the degree to which it can benefit from the transactional reliability of synchronous processing, as opposed to the high throughput of the queued-asynchronous alternative. Mule automatically weighs key factors specific to each flow and decides whether to set up synchronous or asynchronous processing.

To select a processing strategy for a given flow, Mule evaluates that flow's exchange pattern.

Flows employ one of three exchange patterns, as follows (note that a flow's exchange pattern is determined by the exchange pattern of its inbound endpoint):

- Request-Response exchange patterns involve a message submitted to the flow's receiver by some external sender, who then waits for a response from the flow.
- One-way exchange patterns.

In addition, either of these exchange patterns might involve a transaction.

As the following table details, Mule maps a flow processing strategy to the exchange pattern used by each flow:

Exchange Pattern	Transactional?	Flow Processing Strategy
Request-Response	yes	Synchronous
Request-Response	no	Synchronous
One-way	yes	Synchronous
One-way	no	Queued-Asynchronous

## Changing the Processing Strategy for Your Flow

Mule's choice of processing strategy is almost always optimal for the flow to which it is applied. However, the following options exist:

- In cases where Mule has selected queued-asynchronous processing for a flow, you can specify a synchronous flow instead. See [Forcing a Specific Flow Processing Strategy](#). Note that you cannot force a synchronous flow to become asynchronous.
- You can accept Mule's choice of flow processing strategy, but then proceed to fine-tune that strategy as well. See [Fine-Tuning a Processing Strategy](#). Note that you can only fine-tune a queued-asynchronous strategy. You cannot do any fine-tuning for a synchronous flow.
- You can create a custom flow processing strategy to fit your exact needs. For instance, you might prefer a queued-asynchronous flow that uses an increased number of threads to handle high peak loads. See [Creating a Processing Strategy](#).

## Forcing a Specific Flow Processing Strategy

The procedure to change a processing strategy for an individual flow is straightforward – you don't need to set queuing profiles or service threading profiles. The only change allowed is to force a flow that would otherwise be asynchronous to be synchronous. Flows configured for request-response or transactional processing cannot be converted from synchronous to asynchronous. To force a flow to be synchronous, add the `processingStrategy` attribute to the flow that you want to change and set it to `synchronous`. This is illustrated in the code examples below:

The original code:

```
<flow name="asynchronousToSynchronous">
    <vm:inbound-endpoint path="anyUniqueEndpointName" exchange-pattern="one-way" />
    <vm:outbound-endpoint path="output" exchange-pattern="one-way" />
</flow>
```

The modified code:

```
<flow name="asynchronousToSynchronous" processingStrategy="synchronous">
  <vm:inbound-endpoint path="anyUniqueEndpointName" exchange-pattern="one-way" />
  <vm:outbound-endpoint path="output" exchange-pattern="one-way" />
</flow>
```

## Fine-Tuning a Processing Strategy

You can fine-tune a queued-asynchronous processing strategy by:

- Changing the number of threads available to the flow.
- Limiting the number of messages that can be queued.
- Specifying a queue store to persist data.

You achieve this fine-tuning by specifying parameters for a global processing strategy, then referencing the parameters within the flow or flows you wish to fine-tune. If you don't specify a certain configuration parameter at either the global or local levels, Mule sets a default value for that parameter.

The following example defines a global processing strategy (`asynchronous-processing-strategy`), which specifies `maxThreads="500"`. Together, this parameter and its value specify the maximum number of threads available for use by the queue. The example also presents a flow which references the global processing strategy. This flow:

- Will be asynchronous, because it refers to the asynchronous-processing strategy.
- Will allow up to 500 concurrent threads, because of the value set for `maxThreads`.

```
<queued-asynchronous-processing-strategy name="allow500Threads" maxThreads="500" />

<flow name="manyThreads" processingStrategy="allow500Threads">
  <vm:inbound-endpoint path="manyThreads" exchange-pattern="one-way" />
  <vm:outbound-endpoint path="output" exchange-pattern="one-way" />
</flow>
```

The following table lists the configuration parameters you can specify for the queued-asynchronous strategy. (The synchronous processing strategy cannot be configured):

<b>name</b>	<b>type</b>	<b>queued only</b>	<b>description</b>	<b>optional</b>
maxBufferSize	integer	no	Determines how many requests are queued when the pool reaches maximum capacity and the pool exhausted action is WAIT. The buffer is used as an overflow.	yes
maxQueueSize	integer	yes	The maximum number of messages that can be queued.	yes
maxThreads	integer	no	The maximum number of threads that can be used.	yes
minThreads	integer	no	The number of idle threads kept in the pool when there is no load.	yes
poolExhaustedAction	enum	no	When the maximum pool size or queue size is bounded, this value determines how to handle incoming tasks	yes
queueTimeout	integer	yes	The timeout used when taking events from the queue.	yes
threadTTL	integer	no	Determines how long an inactive thread is kept in the pool before being discarded.	yes
threadWaitTimeout	integer	no	How long to wait in milliseconds when the pool exhausted action is WAIT. If the value is negative, the wait is infinite.	yes

## Configuring the Queue Object store

For the queued-asynchronous strategy, you can implement message persistence by specifying a queue store. For details, see [Mule Object Stores](#)

## Creating a Processing Strategy

If neither the synchronous nor asynchronous processing strategies fit your needs, and fine-tuning the asynchronous strategy is not sufficient, you

can create a custom processing strategy. You create the custom strategy through the `<custom-processing-strategy>` element and configure it using Spring bean properties. This custom processing strategy must implement the `org.mule.CustomProcessingStrategy` interface.

The following code example illustrates a custom processing strategy:

```
<custom-processing-strategy name="customStrategy" class="org.mule.CustomProcessingStrategy">
  <spring:property name="threads" value="500"/>
</custom-processing-strategy>
```

## Reusing Processing Strategies

You can use a named processing strategy, such as the ones created in the previous two sections, on as many flows in an application as you like. Simply:

- Declare the processing strategy, as in:

```
<queued-asynchronous-processing-strategy name="allow500Threads" maxThreads="500"/>
```

- Refer to it in appropriate flows, for instance:

```
<flow name="acceptOrders" processingStrategy="allow500Threads">
  <vm:inbound-endpoint path="acceptOrders" exchange-pattern="one-way"/>
  <vm:outbound-endpoint path="commonProcessing" exchange-pattern="one-way"/>
</flow>

<flow name="processNewEmployee" processingStrategy="allow500Threads">
  <vm:inbound-endpoint path="processNewEmployee" exchange-pattern="one-way"/>
  <vm:outbound-endpoint path="commonProcessing" exchange-pattern="one-way"/>
</flow>

<flow name="receiveInvoice" processingStrategy="allow500Threads">
  <vm:inbound-endpoint path="receiveInvoice" exchange-pattern="one-way"/>
  <vm:outbound-endpoint path="commonProcessing" exchange-pattern="one-way"/>
</flow>
```

Your Rating: 

Results:  0 rates

## Extending Mule ESB 3

### Extending Mule ESB 3

[ [Creating Extensions](#) ] [ [Developing Your Extension](#) ] [ [Promoting Your Extension on MuleForge](#) ] [ [Internationalizing Mule](#) ] [ [Creating Configuration Patterns](#) ]

Mule ESB provides a great deal of default functionality that you can use in your implementation. If you need different functionality, you can extend Mule as described on this page.

### Creating Extensions

You can create five basic types of extensions to Mule: projects, modules, transports, examples, and enterprise patterns.

- A **project** is a stand-alone Mule application.
- A **module** is a package of related functionality in Mule, such as the XML module, which provides XML-based utilities like filters and routers. For a list of the available modules you can use, see [Modules Reference](#).
- A **transport** is a type of module that carries messages between Mule services via a specific protocol. For a list of the available transports

you can use, see [Transports Reference](#).

- An **example** is a sample application that you create to help users get up and running more quickly. Several examples are provided with Mule.
- A **catalog** is a Mule-created Maven archetype created to help Mule users create [custom configuration patterns](#).

Mule provides Maven archetypes that create the templates for each of these types of functionality in seconds, including the configuration files, unit tests, and packages.

An archetype acts as a wizard, prompting you to provide input, and then creates template configuration, source, and unit test files. Furthermore, if you run an archetype on an existing project or module you created, Maven will update it for you.

When working with transports, note that you can [configure an existing transport](#), or you can [create a new one](#). The recommended approach is to try to use and configure an existing transport first.

- Extending Components
- Creating Example Archetypes
- Creating a Custom XML Namespace
- Creating Module Archetypes
- Creating Catalog Archetypes
- Creating Project Archetypes
- Creating Transports
- Creating Custom Routers

## Developing Your Extension

After using the Maven archetype to get started, the recommended practice is to use an integrated development environment (IDE) such as Eclipse or IntelliJ to develop your Mule project, transport, module, or example. The Mule IDE allows you to quickly get up and running developing with Mule in Eclipse. For more information, see [Using IDEs](#).

## Promoting Your Extension on MuleForge

After you have created a new extension, you can submit it as a project on MuleForge. This allows you to share it with the Mule community so you can get feedback on the quality and design of the module before putting it into production. By submitting to MuleForge, you get the benefit of others trying out your module, and others get the benefit of your work. For more information, see [About MuleForge](#).

## Internationalizing Mule

If you will use Mule in countries where English is not spoken, you can extend Mule by internationalizing the strings in the messages and exceptions. Additionally, there are guidelines you should take into consideration to make sure your code handles different locales. For more information, see [Internationalizing Strings](#) and [Internationalization Guidelines](#).

## Creating Configuration Patterns

The creation of custom configuration patterns can allow you to reach new levels of productivity with Mule. Follow this tutorial for [starting to create your own patterns now](#).

Your Rating:  Results:  1 rates

## Extending Components

### Extending Components

There may be cases which call for extending Mule components. (we have [Developing Components](#) for this)

This can be fairly straightforward, such as implementing your own transport or filter. (see [Creating Transports](#))

How can you write your own transformers, filters, etc.

Basic extending mule (primarily filters and transformers)

Extending Mule components may mean something more advanced, including creating custom routers or even implementing custom exceptions strategies. (we have [Creating Custom Routers](#) for this)

Custom components, custom routers, exceptions strategies (rarer)  
implementing your own connectors, other advanced topics

(current - impl your own transports)

Your Rating:  0 rates

Results:  0 rates

## Creating Example Archetypes

### Creating Example Archetypes

[ Configuring Maven ] [ Using the Archetype ] [ The Questions Explained ] [ Example Console Output ] [ Command Line Options ]

Mule provides Maven archetypes that you can use as code templates for your Mule projects. These templates include a set of implementation notes and "todo" pointers that help you get started quickly. The Mule example archetype will help you generate a tailored boilerplate example project in seconds. For more information on using Maven, see [Using Maven](#).

Follow the instructions below to create template files for a new Mule example, including all the necessary Java boilerplate and detailed implementation instructions in comments.

### Configuring Maven

Add the following to the file `settings.xml` (usually in your Maven `conf` or `$HOME/.m2` directory) so that Maven will allow you to execute Mule plug-ins.

```
settings.xml

<settings>
  <pluginGroups>
    <pluginGroup>org.mule.tools</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

### Using the Archetype

First, open a command shell and change to the directory where you want to create your example project.

```
> cd yourDir
```

Next, you will execute the archetype and generate the code. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-example-archetype:create -DartifactId=xxx -DmuleVersion=2.2.0
```

At minimum, you pass in two system parameters:

- `artifactId`: The short name for the project (such as 'myExample'). This must be a single word in lower case with no spaces, periods, hyphens, etc.
- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.



#### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create an example for an earlier version of Mule. In this case, run the `mule-example-archetype` specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-example-archetype:2.1.2:create ...
```



### artifactId

As of Mule 2.2 the artifactId can contain characters such as underscore or hyphen. However, the plug-in will convert the name into a usable form suitable for Java. For example, if the argument is specified as `-DartifactId=My#Awesome-Mule_Project`, the project will be created in a directory of that name, but the project name will be `MyAwesomeMuleProject` and the package name will be `.myawesomemuleproject`

The plug-in will ask various questions (described below) and then generate the files. You can also use this plug-in without user prompts by entering all the arguments at the command line. For a full list of arguments that can be passed in, see the [Command Line Options](#).

After you have answered all the questions, the archetype creates a directory using the example project name you specified that includes a POM file for building with Maven, a Mule configuration file (`conf\mule-config.xml`) that includes the namespaces for the transports and modules you specified and has placeholder elements for creating your first service, and a `package.html` file under `src\main\java` using the package path you specified. It also creates files in the `dist` folder that you need to distribute to users, including a `readme` file and a `batch` file for running the example. Lastly, it creates some template files under `src\test` to help you get started creating a unit test for the example. A new `MULE-README.txt` file will be created in the root of your project explaining what files were created.

## The Questions Explained

The plug-in prompts you to answer several questions about the example you are creating. These may vary according to the options you select. An example of the output is shown [below](#).

### **Provide a description of what the example does:**

You should provide an accurate description of the example with any high-level details of what you can or cannot do with it. This text will be used where a description of the example is required.

### **Which version of Mule is this example targeted at?**

The version of Mule you want to use for your example. This will default to the archetype version passed in on the command line.

### **Will this project be hosted on MuleForge?**

If the example will be hosted on [MuleForge](#), additional information will be added to your example for linking to its issue tracker, web site, build server, and deployment information.

### **Which Mule transports do you want to include in this project?**

A comma-separated list of the transports you plan to use in this example (such as HTTP and VM). This will add the namespaces for those transports to the configuration file.

### **Which Mule modules do you want to include in this project?**

A comma-separated list of the modules you plan to use in this example (such as XML and Scripting). This will add the namespaces for those modules to the configuration file.

## Example Console Output

```
[INFO] description:
*****
Provide a description of what the example does:

[default: ]
*****
```

[INFO] muleVersion:
\*\*\*\*\*

Which version of Mule is this example targeted at?

```
[default: 2.2.0]
*****
```

[INFO] forgeProject:
\*\*\*\*\*

Will This project be hosted on MuleForge? [y] or [n]

```
[default: y]
*****
```

[INFO] transports:
\*\*\*\*\*

Which Mule transports do you want to include in this project?  
(options: axis,cxf,ejb,file,ftp,http,https,imap,imaps,jbpm,jdbc,  
jetty,jms,multicast,pop3,pop3s,quartz,rmi,servlet,smtp,  
smtps,servlet,ssl,tls,stdio,tcp,udp,vm,xmpp):

```
[default: cxf,file,http,jdbc,jms,stdio,vm]
*****
```

[INFO] modules:
\*\*\*\*\*

Which Mule modules do you want to include in this project?  
(options: bulders,client,jaas,jbossts,management,ognl,pgp,scripting,  
spring-extras,sxc,xml):

```
[default: client,management,scripting,sxc,xml]
*****
```

## Command Line Options

By default, this plug-in runs in interactive mode, but it's possible to run it in 'silent' mode by using the following option:

-Dinteractive=false

The following options can be passed in:

Name	Example	Default Value
groupId	-DgroupId=org.mule.examplexxx	org.mule.example.<artifactId>
forgeProject	-DforgeProject=n	y
packagePath	-DpackagePath=org/mule/example	none
transports	-Dtransports=http,vm	cxf,file,http,jdbc,jms,stdio,vm
muleVersion	-DmuleVersion2.2.0	none
packageName	-DpackageName=myPkg	none

description	-Ddescription="some text"	none
modules	-Dmodules=xml,scripting	client,management,scripting,sxc,xml
basedir	-Dbasedir=/projects/mule/tools	<current dir>
package	-Dpackage=org/mule/example/myPkg	none
artifactId	-DartifactId=myMuleExample	mule-application-<artifactId>
version	-Dversion=2.2-SNAPSHOT	<muleVersion>

Your Rating: 

Results:  0 rates

## Creating a Custom XML Namespace

### Creating a Custom XML Namespace

XML schema definitions are used for each module to define the objects and properties that the module makes available to Mule ESB. These configuration elements are introduced using a namespace for each module and associating the namespace with the schema. This page describes how configuration is handled in Mule and what steps are required when writing a new module or transport in Mule.

### Advantages of Using Namespaces

The use of namespaces provides the following advantages:

- Class names are removed from XML so that implementation details are hidden.
- All objects introduced by the module are self-contained by a namespace.
- The schema provides a domain-specific language (DSL) for the module where all objects and properties are described in the schema with type validation.
- The schema can provide additional validation for types, value ranges, and required properties.

### Using Module Schemas

Schemas are located in each package's `main/resources/META-INF` directory. The core schema is in the `mule-core` package, the TCP schema is in the `tcp` package, and so on.

The Mule schema can be used directly or embedded inside Spring. In addition, Spring beans can be created directly inside the Mule schema (just use `<spring:bean ... />`) and elements from other namespaces can be placed in `<other>...</other>`.

### Mule Namespace

NOTE: As of Mule 3.0, version is no longer part of the namespace declaration. Also, "mulesource" is replaced with "mulesoft" in all schema and xml config files.

The default namespace for Mule xml configuration files is `mule`:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core/3.0 META-INF/mule.xsd">
    <!-- Mule config here -->
    <spring:bean ...you can also embed Spring bean definitions directly.../>

    <spring:beans>
        <!-- and you can have nested spring definitions -->
    </spring:beans>
</mule>

```

Note here we have a `spring` namespace declared so we can embed spring beans directly inside your Mule configuration file.

## More on Mixing Mule and Spring

The Mule schema includes the ability to use Spring elements at certain points by including `<spring:bean>` inside `<mule>`. These elements are handled explicitly by Mule code, which delegates their processing to Spring.

Be careful when using Spring elements in your own schema, and check that they behave as expected. The `<bean>` and `<beans>` elements are all forwarded to Spring for processing. In addition, the predefined `mule:mapType` can be used and, when associated with the `ChildMapDefinitionParser`, will automatically construct a map using Spring's `<entry>` elements (this is the **only** way that `<entry>` can be used directly inside Mule elements). For examples, see the use of `mapType` in the Mule schema. Similar behavior with `ChildPropertiesDefinitionParser` should also be possible (but `ChildMapEntry` and `ChildListEntryDefinitionParsers` are unrelated to Spring).

Other namespaces can be introduced via `<spring:beans>` or by adding a dedicated element in a module (see the [Scripting module's `<lang>` element](#)).

## Documentation

You add documentation to the schema using the `<xsd:annotation>` and `<xsd:documentation>` tags:

```
<xsd:element name="my-element" type="myType">
  <xsd:annotation>
    <xsd:documentation>This element does this</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:complexType name="myType">
  <xsd:annotation>
    <xsd:documentation>This type does that</xsd:documentation>
  </xsd:annotation>
</xsd:complexType>
```

While documentation can be added in various places within the schema, tools that use this information follow certain conventions (see below). As a consequence, embedded documentation should:

- Be placed in the element, attribute, or associated type
- Avoid duplicating information in element and type
- Avoid reference elements (`<xsd:element ref="..." />`)
- Make documentation at each level correct and distinct (do not rely on inheritance, but try to avoid duplication)

### IntelliJ Idea

Idea will show documentation defined for an element or attribute, or for the associated type if those are missing. The information is displayed when the user presses Ctrl-J. For more information see [this post about how to work with Mule schemas in IntelliJ](#).

### Eclipse

The Web Tools Platform (WTP) XML editor shows documentation defined for an element or attribute, or for the associated type if those are missing. The information is displayed when you press F2 when an element or attribute is selected or has the cursor on it. The same information is also shown when using the context-sensitive auto-completion functionality by pressing the "CTRL-." key combination.

The WTP XML editor will display "inherited" documentation but does not show documentation associated with referenced global elements.

## Writing Configuration Handlers

When writing a new Mule transport or module, you will need to write a schema definition and the code necessary to parse the XML, but most of the work is done for you. The following section will walk through the process of:

- Defining the *XML Schema*: Describes all the objects that your module exposes, such as transformers, components, filters, routers, agents, etc.
- Writing the *Namespace Handler*: Responsible for configuring the XML parsers for each of the elements that your schema defines.
- Writing a *Definition Parser* for each of the elements (objects) defined in the schema
- Testing your Namespace Handler

### Defining the Schema

If you are not familiar with XML schema, you may want to take an introductory course [here](#). However, Mule defines most of what you need out of the box, so it's fairly straightforward to jump in and write your own. Following are a few key concepts:

- *Complex Types* are defined for each object in the module. Complex types define the *elements* and *attributes* that make up the type. For example, a `connectorType` would define shared attributes for all connectors and define any nested elements such as `<service-overrides>`.

```

<xsd:complexType name="connectorType" mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="receiver-threading-profile" type="threadingProfileType"
minOccurs="0"
                           maxOccurs="1"/>
        <xsd:element name="dispatcher-threading-profile" type="threadingProfileType"
minOccurs="0"
                           maxOccurs="1"/>
        <xsd:group ref="exceptionStrategies" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="service-overrides" type="serviceOverridesType" minOccurs="0"
maxOccurs="1"/>
    </xsd:choice>

    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="createDispatcherPerRequest" type="xsd:boolean"/>
    <xsd:attribute name="createMultipleTransactedReceivers" type="xsd:boolean"/>
</xsd:complexType>

```

Note that complex types can be extended (much like inheritance), so new complex types can be built upon existing ones. Mule provides a number of base complex types out of the box for connectors, agents, transformers, and routers. If you write one of these, your schema should extend the corresponding complex type. Using TCP as an example, here is an excerpt from where we define the `noProtocolTcpConnectorType`:

```

<xsd:import namespace="http://www.mulesoft.org/schema/mule/core/3.0"/>

<xsd:complexType name="noProtocolTcpConnectorType">
    <xsd:complexContent>
        <xsd:extension base="mule:connectorType">
            <xsd:attribute name="sendBufferSize" type="mule:substitutableInt">
                <xsd:annotation>
                    <xsd:documentation>
                        The size of the buffer (in bytes) used when sending data, set on the socket itself.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            <xsd:attribute name="receiveBufferSize" type="mule:substitutableInt">
                <xsd:annotation>
                    <xsd:documentation>
                        The size of the buffer (in bytes) used when receiving data, set on the socket
itself.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
            ...
            <xsd:attribute name="validateConnections" type="mule:substitutableBoolean">
                <xsd:annotation>
                    <xsd:documentation>
                        This "blips" the socket, opening and closing it to validate the connection when
first accessed.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:attribute>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

```

This complex type extends the `mule:connectorType` type. Notice that we need to import the Mule core schema since that is where the `connectorType` is defined.

## Schema Types

Note that the types we use for int, boolean, and all numeric types are custom types called `substitutableInt` or `substitutableBoolean`. These types allow for int values and boolean values but also allow developers to use property placeholders, such as  `${tcp.keepAlive}` as a valid value for the property. These placeholders will be replaced at run-time by real values defined in property files.

*Element definitions* describe what elements are available in the schema. An element has a *type*, which should be declared as a *Complex Type*. For example:

```
<xsd:element name="connector" type="tcpConnectorType"/>
```

This makes the `connector` element available within the `tcp` namespace.

The schema should be called `mule-<short module name>.xsd` and stored in the META-INF of the module or transport.

## Versioning

In Mule, the version of the schema is maintained in the schema URI. This means that the `namespace` and the `targetNamespace` implicitly contain the schema version. Schema URIs use the following convention:

```
http://www.mulesoft.org/schema/mule/core/3.0
```

The first part of the URI `http://www.mulesoft.org/schema/mule/` is the same for each schema. It is then followed by the module's short name, followed by the version of the schema.

## Schema Mapping

To stop the XML parser from loading Mule schemas from the Internet, you add a mapping file that maps the remote schema location to a local classpath location. This mapping is done in a simple properties file called `spring.schemas` located in the META-INF directory for the module/transport.

### spring.schemas

```
http\://www.mulesoft.org/schema/mule/tcp/3.0/mule-tcp.xsd=META-INF/mule-tcp.xsd
```

## Namespace Handler

The namespace handler is responsible for registering definition parsers, so that when an element in the configuration is found, it knows which parser to use to create the corresponding object.

A namespace handler is a single class that is directly associated with a namespace URI. To make this association, there needs to be a file called `spring.handlers` in the root of the META-INF directory of the module or transport. The file contains the following:

### spring.handlers

```
http\://www.mulesoft.org/schema/mule/tcp/3.0=org.mule.transport.tcp.config.TcpNamespaceHandler
```

The `TcpNamespaceHandler` code is very simple because there is a base support class provided:

### TcpNamespaceHandler.java

```
public class TcpNamespaceHandler extends NamespaceHandlerSupport
{
    public void init()
    {
        registerBeanDefinitionParser("connector", new OrphanDefinitionParser(TcpConnector.class, true));
    }
}
```

Here, there should be one or more registrations binding an element name with a definition parser.

## Definition Parsers

The definition parser is where the actual object reference is created. It includes some Spring-specific classes and terminology, so it's worth reading [this introduction](#).

Mule already includes a number of useful definition parsers that can be used for most situations or extended to suit your needs. You can also create a custom definition parser. The following table describes the existing parsers. To see how they are used, see [org.mule.config.spring.handlers.MuleNamespaceHandler](#).

Parser
Description
<a href="#">org.mule.config.spring.parsers.generic.OrphanDefinitionParser</a>
Constructs a single, standalone bean from an element. It is not injected into any other object. This parser can be configured to automatically set the class of the object, the init and destroy methods, and whether this object is a singleton.
<a href="#">org.mule.config.spring.parsers.generic.ChildDefinitionParser</a>
Creates a definition parser that will construct a single child element and inject it into the parent object (the enclosing XML element). The parser will set all attributes defined in the XML as bean properties and will process any nested elements as bean properties too, except the correct definition parser for the element will be looked up automatically. If the class is read from an attribute (when class is null), it is checked against the constraint. It must be a subclass of the constraint.
<a href="#">org.mule.config.spring.parsers.generic.ParentDefinitionParser</a>
Processes child property elements in XML but sets the properties on the parent object. This is useful when an object has lots of properties and it's more readable to break those properties into groups that can be represented as a sub-element in XML.
<a href="#">org.mule.config.spring.parsers.collection.ChildMapEntryDefinitionParser</a>
Allows a series of key value pair elements to be set on an object as a Map. There is no need to define a surrounding 'map' element to contain the map entries. This is useful for key value pair mappings.
<a href="#">org.mule.config.spring.parsers.AbstractHierarchicalDefinitionParser</a>
This definition parser introduces the notion of hierarchical processing to nested XML elements. Definition parsers that extend this class are always child beans that get set on the parent definition parser. A single method <code>getPropertyName</code> must be overridden to specify the name of the property to set on the parent bean with this bean. Note that the property name can be dynamically resolved depending on the parent element. This implementation also supports collections and Maps. If the bean class for this element is set to <code>MapEntryDefinitionParser.KeyValuePair</code> , it is assumed that a Map is being processed and any child elements will be added to the parent Map.
<a href="#">org.mule.config.spring.parsers.AbstractMuleBeanDefinitionParser</a>
This parser extends the Spring provided <code>AbstractBeanDefinitionParser</code> to provide additional features for consistently customizing bean representations for Mule bean definition parsers. Most custom bean definition parsers in Mule will use this base class. The following enhancements are made:
<ul style="list-style-type: none"><li>Attribute mappings can be registered to control how an attribute name in Mule XML maps to the bean name in the object being created.</li><li>Value mappings can be used to map key value pairs from selection lists in the XML schema to property values on the bean being created. These are a comma-separated list of key=value pairs.</li></ul>

- Provides an automatic way of setting the `init-method` and `destroy-method` for this object. This will then automatically wire the bean into the lifecycle of the application context.
- The `singleton` property provides a fixed way to make sure the bean is always a singleton or not.

## Naming Conventions

The number and variety of definition parsers is growing rapidly. To make them more manageable, please use the following conventions.

- Group by function. Abstract bases live in `org.mule.config.spring.parsers`. Under that we have generic, specific, and collection, which should be self-explanatory. Inside those you may want to add further grouping (e.g., `specific.security`).
- Use consistent names for the relationship of the object being created with the surrounding context:
  - **Child** objects are injected into parents (the enclosing DOM element)
  - **Grandchild** are like child, but recurse up the DOM tree more than one generation
  - **Orphan** objects stand alone
  - **Named** objects are injected into a target identified by name rather than DOM location.
  - **Parent** definition parsers are something like facades, providing an alternative interface to the parent.

## Testing

Testing the namespace handler is pretty simple. You configure the object in Mule XML, start the server, and check that the values have been set correctly. For example:

```
public class TcpNamespaceHandlerTestCase extends FunctionalTestCase
{
    protected String getConfigResources()
    {
        return "tcp-namespace-config.xml";
    }

    public void testConfig() throws Exception
    {
        TcpConnector c = (TcpConnector) muleContext.getRegistry().lookupConnector("tcpConnector");
        assertNotNull(c);
        assertEquals(1024, c.getReceiveBufferSize());
        assertEquals(2048, c.getSendBufferSize());
        assertEquals(50, c.getReceiveBacklog());
        assertEquals(3000, c.getReceiveTimeout());
        assertTrue(c.isKeepAlive());
        assertTrue(c.isConnected());
        assertTrue(c.isStarted());
    }
}
```

## Extending Existing Handlers

Instead of creating a new handler, you can extend an existing transport and add new properties and elements. For example, the SSL transport extends the TCP transport.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns="http://www.mulesoft.org/schema/mule/ssl/2.2"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:mule="http://www.mulesoft.org/schema/mule/core/2.2"
    xmlns:tcp="http://www.mulesoft.org/schema/mule/tcp/2.2"
    targetNamespace="http://www.mulesoft.org/schema/mule/ssl/2.2"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"/>
    <xsd:import namespace="http://www.mulesoft.org/schema/mule/core/2.2"
        schemaLocation="http://www.mulesoft.org/schema/mule/core/2.2/mule.xsd" />
    <xsd:import namespace="http://www.mulesoft.org/schema/mule/tcp/2.2"
        schemaLocation="http://www.mulesoft.org/schema/mule/tcp/2.2/mule-tcp.xsd"/>

    <xsd:element name="connector" substitutionGroup="mule:abstract-connector">
        <xsd:annotation>
            <xsd:documentation>
                Connect Mule to an SSL socket, to send or receive data via the network.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="tcp:tcpConnectorType">
                    <xsd:sequence>
                        <xsd:element minOccurs="0" maxOccurs="1" name="client" type=
                            "mule:tlsClientKeyStoreType"/>
                            <xsd:element minOccurs="0" maxOccurs="1" name="key-store" type=
                            "mule:tlsKeyStoreType"/>
                            <xsd:element minOccurs="0" maxOccurs="1" name="server" type=
                            "mule:tlsServerTrustStoreType"/>
                            <xsd:element minOccurs="0" maxOccurs="1" name="protocol-handler" type=
                            "mule:tlsProtocolHandler"/>
                        </xsd:sequence>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
        </xsd:element>
    
```

## Simple Recipe

The following recipe is sufficient for a simple transport (like UDP). The ordering helps guarantee complete coverage.

1. Write a test case for the connector.
  - a. Use IDE's auto completion to test each public getter (as a first approximation to the public API - tidy by hand).
  - b. Set the test value to something other than the default.
2. Write the XML configuration for the connector (`test/resources/foo-connector-test.xml`) using the properties from the test (make sure the import section is correct).
3. Write the schema definition (tweaking until the XML connector config shows no errors) (`META-INF/mule-foo.xsd`).
4. Write the namespace handler (and any needed definition parsers) (`src/main/java/org/mule/providers/foo/config/FooNamespaceHandler`)
5. Set the Spring handler mapping (`META-INF/spring.handlers`).
6. Set the local schema mapping (`META-INF/spring.schemas`).
7. Make sure the test runs.
8. Check properties against the documentation and make consistent (but note that things like connection strategy parameters are handled by an embedded element that is itself inherited from the `connectorType`) and then re-run the test.

## Resources

- A useful set of PDF slides that give an overview of the new approach in Spring and (slides 29 on) given an introductory example. The Mule code is more complex, but follows the same structure: `org.mule.config.spring.handlers.MuleNamespaceHandler` is the namespace handler; `org.mule.config.spring.parsers.AbstractMuleBeanDefinitionParser` and subclasses are the bean definition parsers.
- A couple of blog posts (1, 2) that give a developer's-eye overview.
- Useful papers on mutable/extensible containers 1, 2

# Creating Module Archetypes

## Creating Module Archetypes

[ [Configuring Maven](#) ] [ [Using the Archetype](#) ] [ [The Questions Explained](#) ] [ [Example Console Output](#) ] [ [Updating an Existing Module](#) ] [ [Command Line Options](#) ]

Mule provides Maven archetypes that you can use as code templates for your Mule modules that you want to host on MuleForge or include with the Mule distribution. These templates include a set of implementation notes and "todo" pointers that help you get started quickly. The Mule module archetype will help you generate a tailored boilerplate module in seconds. For more information on using Maven, see [Using Maven](#).

### Updating Existing Modules and Transports

The Module archetype allows developers to create new Mule modules or upgrade existing Mule modules adding support for schemas and registry bootstrapping. See [Updating an Existing Module](#) for more information.

Follow the instructions below to create template files for a new Mule module, including all the necessary Java boilerplate and detailed implementation instructions in comments.

### Configuring Maven

Add the following to the file `settings.xml` (usually in your Maven `conf` or `$HOME/.m2` directory) so that Maven will allow you to execute Mule plug-ins.

```
settings.xml

<settings>
  <pluginGroups>
    <pluginGroup>org.mule.tools</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

### Using the Archetype

First, open a command shell and change to the directory where you want to create your module.

```
> cd yourDir
```

Next, you will execute the archetype and generate the code. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-module-archetype:create -DartifactId=xxx -DmuleVersion=3.1.1
                                         -DarchetypeArtifactId=mule-module-archetype
```

At minimum, you pass in these system parameters:

- `artifactId`: The short name for the project (such as 'myApp'). This must be a single word in lower case with no spaces, periods, hyphens, etc.
- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.
- `archetypeArtifactId`: Enter `mule-module-archetype` when running the module archetype.



### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create a module for an earlier version of Mule. In this case, run the mule-module-archetype specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-module-archetype:3.1.1:create ...
```



### artifactId

The artifactId can contain characters such as underscore or hyphen. However, the plug-in will convert the name into a usable form suitable for Java. For example, if the argument is specified as **-DartifactId=My#Awesome-Mule\_Project**, the project will be created in a directory of that name, but the project name will be **MyAwesomeMuleProject** and the package name will be **.myawesomemuleproject**

The plug-in will ask various questions (described below) and then generate the files. You can also use this plug-in without user prompts by entering all the arguments at the command line. For a full list of arguments that can be passed in, see the [Command Line Options](#).

After you have answered all the questions, the archetype creates a directory using the module name you specified that includes a POM file for building with Maven, a Mule configuration file (`src\main\resources\mule-config.xml`) that includes the namespaces for the transports and modules you specified and has placeholder elements for creating your first service, and a `package.html` file under `src\main\java` using the package path you specified. Lastly, it creates some template files under `src\test` to help you get started creating a unit test for the module. A new `MULE-README.txt` file will be created in the root of your project explaining what files were created.

## The Questions Explained

The plug-in prompts you to answer several questions about the project you are writing. These may vary according to the options you select. An example of the output is shown [below](#).

### **Are you creating a new module (rather than updating an existing one)?**

If you are creating a brand new Mule module, chose yes here. The wizard will then ask you what resources you want to create. If you are updating an existing module, choose no, and see [Updating an Existing Module](#) for more information. The following questions get asked if you are creating a new module.

### **Provide a description of what the project does:**

You should provide an accurate description of the module with any high-level details of what you can or cannot do with it. This text will be used where a description of the module is required.

### **Which version of Mule is this project targeted at?**

The version of Mule you want to use for your module. This will default to the archetype version passed in on the command line.

### **Will this project be hosted on MuleForge?**

If the module is going to be hosted on [MuleForge](#), additional information will be added to your project for linking to its issue tracker, web site, build server and deployment information.

### **Which Mule transports do you want to include in this project?**

A comma-separated list of the transports you plan to use in this module (such as HTTP and VM). This will add the namespaces for those transports to the configuration file.

### **Which Mule modules to you want to include in this project?**

A comma-separated list of the modules you are extending with this module (such as XML and Scripting). This will add the namespaces for those modules to the configuration file.

### **Will this module have a custom schema for configuring the module in Xml?**

All new modules should define an XML schema that defines how the module is configured. If you do not use this option, users will have to use generic configuration to use your module.

### **Will this module make objects available in the Registry as soon as it's loaded?**

The [registry bootstrap](#) is a properties file that specifies class names of simple objects that can be made available in the Mule Registry as soon as the module is loaded. This is useful for registering custom transformers or [expression evaluators](#).

## Example Console Output

```
*****
Are you creating a new module (rather than updating an existing one)? [y] or [n]
[default: y]
*****
Y
[INFO] description:
*****
Provide a description of what the module does:
[default: ]
*****
foo Bar
[INFO] muleVersion:
*****
Which version of Mule is this module targeted at?
[default: 3.1.1]
*****
[INFO] forgeProject:
*****
Will this module be hosted on MuleForge? [y] or [n]
[default: y]
*****
[INFO] transports:
*****
Which Mule transports do you want to include in this module?

(options: axis, cxf, ejb, file, ftp, http, https, imap, imaps, jbpm, jdbc,
jetty, jetty-ssl, jms, jnp, multicast, pop3, pop3s, quartz, rmi, servlet,
smtp, smtpts, servlet, ssl, tls, stdio, tcp, udp, vm, xmpp):
[default: vm]
*****
[INFO] modules:
*****
Which Mule modules do you want to include in this module?

(options: bulders, client, jaas, jbossts, management, ognl, pgp, scripting,
spring-extras, sxc, xml):
[default: client]
*****
[INFO] hasCustomSchema:
*****
Will this module have a custom schema for configuring the module in Xml? [y] or [n]
[default: y]
*****
[INFO] hasBootstrap:
*****
Will this module make objects available in the Registry as soon as it's loaded? [y] or [n]
[default: n]
*****
```

## Updating an Existing Module

The module archetype can be used for updating existing modules and transports. It allows developers to add template code for schema configurations and [bootstrap the registry](#). It will leave your existing code untouched.

For example, if your existing module or transport is located under `/projects/foo`, you update the project by running the following commands:

```
cd /project/foo  
mvn mule-module-archetype:create -DartifactId=foo -DmuleVersion=3.1.1  
-DarchetypeArtifactId=mule-module-archetype
```

Notice that the `artifactId` must be set to the name of your project. This ensures that any new classes will be created with the same naming scheme.

When you run this command, you will be prompted with three questions. The first question will ask you whether this is a new project. Make sure you select 'n' so that the wizard will upgrade your existing module or transport. It then asks you the last two questions about the custom schema and registry bootstrap. After you answer the questions, the code will be created and a new `MULE-UPDATE-README.txt` file will be created in the root of your project explaining what files were created.

## Command Line Options

By default, this plug-in runs in interactive mode, but it's possible to run it in 'silent' mode by using the following option:

```
-Dinteractive=false
```

The following options can be passed in:

Name	Example	Default Value
groupId	<code>-DgroupId=org.mule.applicationxxx</code>	<code>org.mule.application.&lt;artifactId&gt;</code>
packagePath	<code>-DpackagePath=org/mule/application</code>	<code>none</code>
transports	<code>-Dtransports=http,vm</code>	<code>cx,http,jdbc,jms,stdio,vm</code>
muleVersion	<code>-DmuleVersion=3.1.1</code>	<code>none</code>
packageName	<code>-DpackageName=myPkg</code>	<code>none</code>
description	<code>-Ddescription="some text"</code>	<code>none</code>
modules	<code>-Dmodules=xml,scripting</code>	<code>client,management,scripting,sxc,xml</code>
basedir	<code>-Dbasedir=/projects/mule/tools</code>	<code>&lt;current dir&gt;</code>
package	<code>-Dpackage=org/mule/application/myPkg</code>	<code>none</code>
artifactId	<code>-DartifactId=myMuleProject</code>	<code>mule-application-&lt;artifactId&gt;</code>
version	<code>-Dversion=1.0-SNAPSHOT</code>	<code>&lt;muleVersion&gt;</code>

Your Rating: 

Results:  0 rates

## Creating Catalog Archetypes

### Creating Catalog Archetypes

[ [Implementation Overview](#) ] [ [Using Maven](#) ] [ [Configuring Maven](#) ] [ [Creating a New Pattern Catalog](#) ] [ [Creating a New Pattern](#) ] [ [The Questions Explained](#) ] [ [Example Console Output](#) ]

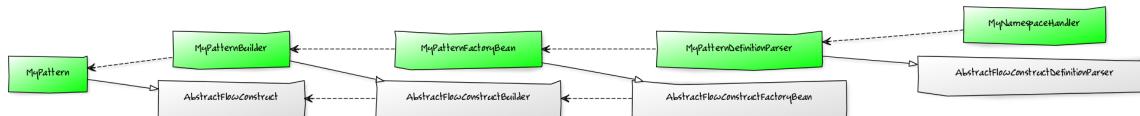
Configuration patterns are elements that simplify configuring specific and recurring activities. Mule comes with some [ready made patterns](#) that cover generic scenarios.

You can create your own configuration patterns and even build catalogs of patterns tailored to your needs. The benefits of such catalogs are numerous. Indeed, encapsulating bits of know-how in ready-to-be-consumed configuration elements can increase the productivity of your development teams, reduce the risk of errors, and facilitate the application of good practices.

## Implementation Overview

A dedicated Mule module makes the best host for your patterns, as it provides all the necessary infrastructure for supporting the custom XML configuration elements your patterns will be based on.

A configuration pattern is actually composed of a core class, a builder, and two configuration-related classes. These all rely on existing abstract classes for the bulk of their behavior. In the diagram below, the pattern concrete classes are highlighted in green.



On top of these classes, you also need specific XML schema elements. These schema elements allow the pattern to be used in any Mule configuration file.

Since creating all these artifacts by hand can be tedious, the best approach is to use the Maven archetype we have created to generate most of the pattern automatically.



The Pattern Catalog Archetype is available in Mule 3.0.1 and above.

## Using Maven

The next sections describe how to configure and use the Maven archetype so that it automatically generates the bulk of a pattern.

### Configuring Maven

Before you start, add the following to the file `settings.xml` (usually in your Maven `conf` or `$HOME/.m2` directory) so that Maven will allow you to execute Mule plug-ins.

```
settings.xml

<settings>
    <pluginGroups>
        <pluginGroup>org.mule.tools</pluginGroup>
    </pluginGroups>
    ...
</settings>
```

### Creating a New Pattern Catalog



A pattern catalog is actually a regular Mule module. If you already have an existing module (most probably created with the Module Archetype) and are happy storing your patterns in it, you can directly proceed to the next section.

First, open a command shell and change to the directory where you want to create your module.

```
> cd yourDir
```

Next, you will execute the archetype and generate the module to host the catalog later. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-module-archetype:create -DartifactId=xxx -DmuleVersion=3.1.0
```

At minimum, you pass in these system parameters:

- `artifactId`: The short name for the project (such as 'myApp'). This must be a single word in lower case with no spaces, periods, hyphens, etc.
- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.



### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create a module for an earlier version of Mule. In this case, run the mule-module-archetype specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-catalog-archetype:3.1.0:create ...
```



### artifactId

The artifactId can contain characters such as underscore or hyphen. However, the plug-in will convert the name into a usable form suitable for Java. For example, if the argument is specified as **-DartifactId=My#Awesome-Mule\_Project**, the project will be created in a directory of that name, but the project name will be **MyAwesomeMuleProject** and the package name will be **.myawesomemuleproject**

The module archetype will [ask various questions](#) and then generate the files. You can also use this plug-in without user prompts by entering all the arguments at the command line. For a full list of arguments that can be passed in, see the [Command Line Options](#).

After you have answered all the questions, the archetype creates a directory using the module name you specified that includes a POM file for building with Maven, a Mule configuration file (`src\main\resources\mule-config.xml`) that includes the namespaces for the transports and modules you specified and has placeholder elements for creating your first service, and a `package.html` file under `src\main\java` using the package path you specified. Lastly, it creates some template files under `src\test` to help you get started creating a unit test for the module. A new `MULE-README.txt` file will be created in the root of your project explaining what files were created.

## Creating a New Pattern

First, open a command shell and change to the root directory of your pattern catalog.

```
> cd yourPatternCatalog
```

Next, you will execute the archetype and generate the configuration pattern. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-catalog-archetype:new-pattern -DmuleVersion=3.1.0
```

At minimum, you pass in this system parameters:

- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.



### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create a module for an earlier version of Mule. In this case, run the mule-module-archetype specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-catalog-archetype:3.1.0:new-pattern ...
```

The plug-in will ask various questions and then generate the files.

After you have answered all the questions, the archetype will have created all the classes mentioned above. It will have also created a `TODO` file named after your pattern (like: `my-pattern.todo`) that contains information on how to finish the implementation of your pattern.

## The Questions Explained

The plug-in prompts you to answer several questions about the pattern you are creating.

### **Are you creating a new module (rather than updating an existing one)?**

If you are creating an brand new Mule module, chose yes here. The wizard will then ask you what resources you want to create. If you are updating an existing module, choose no, and see [Updating an Existing Module](#) for more information. The following questions get asked if you are creating a new module.

### **What XML tag name should be used for the new pattern?**

This name will be used in your XML configuration. It usually is all lower case with dash ( - ) used as a separator.

### **What is the fully qualified class name of the new pattern?**

All the scaffolding classes and their package names will be inferred from the fully qualified name of the core pattern class. You must not target the default package.

### **What will be the type of this pattern?**

This specifies what will be the level of flexibility your pattern will allow in its configuration.

- **mp**: The pattern is a pure message processor designed to be used within a flow alongside other message processors. It doesn't support an inbound source of message like an endpoint or a router.
- **ms**: The pattern receives messages from any kind of message source, like endpoints or routers.
- **si**: The pattern receives messages from a single inbound endpoint. It can optionally be configured with inbound transformers. The [Simple Service](#) pattern is of this kind.
- **siso**: The pattern receives messages from a single inbound endpoint and dispatches to a single outbound endpoint. The [Bridge](#), [Validator](#) and [Web Service Proxy](#) patterns are of this kind.

## **Example Console Output**

```
*****
What XML tag name should be used for the new pattern?
(Prefer lower-case and use dashes as separators, like: my-pattern)
[default: null]
*****
my-pattern

[INFO] patternFQCN:
*****
```

```
What is the fully qualified class name of the new pattern?
(For example: com.acme.pattern.MyPattern
Note that supporting classes will be created in: com.acme.pattern.builder and com.acme.pattern.config)
[default: null]
*****
com.acme.pattern.MyPattern

[INFO] patternType:
*****
```

```
What will be the type of this pattern? [mp] or [ms] or [si] or [siso]
(Details of each type:
mp: the pattern is a pure message processor designed to be used within a flow alongside other message
processors
ms: the pattern receives messages from any kind of message source, like endpoints or routers
si: the pattern receives messages from a single inbound endpoint
siso: the pattern receives messages from a single inbound endpoint and dispatches to a single outbound
endpoint)
[default: mp]
*****
siso
```

Your Rating:  Results:  0 rates

## **Creating Project Archetypes**

### **Creating Project Archetypes**

[ Configuring Maven ] [ Using the Archetype ] [ The Questions Explained ] [ Example Console Output ] [ Command Line Options ]

Mule provides Maven archetypes that you can use as code templates for your Mule projects. These templates include a set of implementation notes and "todo" pointers that help you get started quickly. The Mule project archetype will help you generate a tailored boilerplate project in seconds. For more information on using Maven, see [Using Maven](#).

Follow the instructions below to create template files for a new project, including all the necessary Java boilerplate and detailed implementation instructions.

## Configuring Maven

Add the following to the file `settings.xml` (usually in your Maven `conf` or `$HOME/.m2` directory) so that Maven will allow you to execute Mule plug-ins.

```
<settings>
  <pluginGroups>
    <pluginGroup>org.mule.tools</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

## Using the Archetype

First, open a command shell and change to the directory where you want to create your project.

```
> cd yourDir
```

Next, you will execute the archetype and generate the code. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-project-archetype:create -DartifactId=xxx -DmuleVersion=3.1.1
```

At minimum, you pass in two system parameters:

- `artifactId`: The short name for the project (such as 'myApp'). This must be a single word in lower case with no spaces, periods, hyphens, etc.
- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.

### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create a project for an earlier version of Mule. In this case, run the `mule-project-archetype` specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-project-archetype:3.1.1:create ...
```

### artifactId

The `artifactId` can contain characters such as underscore or hyphen. However, the plug-in will convert the name into a usable form suitable for Java. For example, if the argument is specified as `-DartifactId=My#Awesome-Mule_Project`, the project will be created in a directory of that name, but the project name will be `MyAwesomeMuleProject` and the package name will be `.myawesomemuleproject`

The plug-in will ask various questions (described below) and then generate the files. You can also use this plug-in without user prompts by entering all the arguments at the command line. For a full list of arguments that can be passed in, see the [Command Line Options](#).

After you have answered all the questions, the archetype creates a directory using the project name you specified that includes a POM file for building with Maven, a Mule configuration file (`src\main\resources\mule-config.xml`) that includes the namespaces for the transports and modules you specified and has placeholder elements for creating your first service, and a `package.html` file under `src\main\java` using the package path you specified. Lastly, it creates some template files under `src\test` to help you get started creating a unit test for the project. A new `MULE-README.txt` file will be created in the root of your project explaining what files were created.

## The Questions Explained

The plug-in prompts you to answer several questions about the project you are writing. These may vary according to the options you select. An example of the output is shown [below](#).

### ***Provide a description of what the project does:***

You should provide an accurate description of the project with any high-level details of what you can or cannot do with it. This text will be used where a description of the project is required.

### ***Which version of Mule is this project targeted at?***

The version of Mule you want to use for your project. This will default to the archetype version passed in on the command line.

### ***What is the base Java package path for this project?***

This should be a Java package path for your project, such as com/mycompany/project. Note that you must use slashes for separators, not periods.

### ***Which Mule transports do you want to include in this project?***

A comma-separated list of the transports you plan to use in this project (such as HTTP and VM). This will add the namespaces for those transports to the configuration file.

### ***Which Mule modules do you want to include in this project?***

A comma-separated list of the modules you plan to use in this project (such as XML and Scripting). This will add the namespaces for those modules to the configuration file.

## Example Console Output

```
[INFO] description:  
*****
```

Provide a description of what the project does:

```
[default:]  
*****
```

```
[INFO] muleVersion:  
*****
```

Which version of Mule is this module targeted at?

```
[default: 3.1.0]  
*****
```

```
[INFO] package:  
*****
```

What is the base Java package path for this project? (i.e. com/mycompany/project):

```
[default:]  
*****
```

```
[INFO] transports:  
*****
```

Which Mule transports do you want to include in this project?

```
(options: axis,cxf,ejb,file,ftp,http,https,imap,imaps,jbpm,jdbc,  
jetty,jms,multicast,pop3,pop3s,quartz,rmi,servlet,smtp,  
smtps,servlet,ssl,tls,stdio,tcp,udp,vm,xmpp):
```

```
[default: cxf,file,http,jdbc,jms,stdio,vm]  
*****
```

```
[INFO] modules:  
*****
```

Which Mule modules do you want to include in this project?

```
(options: bulders,client,jaas,jbossts,management,ognl,pgp,scripting,  
spring-extras,sxc,xml):
```

```
[default: client,management,scripting,sxc,xml]  
*****
```

## Command Line Options

By default, this plug-in runs in interactive mode, but it's possible to run it in 'silent' mode by using the following option:

```
-Dinteractive=false
```

The following options can be passed in:

Name	Example	Default Value
groupId	-DgroupId=org.mule.applicationxxx	org.mule.application.<artifactId>
packagePath	-DpackagePath=org/mule/application	none
transports	-Dtransports=http,vm	cx,f,file,http,jdbc,jms,stdio,vm
muleVersion	-DmuleVersion=3.1.0	none
packageName	-DpackageName=myPkg	none
description	-Ddescription="some text"	none

modules	-Dmodules=xml,scripting	client,management,scripting,sxc,xml
basedir	-Dbasedir=/projects/mule/tools	<current dir>
package	-Dpackage=org/mule/application/myPkg	none
artifactId	-DartifactId=myMuleProject	<artifactId>
version	-Dversion=1.0-SNAPSHOT	<muleVersion>

Your Rating: 

Results:  1 rates

## Creating Transports

[ [Overview](#) ] [ [Transport Interfaces](#) ] [ [Implementation](#) ] [ [Connectors](#) ] [ [Message Receivers](#) ] [ [Message Dispatchers](#) ] [ [Message Requesters](#) ] [ [Service Descriptors](#) ] [ [Coding Standards](#) ] [ [Package Structure](#) ]

Transports are used to implement message channels and provide connectivity to an underlying data source or message channel in a consistent way. Mule ESB provides transports for many different protocols, including File, FTP, HTTP, JMS, JDBC, Quartz, and many more. For a complete list, see [Transports Reference](#). There are also community-created transports on [MuleForge](#). If you need to send messages on a protocol other than those provided, you can create a new transport.

### Overview

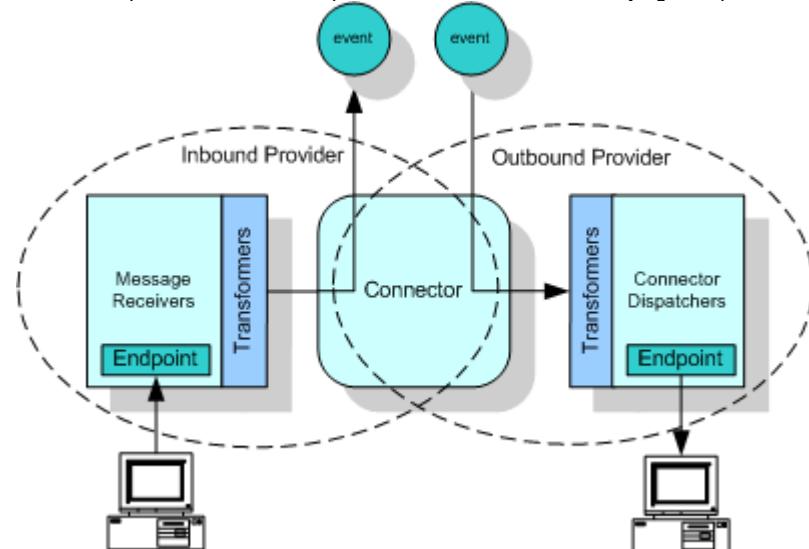
When creating a new transport, you must implement a set of Mule interfaces in the `org.mule.transport` package, and then extend the provided abstract classes. For a quick start, you can use the [Maven transport archetype](#) as a code template for your transports. If you want to update an existing transport, use the [Creating Module Archetypes](#) instead.

Mule transports can be one of the following types:

1. inbound-only: Components can only subscribe to events. They cannot dispatch events.
2. outbound-only: Components can only dispatch events. They cannot subscribe to events.
3. inbound-outbound: Components can subscribe and dispatch events

### Transport Interfaces

A transport consists of a set of interface implementations that expose the features for the underlying transport.



Interface	Role
Connector	Used to manage Receivers, Dispatchers and Requesters and to store any configuration information.

Message Receiver	Implements the server part of the transport. For example, a TcpMessageReceiver creates a server socket to receive incoming requests. A MessageReceiver instance is created when this transport is used for inbound communication.
Message Dispatcher	Implements the client part of the transport. For example, a TcpMessageDispatcher opens a socket to send requests. A MessageDispatcher instance is created when this transport for outbound communication.
Message Requester	Is also used to receive incoming messages like the MessageReceiver but rather than subscribing to inbound events or polling a resource or message channel messages are only received on "request". Inbound endpoints on services always use a MessageReceiver rather than a MessageRequester but they can be used elsewhere programmatically to request a single message from a message channel or resource.
Message Factory	Creates a MuleMessage instance from the incoming transport message. For example, the HTTP message factory creates a new MuleMessage by using the HTTP request's input stream as payload and puts all request headers as properties to the MuleMessage.
Transformers	Transformers are used to convert data between the transport-specific data format and another format, such as from an HTTP response to a string. The dispatcher must call <code>MuleEvent.transformMessage()</code> to transform the message.
Endpoints	The means of configuring the use of message channels or resource as part of service configuration. The endpoint defines which transport to use and includes settings like the host or queue name, the filter to use, and transaction info. Defining an endpoint on a service will cause Mule to create the necessary transport connector for the protocol being used.

When writing a transport, you must implement the following interfaces that define the contract between Mule and the underlying technology.

#### ***org.mule.api.transport.Connector***

The connector is used by Mule to register listeners and create message dispatchers for the transport. Configuration parameters that should be shared by all Message Receivers, Dispatchers and Requesters are stored on the connector. Usually, only one connector instance is needed for multiple inbound and outbound endpoints as multiple Message Receivers, Dispatchers and Requesters can be associated with a connector. However, where the underlying transport API has the notion of a connection such as the JMS or JDBC API, there should be a one-to-one mapping between the Mule connector and the underlying connection.

#### ***org.mule.api.transport.MessageReceiver***

The Message Receiver is used to receive incoming data from the underlying transport and package it as an event. The Message Receiver is essentially the server implementation of the transport (where the Message Dispatcher is a client implementation). For example, the HTTP Message Receiver is an HTTP server implementation that accepts HTTP requests. An implementation of this class is needed if the transport supports inbound communication.

#### ***org.mule.api.transport.MessageDispatcher***

The Message Dispatcher is used to send messages, which is akin to making client calls with the underlying technology. For example, the CXF Message Dispatcher will make a web service call. An implementation of this class is needed if the transport supports outbound communication. The Message Dispatcher must call `MuleEvent.transformMessage()` to invoke any necessary transformers before dispatching it.

#### ***org.mule.api.transport.MessageRequester***

The Message Requester is used to request messages from a message channel or resource rather than subscribing to inbound events or polling a resource for messages. This is often used programmatically but is not used for inbound endpoints configured on services. An implementation of this class is needed if the transport supports the requesting of messages from a message channel.

#### ***org.mule.api.transport.MessageDispatcherFactory***

This is a factory class used to create MessageDispatcher instances. An implementation of this class is needed if the transport supports outbound communication.

#### ***org.mule.api.transport.MuleMessageFactory***

The message factory is used to provide a consistent way of creating MuleMessage instances from the transport's message format.

### **Implementation**

Mule provides abstract implementations for all of the above interfaces. These implementations handle all the Mule specifics, leaving a few abstract methods where custom transport code should be implemented. Therefore, writing a custom transport is as easy as writing/embedding client and or server code specific to the underlying technology. The following sections describes the implementations available to you.

For a quick start, use the Maven [Transport Archetype](#).

### **Connectors**

The [org.mule.transport.AbstractConnector](#) implements all the default functionality required for Mule connectors, such as threading configuration and receiver/dispatcher management. For details about the standard connector properties, see [Configuring a Transport](#).

You can set further properties on the connector that act as defaults. For example, you can set endpoint properties that are used by default unless you override them when configuring a specific endpoint.

Sometimes the connector is responsible for managing a connection resource of the transport where the underlying technology has the notion of a connection, such as in JMS or JDBC. These types of connectors will have a one-to-one mapping between a Mule connector and the underlying connection. Therefore, if you want to have two or more physical JMS connections in a single Mule instance, a new connector should be created for each connection.

For other transports, there will be only one connector of a particular protocol in a Mule instance that manages all endpoint connections. One such example would be socket-based transports such as TCP where each receiver manages its own ServerSocket and the connector manages multiple receivers.

### Methods to Implement

Method Name	Description	Required
doInitialise()	Is called once all bean properties have been set on the connector and can be used to validate and initialize the connector's state.	No
doStart()	If there is a single server instance or connection associated with the connector (such as AxisServer or a JMS or JDBC Connection), this method should put the resource in a started state.	No
doConnect()	Makes a connection to the underlying resource if this is not handled at the receiver/dispatcher level.	No
doDisconnect()	Close any connection made in doConnect().	No
doStop()	Should put any associated resources into a stopped state. Mule automatically calls the stop() method.	No
doDispose()	Should clean up any open resources associated with the connector.	No

## Message Receivers

Message Receivers will behave a bit differently for each transport, but Mule provides some standard implementations that can be used for polling resources and managing transactions for the resource. Usually there are two types of Message Receivers: Polling and Listener-based.

- A Polling Receiver polls a resource such as the file system, database, and streams.
- A Listener-based receiver registers itself as a listener to a transport. Examples would be JMS (`javax.message.MessageListener`) and Pop3 (`javax.mail.MessageCountListener`). These base types may be transacted.

The abstract implementations provided by Mule are described below.

### Abstract Message Receiver

The [AbstractMessageReceiver](#) provides methods for routing events. When extending this class, you should set up the necessary code to register the object as a listener to the transport. This will usually be a case of implementing a listener interface and registering itself.

### Methods to Implement

Method name	Description	Required
doConnect()	Should make a connection to the underlying transport, such as to connect to a socket or register a SOAP service. When there is no connection to be made, this method should be used to check that resources are available. For example, the FileMessageReceiver checks that the directories it will be using are available and readable. The MessageReceiver should remain in a 'stopped' state even after the doConnect() method is called. This means that a connection has been made but no events will be received until the start() method is called. Calling start() on the MessageReceiver will call doConnect() if the receiver hasn't connected.	Yes
doDisconnect()	Disconnects and tidies up any resources allocated using the doConnect() method. This method should return the MessageReceiver in a disconnected state so that it can be connected again using the doConnect() method.	Yes
doStart()	Should perform any actions necessary to enable the receiver to start receiving events. This is different from the doConnect() method, which actually makes a connection to the transport but leaves the MessageReceiver in a stopped state. For polling-based MessageReceivers, the doStart() method simply starts the polling thread. For the Axis message receiver, the start method on the SOAPService is called. The action performed depends on the transport being used. Typically, a custom transport doesn't need to override this method.	No
doStop()	Should perform any actions necessary to stop the receiver from receiving events.	No

doDispose()	Is called when the connector is being disposed and should clean up any resources. The doStop() and doDisconnect() methods will be called implicitly when this method is called.	No
-------------	---	----

## Polling Message Receiver

Some transports poll a resource periodically waiting for new data to arrive. The polling message receiver, which is based on [AbstractPollingMessageReceiver](#), implements the code necessary to set up and destroy a listening thread and provides a single method `poll()` that is invoked repeatedly at a given frequency. Setting up and destroying the listening thread should occur in the `doStart()` and `doStop()` methods respectively.

### Methods to Implement

Method name	Description	Required
poll()	Is executed repeatedly at a configured frequency. This method should execute the logic necessary to read the data and return it. The data returned will be the payload of the new message. Returning null will cause no event to be fired.	Yes

## Transacted Polling Message Receiver

The `TransactedPollingMessageReceiver` can be used by transaction-enabled transports to manage polling and transactions for incoming requests. This receiver uses a transaction template to execute requests in transactions, and the transactions themselves are created according to the endpoint configuration for the receiver. Derived implementations of this class must be thread safe, as several threads can be started at once for an improved throughput.

### Methods to Implement

You implement the following methods for the transacted polling message receiver in addition to those in the standard Message Receiver:

Method name	Description	Required
getMessages()	Returns a list of objects that represent individual message payloads. The payload can be any type of object and will be sent to Mule services wrapped in a <code>MuleEvent</code> object.	Yes
processMessage(Object)	is called for each object in the list returned from <code>getMessages()</code> . Each object processed is managed in its own transaction.	Yes

## Thread Management

It's common for receivers to spawn a thread per request. All receiver threads are allocated using the `WorkManager` on the receiver. The `WorkManager` is responsible for executing units of work in a thread. It has a thread pool that allows threads to be reused and ensures that only a prescribed number of threads will be spawned.

The `WorkManager` is an implementation of [org.mule.api.context.WorkManager](#), which is really just a wrapper of [javax.resource.spi.work.WorkManager](#) with some extra lifecycle methods. There is a `getWorkManager()` method on the [AbstractMessageReceiver](#) that you can use to get a reference to the `WorkManager` for the receiver. Work items (such as the code to execute in a separate thread) must implement [javax.resource.spi.work.Work](#). This interface extends `java.lang.Runnable` and thus has a `run()` method that will be invoked by the `WorkManager`.

When scheduling work with the `WorkManager`, you should call `scheduleWork(...)` on the `WorkManager` rather than `startWork(...)`.

## Message Dispatchers

Whereas a message receiver is equivalent to a server for the transport in that it serves client requests, a message dispatcher is the client implementation of the transport. Message dispatchers are responsible for making client requests over the transport, such as writing to a socket or invoking a web service. The [AbstractMessageDispatcher](#) provides a good base implementation, leaving three methods for the custom `MessageDispatcher` to implement.

### Methods to Implement

Method Name	Description	Required
doSend(MuleEvent)	Sends the message payload over the transport. If there is a response from the transport, it should be returned from this method. The <code>sendEvent</code> method is called when the endpoint is running synchronously, and any response returned will ultimately be passed back to the caller. This method is executed in the same thread as the request thread.	Yes

doDispatch(MuleEvent)	Invoked when the endpoint is asynchronous and should invoke the transport but not return any result. If a result is returned, it should be ignored, and if the underlying transport does have a notion of asynchronous processing, that should be invoked. This method is executed in a different thread from the request thread.	Yes
doConnect()	Makes a connection to the underlying transport, such as connecting to a socket or registering a SOAP service. When there is no connection to be made, this method should be used to check that resources are available. For example, the <code>FileMessageDispatcher</code> checks that the directories it will be using are available and readable. The <code>MessageDispatcher</code> should remain in a 'stopped' state even after the <code>doConnect()</code> method is called.	Yes
doDisconnect()	Disconnects and tidies up any resources that were allocated by the <code>doConnect()</code> method. This method should return the <code>MessageDispatcher</code> into a disconnected state so that it can be connected again using the <code>doConnect()</code> method	Yes
doDispose()	Called when the Dispatcher is being disposed and should clean up any open resources.	No

## Message Requesters

As with message receivers and dispatchers the implementation of a message requester for a transport, if it even applies, will vary greatly. The abstract `AbstractMessageRequester` provides a base from which to extend and implement your own Message Requester and implemented methods for routing events. Although requesters can implement `doConnect` and `doDisconnect` methods given the nature of a requester this can also be done as part of the `doRequest` implementation, it really depending on the underlying transport and if you need to maintain a connection open all the time or not to be able to make arbitrary requests.

Method Name	Description	Required
doRequest(long)	Used to make arbitrary requests to a transport resource. If the timeout is 0, the method should block until a message on the endpoint is received.	
doConnect()	Should make a connection to the underlying transport if required, such as to connect to a socket..	No
doDisconnect()	Disconnects and tidies up any resources allocated using the <code>doConnect()</code> method. This method should return the <code>MessageReceiver</code> in a disconnected state so that it can be connected again using the <code>doConnect()</code> method.	No
doInitialise()	Called when the Requester is being initialized after all properties have been set. Any required initialization can be done here.	No
doStart()	Called when the Requester is started. Any transport specific implementation that is required when the requestor is started should be implemented here.	No
doStop()	Called when the Requester is stopped. Any transport specific implementation that is required when the requestor is stopped should be implemented here.	No
doDispose()	Called when the Requester is being disposed and should clean up any open resources.	No

## Threads and Dispatcher Caching

Custom transports do not need to worry about dispatcher threading. Unless threading is turned off, the Dispatcher methods listed above will be executed in their own thread. This is managed by the `AbstractMessageDispatcher`.

When a request is made for a dispatcher, it is looked up from a dispatcher cache on the `AbstractConnector`. The cache is keyed by the endpoint being dispatched to. If a Dispatcher is not found, one is created using the `MessageDispatcherFactory` and then stored in the cache for later.

## Message Factories

Message factories translate messages from the underlying transport format into a `MuleMessage`. Almost all messaging protocols have the notion of message payload and header properties. Message factories extract that payload and optionally copy all properties of the transport message into the `MuleMessage`. A `MuleMessage` created by a message factory can be queried for properties of the underlying transport message. For example:

```
//JMS message ID
String id = (String)message.getProperty("JMSMessageID");

//HTTP content length
int contentLength = message.getIntProperty("Content-Length");
```

Note that the property names use the same name that is used by the underlying transport; `Content-Length` is a standard HTTP header name, and `JMSMessageID` is the equivalent bean property name on the `javax.jms.Message` interface.

A message factory should extend `org.mule.transport.AbstractMuleMessageFactory`, which implements much of the mundane methods needed by the `org.mule.api.transport.MuleMessageFactory` interface.

### Methods to Implement

Method Name	Description	Required
<code>extractPayload()</code>	Returns the message payload 'as is'.	Yes
<code>addProperties()</code>	Copies all properties of the transport message into the <code>DefaultMuleMessage</code> instance that is passed as parameter.	No
<code>addAttachments()</code>	Copies all attachments of the transport message into the <code>DefaultMuleMessage</code> instance that is passed as parameter	No

## Service Descriptors

Each transport has a service descriptor that describes what classes are used to construct the transport. For complete information, see [Transport Service Descriptors](#).

## Coding Standards

Following are coding standards to use when creating transports.

### Package Structure

All Mule transports have a similar package structure. They follow the convention of:

org.mule.transport.<protocol>

Where protocol is the protocol identifier of the transport such as 'tcp' or 'soap'. Any transformers and filters for the transport are stored in either a 'transformers' or 'filters' package under the main package. Note that if a transport has more than one implementation for a given protocol, such as the Axis and CXF implementations of the SOAP protocol, the package name should be the protocol, such as `soap` instead of `axis` or `cxf`.

### Internationalization

Any exceptions messages used in your transport implementation should be stored in a resource bundle so that they can be [internationalized](#). The message bundle is a standard Java properties file and must be located at:

META-INF/services/org/mule/i18n/<protocol>-messages.properties

Your Rating:  Results:  3 rates

## Transport Archetype

### Transport Archetype

[ [Configuring Maven](#) ] [ [Using the Archetype](#) ] [ [The Questions Explained](#) ] [ [Example Console Output](#) ] [ [Command Line Options](#) ]

Mule provides Maven archetypes that you can use as code templates for your Mule projects. These templates include a set of implementation notes and "todo" pointers that help you get started quickly. The Mule transport archetype will help you generate a tailored boilerplate transport project in seconds. For more information on Maven, see [Using Maven](#).

If you want to update an existing transport instead of creating a new one, such as adding new schema namespaces and registry bootstrapping to the transport, use the [Creating Module Archetypes](#) instead.

Follow the instructions below to create template files for a new transport, including all the necessary Java boilerplate and detailed implementation instructions in comments.

## Configuring Maven

Add the following to the file `settings.xml` (usually in your Maven `conf` or `$HOME/.m2` directory) so that Maven will allow you to execute Mule plug-ins.

```
settings.xml

<settings>
  <pluginGroups>
    <pluginGroup>org.mule.tools</pluginGroup>
  </pluginGroups>
  ...
</settings>
```

## Using the Archetype

First, open a command shell and change to the directory where you want to create your project.

```
> cd yourDir
```

Next, you execute the archetype and generate the code. If this is your first time running this command, Maven will download the archetype for you.

```
> mvn mule-transport-archetype:create -DtransportId=xxx -DmuleVersion=3.1.1
```



### Running the archetype

Maven uses by default the latest available version of the archetype. This can cause problems if you want to create a transport for an earlier version of Mule. In this case, run the `mule-transport-archetype` specifying the full version of the plugin like this:

```
mvn org.mule.tools:mule-transport-archetype:3.1.1:create ...
```

At minimum, you pass in two system parameters:

- `transportId`: The short name for the project (such as 'tcp'). This must be a single word in lower case with no spaces, periods, hyphens, etc. For transports this is usually the short protocol name of the underlying transport being connected.
- `muleVersion`: The version of the Mule project archetype you want to use. This will also be the default Mule version used for the generated artifact.

The plug-in will ask various questions (described below) and then generate the files. You can also use this plug-in without user prompts by entering all the arguments at the command line. For a full list of arguments that can be passed in, see the [Command Line Options](#).

After you have answered all the questions, the archetype creates a directory using the transport name you specified. The directory includes a POM file for building with Maven, a Mule configuration file (`src\main\resources\mule-config.xml`) that includes the namespaces for the transports and modules you specified and has placeholder elements for creating your first service, and a `package.html` file under `src\main\java` using the package path you specified. Lastly, it creates some template files under `src\test` to help you get started creating a unit test for the transport. A new `MULE-README.txt` file will be created in the root of your project explaining what files were created.

## The Questions Explained

The plug-in prompts you to answer several questions about the transport you are writing. These may vary according to the options you select. An example of the output is shown [below](#).

### Provide a description of what the transport does:

You should provide an accurate description of the transport with any high-level details of what you can or cannot do with it. This text will be used where a description of the transport is required.

### Which version of Mule is this transport targeted at?

The version of Mule you want to use for your transport. By default this will default to the version passed in on the command line.

### Will this project be hosted on MuleForge?

If the transport is going to be hosted on [MuleForge](#), additional information will be added to your project for linking to its issue tracker, web site, build server and deployment information.

**Will this transport have a custom schema for configuring the transport in XML?**

All new transports should define an XML schema that defines how the transport is configured. If you do not use this option, users will have to use generic configuration to use your transport.

**Can the transport receive inbound requests?**

Can this transport receive inbound events? For example, the File transport allows you to listen for files written to a directory. JMS allows you to listen for events being written to a topic or queue.

**Does the Message Receiver need to poll the underlying resource?**

To receive a message, some transports must do polling. For example, the File transport must poll a directory to know something has been written there, whereas JMS provides a callback (`MessageListener`) to deliver the message. This question is asked only if the transport can receive inbound requests.

**If this transport will have a default inbound transformer, enter the name of the transformer?**

If the protocol of the application being connected to has its own message type, you can define a default inbound transformer that will be invoked by default when defining endpoints that use this transport. You enter the name of the transformer class (without package name) to generate, such as `JmsMessageToObject`.

**Can the transport dispatch outbound requests?**

Asks whether messages can be written to this transport. With the File transport, you can write file data to a directory, and with JMS you can write to a queue or topic.

**If this transport will have a default outbound transformer, enter the name of the transformer?**

If the protocol of the application being connected to has its own message type, you can define a default outbound transformer that will be invoked by default when defining outbound endpoints that use this transport. You enter the name of the transformer class (without package name) to generate, such as `ObjectToJmsMessage`.

**Does the transport need a custom MuleMessageFactory?**

This is usually only required if the underlying transport has an API that has a message object i.e. `JMSMessage` or `HttpServletRequest`.

**Can the transport request individual messages from the underlying resource?**

If the transport can request messages from a message channel or resource rather than subscribing to inbound events or polling a resource, answer yes to this question. This will generate a `MessageRequester` class.

**Does this transport support transactions?**

If the underlying resource for this transport is transactional, you can have Mule generate a transaction wrapper that will allow users to enable transactions on endpoints defined using this transport.

**Does this transport use a non-JTA transaction manager?**

Not all technologies (such as JavaSpaces) support the standard JTA transaction manager. Mule can still work with different non-JTA transaction managers, and this archetype can generate the necessary stubs for you.

**What type of endpoints does this transport use?**

Mule supports a number of well-defined endpoints

- Resource endpoints (e.g., `jms://my.queue`)
- URL endpoints (e.g., `http://localhost:1234/context/foo?param=1`)
- Socket endpoints (e.g., `tcp://localhost:1234`)
- Custom

The Custom option allows you to deviate from the existing endpoint styles and parse your own.

**Which Mule transports do you want to include in this project?**

If you are extending one or more existing transports, specify them here in a comma-separated list.

### Which Mule modules do you want to include in this project?

By default, the Mule client module is included to enable easier testing. If you want to include other modules, specify them here in a comma-separated list.

### Example Console Output

```
*****
Provide a description of what the transport does: [default: ]
*****
[INFO] muleVersion:
*****
Which version of Mule is this transport targeted at? [default: 3.1.1]
*****
[INFO] forgeProject:
*****
Will this project be hosted on MuleForge? [y] or [n] [default: y]
*****
[INFO] hasCustomSchema:
*****
Will this transport have a custom schema for configuring the transport in Xml? [y] or [n] [default: y]
*****
[INFO] hasReceiver:
*****
Can the transport receive inbound requests? [y] or [n] [default: y]
*****
[INFO] isPollingReceiver:
*****
Does the Message Receiver need to poll the underlying resource? [y] or [n] [default: n]
*****
[INFO] inboundTransformer:
*****
If this transport will have a default inbound transformer, enter the name of the transformer? (i.e. JmsMessageToObject) [default: n]
*****
[INFO] hasDispatcher:
*****
Can the transport dispatch outbound requests? [y] or [n] [default: y]
*****
[INFO] outboundTransformer:
*****
If this transport will have a default outbound transformer, enter the name of the transformer? (i.e. ObjectToJmsMessage) [default: n]
*****
[INFO] hasCustomMessageFactory:
*****
Does the transport need a custom MuleMessageFactory? [y] or [n] (This is usually only required if the underlying transport has an API that has a message object i.e. JMSMessage or HttpServletRequest) [default: n]
*****
[INFO] hasRequester:
```

```

Can the transport request incoming messages programmatically? [y] or [n]
[default: y]
*****
[INFO] hasTransactions:
*****



Does this transport support transactions? [y] or [n]
[default: n]
*****
[INFO] hasCustomTransactions:
*****



Does this transport use a non-JTA Transaction manager? [y] or [n]
(i.e. needs to wrap proprietary transaction management)
[default: n]
*****
[INFO] endpointBuilder:
*****



What type of endpoints does this transport use?
- [r]esource endpoints (i.e. jms://my.queue)
- [u]rl endpoints (i.e. http://localhost:1234/context/foo?param=1)
- [s]ocket endpoints (i.e. tcp://localhost:1234)
- [c]ustom - parse your own
[default: r]
*****
[INFO] transports:
*****



Which Mule transports do you want to include in this project? If you intend extending a
transport you should add it here:

(options: axis,cxf,ejb,file,ftp,http,https,imap,imaps,jbpm,jdbc,
jetty,jms,multicast,pop3,pop3s,quartz,rmi,servlet,smtp,
smtps,servlet,ssl,tls,stdio,tcp,udp,vm,xmpp):
[default: vm]
*****



[INFO] modules:
*****



Which Mule modules do you want to include in this project? The client is added f
or testing:

(options: builders,client,jaas,jbossts,management,ognl,pgp,scripting,
spring-extras,sxc,xml):
[default: client]

```

\*\*\*\*\*

## Command Line Options

By default, this plug-in runs in interactive mode, but it's possible to run it in silent mode by using the following option:

```
-Dinteractive=false
```

The following options can be passed in:

Name	Example	Default Value
transportId	-DtransportId=tcp	none
description	-Ddescription="some text"	none
muleVersion	-DmuleVersion=3.1.1	none
hasCustomSchema	-DhasCustomSchema=true	true
forgeProject	-DforgeProject=true	true
hasDispatcher	-DhasDispatcher=true	true
hasRequester	-DhasRequester=true	true
hasCustomMessageFactory	-DhasCustomMessageFactory=true	false
hasTransactions	-DhasTransactions=false	false
version	-Dversion=1.0-SNAPSHOT	<muleVersion>
inboundTransformer	-DinboundTransformer=false	false
groupId	-DgroupId=org.mule.transport.tcp	org.mule.transport.<transportId>
hasReceiver	-DhasReceiver=true	true
isPollingReceiver	-DisPollingReceiver=false	false
outboundTransformer	-DoutboundTransformer=false	false
endpointBuilder	-DendpointBuilder=s	r
hasCustomTransactions	-DhasCustomTransactions=false	false
transports	-Dtransports=vm,jms	vm
modules	-Dmodules=client,xml	client

Your Rating: 

Results:  3 rates

## Transport Service Descriptors

### Transport Service Descriptors

A service descriptor is a file containing properties that describes how the internals of a transport is configured, such as which dispatcher factory to use or which endpoint builder to use. The service descriptor file must have the same name as the protocol of the transport and must be stored in the META-INF directory.

```
META-INF/services/org/mule/transport/<protocol>.properties.
```

Following are the properties that can be set in a transport service descriptor.

Property	Description	Required
----------	-------------	----------

connector	The name of the default connector class to use. This must be an implementation of <a href="#">org.mule.api.transport.Connector</a> .	Yes
dispatcher.factory	The name of the dispatcher factory class to use. This must be an implementation of <a href="#">org.mule.api.transport.MessageDispatcherFactory</a> .	No (if inbound only)
requester.factory	The name of the requester factory class to use. <a href="#">org.mule.api.transport.MessageRequesterFactory</a> .	No
message.receiver	The name of the message receiver class to use. This must be an implementation of <a href="#">org.mule.api.transport.MessageReceiver</a> .	No (if inbound only)
transacted.message.receiver	The name of the message receiver class to use for transacted messages. Some transports implement a transacted message receiver separately, in which case the MessageReceiver class can be specified here so Mule knows which receiver to use when creating endpoints that are transacted. This must be an implementation of <a href="#">org.mule.api.transport.MessageReceiver</a>	No
xa.transacted.message.receiver	If the transport supports XA transactions, the name of the XA transacted message receiver implementation to use. Some transports implement an XA transacted message receiver separately, in which case the MessageReceiver class can be specified here so Mule knows which receiver to use when creating endpoints that are XA transacted. This must be an implementation of <a href="#">org.mule.api.transport.MessageReceiver</a> .	No
message.factory	The name of the message factory class to use for this connector. This must be an implementation of <a href="#">org.mule.api.transport.MuleMessageFactory</a> .	Yes
inbound.transformer	The default transformer to use on inbound endpoints using this transport if no transform has been explicitly set on the endpoint. The property is the class name of a transformer that implements <a href="#">org.mule.api.transformer.Transformer</a> .	No
response.transformer	The default transformer to use on inbound endpoints using this transport if no transformer has been explicitly set for the response message flow in Request/Response style messaging. The property is the class name of a transformer that implements <a href="#">org.mule.api.transformer.Transformer</a> .	No
outbound.transformer	The default transformer to use on outbound endpoints using this transport if no transform has been explicitly set on the endpoint. The property is the class name of a transformer that implements <a href="#">org.mule.api.transformer.Transformer</a> .	No
endpoint.builder	The class name of the endpoint builder used to parse the endpoint and create the URI. Mule provides a standard set of endpoint builders such as ResourceNameEndpointURIBuilder used by JMS and VM, SocketEndpointURIBuilder used by TCP, HTTP, and UDP, and UrlEndpointURIBuilder used by SOAP. Custom endpoint builders should extend <a href="#">org.mule.endpoint.AbstractEndpointBuilder</a> .	Yes
session.handler	The name of the session handler class to use for reading and writing session information to and from the current message. This must be an implementation of <a href="#">org.mule.api.transport.SessionHandler</a> .	No
inbound.exchange.patterns	Comma-separated list of supported exchange patterns for inbound endpoints.	Yes
outbound.exchange.patterns	Comma-separated list of supported exchange patterns for outbound endpoints.	Yes
default.exchange.pattern	Exchange pattern to use for an endpoint if none was configured	Yes

Your Rating: 

Results:  0 rates

## Creating Custom Routers

### Creating Custom Routers

Typically, you implement custom routing in Mule ESB with [filters](#), but occasionally you may need to implement a custom router.

#### Custom Outbound Routers

Outbound routers control how a message gets routed to a list of endpoints. For example, sometimes a message gets routed based on simple rules or business logic, whereas in other cases you may multicast a message to every router.

The easiest way to write an outbound router is to extend the [org.mule.routing.outbound.AbstractOutboundRouter](#) class:

```

import org.mule.routing.outbound.AbstractOutboundRouter;

public class CustomOutboundRouter extends AbstractOutboundRouter
{
    ...
}

```

There are two methods you must implement that control how messages will be routed through the system. First, you must implement the `isMatch` method. This determines if a message should be processed by the router.

For example, to route only messages that have a payload containing the string "hello":

```

public boolean isMatch(MuleMessage message) throws RoutingException
{
    return "hello".equals(message.getPayloadAsString());
}

```

The second method you must implement is the `route` method. Each outbound router has a list of endpoints that are associated with it. The `route` method contains the logic to control how the event is propagated to the endpoints.

For example, if there were two endpoints you want to route to based on a condition, you would use this method to select the endpoint:

```

MuleEvent route(MuleEvent event) throws MessagingException
{
    OutboundEndpoint ep = null;
    if (isConditionMet(event))
    {
        ep = getRoutes().get(0);
    }
    else
    {
        ep = getRoutes().get(1);
    }
    ...
}

```

Once you've selected an endpoint, you must call `process()` on it.

Your Rating:  Results:  2 rates

## Deploying Mule ESB 3

### Deploying Mule ESB 3

As of version 3.1, Mule standalone can run multiple applications. This means you can include the same name spaces within different applications and they will neither collide nor share information.

- In effect, it's now an app server.
- In addition, Mule checks for new apps (in the `apps` directory) and will start them.
- It will also restart apps if it detects that they have changed, meaning you can drop a revised version of an existing, running application into the `/apps` directory, Mule will stop the existing app and restart using the new files.
- This results in one fewer reason to run Mule inside a container.
  
- Deployment Scenarios
- Mule Deployment Model
- Configuring Logging
- Mule Server Notifications
- Profiling Mule
- Hardening your Mule Installation
- Mule High Availability

- Configuring Mule for Different Deployment Scenarios

Your Rating:  Results:  0 rates

## Deployment Scenarios

### Deployment Scenarios

[ [Running Mule in Standalone Mode](#) ] [ [Embedding Mule in a Java Application or Webapp](#) ] [ [Using Spring](#) ]

There are several ways in which you can deploy Mule ESB. This page describes the various approaches.

Deployment Mode	Container	Pros	Cons	HA	MMC
Standalone	Mule ESB	Mule Deployment Model, Run multiple apps, Hot Deployment, Minimal Resource Requirements, Easy to Install	Mule must be provisioned into your environment	Mule HA Supported	Supported
Embedded WAR	App Server	Run multiple apps, All dependencies contained in webapp, Use Webapp deployment model	Each web app runs own mule instance, Increases size of webapp	Use App Server HA	Supported*
Embedded Java	Java App / IDE	No external container required	No Hot Deployment	Not Supported	Supported*

\* When Mule is embedded in some other container, the management console cannot perform auto-discovery or server restarts.

### Running Mule in Standalone Mode

The recommended approach is to run Mule ESB standalone from the command prompt, as a service or daemon, or from a script. This is the simplest architecture, so it reduces the number of points where errors can occur. It's typically best for performance as well, since it reduces the number of layers and eliminates the inherent performance impact of an application server on the overall solution. With Mule 3, you can also now run multiple applications side by side in a Mule instance using the new deployment model can support live deployment and hot redeployment of applications. Lastly, standalone mode has full support for the [Mule High Availability](#) module and the [Mule management console](#).

For more information on running Mule standalone, see [Mule Deployment Model](#).

### Embedding Mule in a Java Application or Webapp

You can start and stop Mule from a Java application or embed it in a Webapp (such as a JSP or servlet). For details, see [Embedding Mule in a Java Application or Webapp](#). Following are details on specific application servers:

- Geronimo: The Geronimo application server uses ActiveMQ as its default JMS provider. For details, see [ActiveMQ Integration](#).
- JBoss
- WebLogic
- WebSphere
- Tomcat

For details on how you can support Mule hot deployment within some application servers, see [Application Server Based Hot Deployment](#).

Note that when you embed Mule, the Mule Deployment Model and Mule HA high availability is not supported. Furthermore, because the application server needs control of Mule, the Mule Management Console control/management capabilities are reduced. The monitoring functions work, but server restart is not supported.

### Using Spring

Mule fully integrates with Spring, allowing you to take advantage of Spring's many features, including support for JNDI and EJB session beans. You can also use Spring remoting to access Mule from an external applications. For details, see [Using Mule with Spring](#).

Your Rating:  Results:  0 rates

## Choosing the Right Topology

### Choosing the Right Topology

As described in [Integrating Mule into Your Environment](#) in the Getting Started guide, you can deploy Mule ESB in many different topologies. As you build your Mule application, it is important to think critically about how best to architect your application to achieve the desired availability, fault tolerance, and performance characteristics. This page outlines some of the solutions for achieving the right blend of these characteristics through clustering and other techniques. There is no one correct approach for everyone, and designing your system is both an art and a science. If you need more assistance, MuleSoft Professional Services can help you by reviewing your architecture plan or designing it for you. For more information, [contact us](#).

## About Clustering

Clustering an application is useful for achieving the following:

- High availability (HA): Making your system continually available in the event that one or more servers or a data center fails.
- Fault tolerance (FT): The ability to recover from failure of an underlying component. Typically, the recovery is achieved through transaction rollback or compensating actions.
- Scaling an application horizontally to meet increased demand.

This page describes several possible clustering solutions.

## Mule High Availability

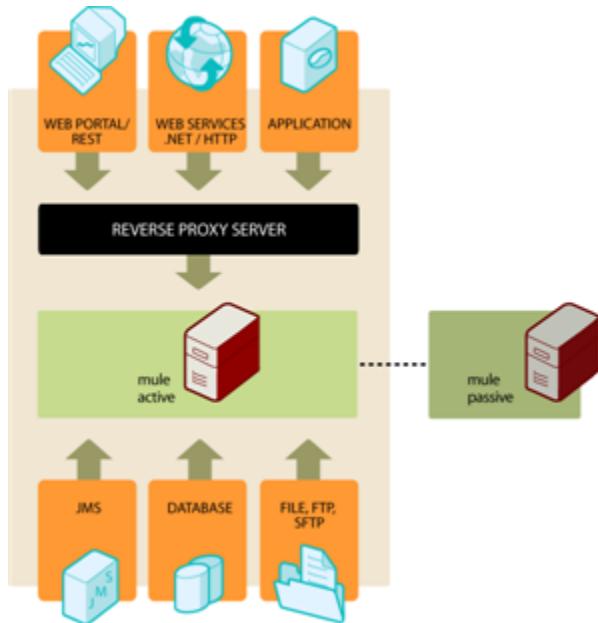
Mule High Availability provides basic failover capability for Mule. When the primary Mule instance becomes unavailable (e.g., because of a fatal JVM or hardware failure or it's taken offline for maintenance), a backup Mule instance immediately becomes the primary node and resumes processing where the failed instance left off. After a system administrator has recovered the failed Mule instance and brought it back online, it automatically becomes the backup node.

Seamless failover is made possible by a distributed memory store that shares all transient state information among clustered Mule instances, such as:

- SEDA service event queues
- In-memory message queues

Mule High Availability is currently available for the following transports:

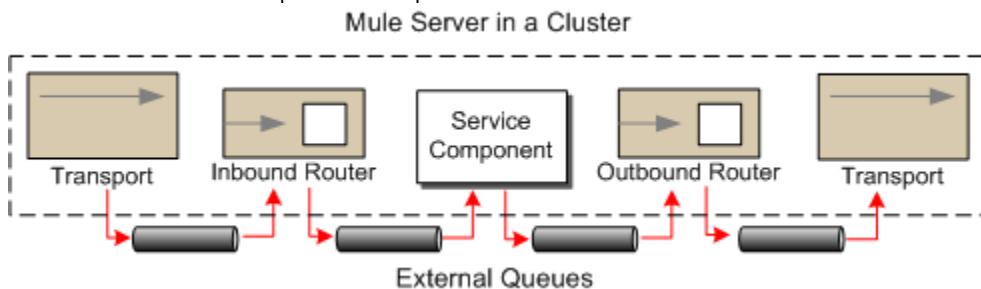
- HTTP (including CXF Web Services)
- JMS
- WebSphere MQ
- JDBC
- File
- FTP
- Clustered (replaces the local VM transport)



Mule High Availability is available with Mule Enterprise subscriptions. For details including technical highlights, requirements, and installation details, [contact us](#).

## JMS Queues

JMS can be used to achieve HA & FT by routing messages through JMS queues. In this case, each message is routed through a JMS queue whenever it moves from component to component.



Pros:

- Easy to do
- Well understood by developers

Cons:

- Requires lots of transactions, and transactions can be complicated
- Performance hit if you're using XA

## Load Balancers

Load balancers simply route requests to different servers based on the current load of each server and which servers are currently up. Load balancers can be software or hardware based. This approach is commonly used with clustered databases (see below).

Pros:

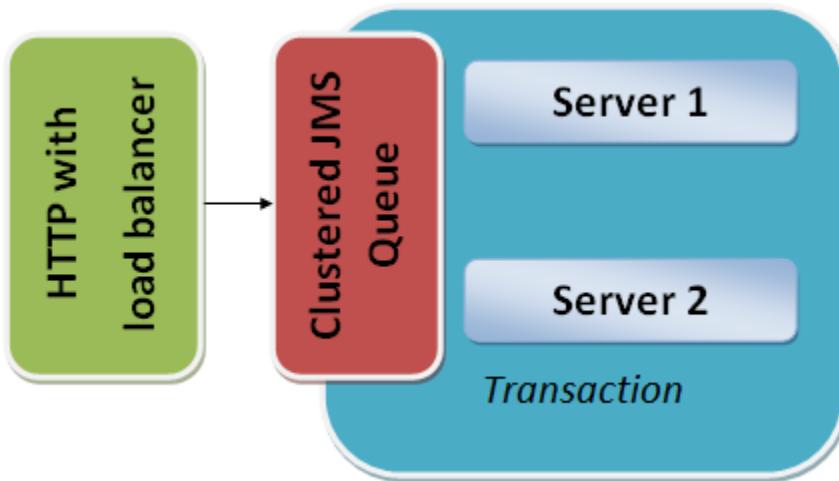
- Straightforward to do
- Well understood by developers

Cons:

- Not a complete solution on its own, doesn't provide fault tolerance.

## Example

In this example architecture, HTTP requests come in through a load balancer and are immediately put on a JMS queue. The JMS queue is clustered between the different servers. A server will start processing a message off the JMS queue and wrap everything in a transaction.



If the server goes down, the transaction will roll back and another server will pick up the message and start processing it.

**Note:** If the HTTP connection is open for the duration of this process, this approach will not work, as the load balancer will not transparently switch connections between servers. In this case, the HTTP client will need to retry the request.

## Maintaining State in a Database

If you have a clustered database, one option for your application is to simply store all state in the database and rely on it to replicate the data consistently across servers.

Pros:

- Straightforward to do
- Well understood by developers

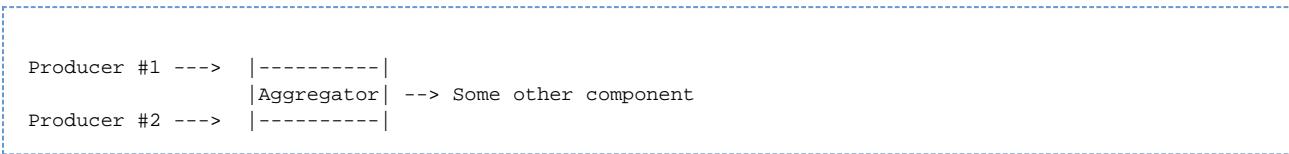
Cons:

- Not all state is amenable to being stored in the database

## Handling Stateful Components

While most applications can be supported by the above techniques, some require sharing state between JVMs more deeply.

One common example of this is an aggregator component. For example, assume you have an aggregator that is aggregating messages from two different producers. Producer #1 sends a message to the aggregator, which receives it and holds it in memory until Producer #2 sends a message.



If the server with the aggregator goes down between Producer #1 sending a message and Producer #2 sending a message, Producer #2 can't just send its message to a different server because that server will not have the message from Producer #1.

The solution to this is to share the state of the aggregator component across different machines through clustering software such as Terracotta, Tangosol Coherence, JGroups, etc. By using one of these tools, Producer #2 can simply fail over to a different server. Note that MuleSoft has not tested Mule with these tools and does not officially support them.

Pros:

- Works for all clustering cases
- Can work as a cache as well

Cons:

- Not officially supported by MuleSoft
- Requires performance tuning to get things working efficiently

## Related Topics

As you design your topology, there are several other topics to keep in mind that are beyond the scope of this documentation:

- Maintaining geo-distributed clusters
- Data partitioning
- ACID vs. BASE transactions
- Compensation and transactions

Your Rating: Results: 0 rates

## Embedding Mule in a Java Application or Webapp

### Embedding Mule in a Java Application or Webapp

This page describes how to start and stop Mule from a Java application or to embed it in a Webapp (such as a JSP or servlet), and how to interact with Mule from your code in both scenarios.

#### Starting Mule from a Java Application

To start Mule from any Java application, you can call one of its configuration builders. To use Mule XML configuration:

```
DefaultMuleContextFactory muleContextFactory = new DefaultMuleContextFactory();
SpringXmlConfigurationBuilder configBuilder = new SpringXmlConfigurationBuilder("mule-config.xml");
muleContext = muleContextFactory.createMuleContext(configBuilder);
```

Make sure you store a reference to the MuleContext, as you will need it to stop Mule.

If you have multiple configuration files, you can provide a comma-separated list or an array of configuration files:

```
SpringXmlConfigurationBuilder configBuilder =
    new SpringXmlConfigurationBuilder(new String[] { "mule-config.xml", "another-config.xml" });
```

You then call the start method to start the server:

```
muleContext.start();
```

### ***Stopping Mule from a Java Application***

To stop Mule, you stop its context like this:

```
muleContext.stop();
muleContext.dispose();
```

### ***Embedding Mule in a Webapp***

To embed Mule inside a webapp, you provide one or more configuration file locations as context params and include a context listener to initialize the Mule Server. If you are using Mule XML configuration, use the following -

```
<context-param>
    <param-name>org.mule.config</param-name>
    <param-value>mule-config-main.xml,mule-components.xml</param-value>
</context-param>

<listener>
    <listener-class>org.mule.config.builders.MuleXmlBuilderContextListener</listener-class>
</listener>
```

The configuration parameter can be a classpath location or file location. You can also specify multiple configuration files on the classpath or on the file system.

### ***Interacting with Mule from Your Code***

To interact with the Mule server from your application, JSP, or servlet, you can use the [Mule Client](#).

```

// in your servlet init() method save servletConfig.getServletContext() to a field
MuleContext muleContext = servletContext.getAttribute(MuleProperties.MULE_CONTEXT_PROPERTY);
// create a client
MuleClient client = new MuleClient(muleContext);

// send a jms message asynchronously
client.dispatch("jms://my.queue", "some data", null);

// or to receive a pop3 message via a configured mailbox
MuleMessage message = client.receive("pop3://myInboxProvider", 3000);

// or synchronous send a inter-vm message
MuleMessage message2 = client.send("vm://my.object", "Some more data", null);

```

Your Rating: 

Results:  1 rates

## Deploying Mule to JBoss

### Deploying Mule to JBoss

The recommended approach for integrating with enterprise application deployed in JBoss is to deploy Mule as a standalone application and use the Mule EJB Transport Reference or components configured using spring EJB proxies (<http://static.springframework.org/spring/docs/2.5.x/reference/ejb.html>) using to integrate with your EJB's in JBoss. If you need to deploy Mule in the JBoss application server for other reasons (e.g., to use JBoss clustering), use the instructions on this page. Note that if you are using JBoss clustering, you cannot use stateful routers.

There are two main approaches you can take to deploying Mule to JBoss:

1. Simple WAR deployment: in this approach, you simply embed Mule in your application and build the WAR. Your custom implementation classes are part of the WAR.
2. EAR application: you can embed the WAR inside the EAR file and include additional files such as EJBs.

### Classloader Isolation

When JBoss comes to classloading, unless classloader isolation is specified, JBoss will first try to use its own classes for deployment and only when these are not found will it look for them in the libraries of the deployment file. Since the versions of the libraries used to load Mule are not the same as the ones used by JBoss, various errors such as ClassCastException can appear, so classloading isolation is important. Therefore, for best results, you should use classloader isolation in your JBoss configuration. For more information, see <http://wiki.jboss.org/wiki/ClassLoadingConfiguration>.

### JBoss MQ Configuration

For information on configuring a JBoss JMS connector, see [JBoss Jms Integration](#).

### Scenarios your User Application with Mule in JBoss

For this scenario, the deployment is very simple: you simply add your own JAR and WAR archives to you EAR file. Because everything will be deployed in the same EAR, all the classes required by both the user application and Mule share the same classloader. However sometimes other classes may be required that are not deployed within your EAR..

### Resolving Cross-dependencies

The situation becomes more complex when you want to deploy Mule-dependent code in a separate EAR file (for example, you have a custom transformer that extends Mule's `AbstractTransformer` class). The user EAR depends on the Mule libraries to be loaded to be able to load the custom transformer library, while Mule expects the user EAR to be loaded to be able to use the transformer class that is found in the user EAR. To solve these cross-dependencies, you can create a shared library (in another EAR file, perhaps) and specify the library in the `<loader-repository>` element of the `jboss-app.xml` file in both the Mule EAR and the user EAR. Mule Enterprise Edition users can see an example of this in the Knowledge Base article titled "Embedding in JBoss: How to Share Classes Between Your Mule EE EAR and Another Application".

Your Rating: 

Results:  2 rates

## Mule as MBean

### Mule as MBean

[ [Creating a Simple MBean](#) ] [ [Creating JBoss Service Descriptor](#) ] [ [Deploying MBean to JBoss](#) ] [ [Copy the Dependencies](#) ] [ [References](#) ]

An MBean is a named managed object representing a resource in an JMX environment. You can easily deploy an MBean with Mule by taking the following steps:

1. Create an MBean
2. Create service descriptor
3. Deploy MBean (as .sar) to application server
4. Copy dependencies to the service's classpath

This page describes these steps using the JBoss application server.

#### ***Creating a Simple MBean***

To create an MBean, you need an interface and an implementation:

```
package foo.mbean;

public interface FooServiceMBean {
    public String getBar();
    public void start();
    public void stop();
}
```

```

package foo.mbean;

import org.jboss.system.ServiceMBeanSupport;
import org.mule.config.spring.SpringXmlConfigurationBuilder;
import org.mule.api.MuleContext;
import org.mule.api.context.notification.ServerNotification;

public class FooService extends ServiceMBeanSupport implements FooServiceMBean {

    public String getBar() {
        return "bar";
    }

    public void start() {
        this.getLog().info("MBean being started");

        try {
            MuleContext context = new DefaultMuleContextFactory().createMuleContext
(new SpringXmlConfigurationBuilder("foo-config.xml"));
            context.registerListener(this);
            context.start();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
        this.getLog().info("MBean started");
    }

    public void stop() {
        this.getLog().info("MBean being stopped");

        try {
            if (context != null) {
                context.stop();
                context.dispose();
            }
            this.getLog().info("Done stopping Mule MBean Service!");
        }
        catch (Exception ex) {
            this.getLog().error("Stopping Mule caused and exception!", ex);
        }
    }
}

```

The extension of `ServiceMBeanSupport` is simply to provide you more control over the API provided by JBoss.

### **Creating JBoss Service Descriptor**

You must create a service descriptor and add it to to `META-INF/`. Following is a simple example:

```

<?xml version="1.0" encoding="UTF-8"?>
<server>
    <mbean code="foo.FooService" name="foo:service=Foo">
    </mbean>
</server>

```

### **Deploying MBean to JBoss**

Based on the examples above, your distribution looks like this:

```
./foo  
./foo/FooService  
./foo/FooServiceMBean  
./META-INF  
./META-INF/jboss-service.xml
```

Package the distribution either as a JAR, which you can then rename to a \*.sar that you will eventually extract, or as a directory called <dirName>.sar.

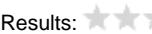
### **Copy the Dependencies**

Follow the steps below to copy the dependencies and complete the deployment:

1. Copy your <dirName>.sar/ directory to JBOSS\_HOME/server/default/deploy/.
2. Copy all dependencies of Mule, such as MULE\_HOME/lib/\*\*/\*.jar to the <dirName>.sar/ directory
3. Start JBoss. You will see the MBean appears in its MBean console.

### **References**

- [Wikipedia - MBean](#)
- [The Java Tutorials - Standard MBeans](#)

Your Rating:  Results:  0 rates

## **Deploying Mule to WebLogic**

### **Deploying Mule to WebLogic**

[ [Creating a WebLogic Domain for Mule](#) ] [ [Deploying Mule inside of a WebLogic web application:](#) ] [ [Overriding commons-lang](#) ] [ [Configuring Logging](#) ] [ [Deploying Mule](#) ] [ [Replacing the Mule Configuration File in the Vanilla RAR](#)  ]

For details on configuring WebLogic JMS in Mule, see [WebLogic JMS Integration](#). For an example of how one Mule user integrated Mule with WebLogic, see the cookbook entry [[Integration with BEA WebLogic|MULE3CB:Integration with BEA WebLogic](#)].

These instructions assume you have downloaded and installed WebLogic Application Server version 10.3. Note that WebLogic 8.x and 9.x are also supported, but these instructions are specific to version 10.3, both on BEA Weblogic and Oracle WebLogic.

**Note:** For an example of Mule running in a webapp under Weblogic, take the webapp example from the standalone distribution and make the changes described above.

### **Creating a WebLogic Domain for Mule**

The first step is to create a WebLogic domain for Mule using the BEA WebLogic Configuration Wizard.

1. Launch the Configuration Wizard. For example, on Windows choose **Start > All Programs > BEA Products > Tools > Configuration Wizard**.
2. In the Welcome screen, select the option to create a new domain and click **Next**.
3. In the Select Domain Source screen, select the option to generate a domain for WebLogic Server and click **Next**.
4. In the Configure Administrator Username and Password screen, enter the user name and password you want to use, and then click **Next**.
5. In the Configure Server Start Mode and JDK screen, select Development Mode, select the Sun SDK 1.5 instead of JRockit, and then click **Next**.
6. In the Customize Environment and Services Settings screen, leave **No** selected and click **Next**.
7. In the Create WebLogic Domain screen, enter **Mule2** for the domain name, leave the location set to the default user projects domains directory, and then click **Create**. Note that you can use whatever domain name you like, but the rest of this page assumes the name **Mule2**.
8. When the domain has been created, click **Done**.

### **Deploying Mule inside of a WebLogic web application:**

1. Override commons-lang as described in 'Overriding commons-lang'.
2. Configure logging as described in 'Configuring Logging'.



Your webapp must contain an updated version of commons-lang to run Mule

## Overriding commons-lang

Mule requires a newer version of Commons lang. You must copy the JAR file into your domain's lib folder and modify your classpath so it is used correctly by Mule and WebLogic.

1. Copy the commons-lang.jar file from your Mule distribution into <WLHome>/user\_projects/domains/<yourdomain>/lib
2. In the <WLHome>/user\_projects/domains/<yourdomain>/bin directory, modify the startWebLogic.sh/.cmd file as follows so that the commons-lang.jar in your lib directory is loaded first.  
Windows:

```
DOMAIN_HOME=<WLHome>/user_projects/domains/<yourdomain>
PRE_CLASSPATH=%DOMAIN_HOME%/lib/commons-lang-2.4.jar;%PRE_CLASSPATH%
```

Linux/OSX:

```
DOMAIN_HOME=<WLHome>/user_projects/domains/<yourdomain>
export DOMAIN_HOME
PRE_CLASSPATH=$DOMAIN_HOME/lib/commons-lang-2.4.jar:$PRE_CLASSPATH
export PRE_CLASSPATH
```

## Configuring Logging

Mule uses Log4j. WebLogic can be configured so that WebLogic's and Mule's logging both use log4j. For more information on using Commons logging and WebLogic, see [http://e-docs.bea.com/wls/docs103/logging/config\\_logs.html#using\\_log4j](http://e-docs.bea.com/wls/docs103/logging/config_logs.html#using_log4j).

1. Copy <WLHOME>/wlserver\_10.3/server/lib/wllog4j.jar to <WLHome>/user\_projects/domains/<yourdomain>/lib.
2. [Download the Log4j logging file](#).
3. Unzip the file and copy log4j.jar to <WLHome>/user\_projects/domains/<yourdomain>/lib.
4. [Download the Commons logging file](#).
5. Unzip the file and copy commons-logging.jar to <WLHome>/user\_projects/domains/<yourdomain>/lib.
6. Launch your WebLogic domain and log into the console.
7. In the left navigation pane, expand **Environment** and select **Servers**.
8. In the Servers table, click the name of the server instance whose logging you want to configure.
9. Select **Logging > General**, and click **Advanced** at the bottom of the page.
10. In the Logging Implementation list box, select **Log4j**.
11. Click **Save**.
12. Restart your WebLogic domain to activate the changes.

## Deploying Mule

There are many ways to deploy applications to the WebLogic server. These instructions demonstrate the two most common approaches: through auto-deployment, which provides a fast method for deploying for testing and evaluation, and through the Administration console, which provides more control over the configuration. Note that this section also applies when deploying an EAR or WAR that embeds Mule to WebLogic, in which case you deploy the EAR or WAR instead of the RAR file.

### To Auto-deploy Mule:

1. Copy mule-enterprise-jca.rar into the <WLHome>/user\_projects/domains/<yourdomain>/autodeploy directory.
2. Restart your domain server instance in development mode. During the starting process, the new RAR file will be auto-discovered and deployed by the domain server.

### To Deploy Mule Using the Administration Console:

1. Start the WebLogic server. For example, on Windows choose **Start > BEA Products > WebLogic Server**.
2. Start the Admin Server for the Mule2 domain. For example, on Windows you would choose **Start > BEA Products > User Projects > <yourdomain> > Start Admin Server for WebLogic Server Domain**.
3. When prompted, log in to the console using the user name and password you specified when creating the domain. If you close the console and need to restart it later, you can go to the URL <http://localhost:7001/console/console.portal>.
4. In the Domain Structure on the left, click **Deployments**.
5. Click **Install**, and then navigate to the location where you downloaded the Mule RAR file.
6. Select the RAR file and click **Next**.
7. Select **Install this deployment as an application**
8. Select **Next**

9. Select **Finish**
10. In the Change Center on the left, click **Activate Change**.

Mule is now deployed to WebLogic via the Mule JCA Resource Adapter. You must now replace the default configuration file in the RAR file with the configuration file for your Mule application.

## Replacing the Mule Configuration File in the Vanilla RAR

Mule includes a placeholder configuration file called `mule-config.xml` in the RAR file under `mule-module-jca-core.jar`. If you simply want to modify this file, you can do the following:

1. Unpackage the RAR and the JAR file.
2. Modify the configuration file.
3. Repackage the JAR and RAR with the updated file and copy the RAR into the `<WLHome>/user_projects/domains/<yourdomain>/autodeploy` directory.
4. Run the `startWebLogic` command.

If you want to use a different configuration file, do the following:

1. Unpackage the RAR file and copy your configuration file to the top level where all the JAR files are located.
2. Open the `META-INF` folder, and then open `weblogic-ra.xml` for editing.
3. Immediately after the `<enable-global-access-to-classes>true</enable-global-access-to-classes>` entry and right before `outbound-resource-adapter`, add the following lines, where `echo-axis-config.xml` is the name of your configuration file:

```
<properties>
  <property>
    <name>Configurations</name>
    <value>echo-axis-config.xml</value>
  </property>
</properties>
```

4. Repackage the RAR file and deploy it by copying it to the `autodeploy` directory and running `startWebLogic`.

Your Rating:  Results:  0 rates

## Deploying Mule to WebSphere

### Deploying Mule to WebSphere

[ [Preparing Your Mule Application as a Web Application](#) ] [ [Creating and Running a Mule Profile in WebSphere](#) ] [ [Deploying the Mule Application](#) ]

This page describes how to deploy Mule ESB to WebSphere Application Server 6.1.

### Preparing Your Mule Application as a Web Application

Before you can deploy your Mule application to WebSphere, you must prepare it as a web application. This involves creating a `web.xml` file for your Mule application and then creating a WAR file of all the supporting files.

#### ***Creating the web.xml File***

The `web.xml` file configures the Mule application as a web application by specifying your Mule configuration file and other options for starting Mule in the WebSphere container. It should look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4">
    <!--Mule configuration (Mule format)-->
    <context-param>
        <param-name>org.mule.config</param-name>
        <param-value>mule-config.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.mule.config.builders.MuleXmlBuilderContextListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>muleServlet</servlet-name>
        <servlet-class>org.mule.transport.servlet.MuleReceiverServlet</servlet-class>
        <load-on-startup>100</load-on-startup>
    </servlet>
    <!--Mule configuration ends-->
</web-app>

```

### **Creating the WAR File**

The WAR file must contain the `web.xml` file as well as the configuration files, libraries, and properties files for your Mule application. To create the WAR file, you can use the JAR command as follows in the root directory of your Mule application:

```
jar -cvf muleApp.war
```

The directory structure where you run the command should look similar to this:

```

META-INF
maven
WEB-INF
classes
    mule-config.xml
    log4J.properties
lib
    mule jars
web.xml

```

### **Creating and Running a Mule Profile in WebSphere**

Before you can deploy the Mule application, you must create a profile for Mule in WebSphere. Following are instructions for using the Profile Management Tool in Windows, but the steps for UNIX servers are similar.

1. From the Windows start menu, launch the Profile Management Tool from the WebSphere program group (e.g., **Start > IBM WebSphere > Application Server v6.1 > Profile Management Tool**).
2. When the profile management wizard starts, click **Next**.
3. Select **Application server** and click **Next**.
4. Select **Advanced profile creation** and click **Next**.
5. Select **Deploy the application console**, clear all other check boxes, and click **Next**.
6. Specify a profile name like "MuleProfile" and specify the directory where you want to save the profile. We recommend you specify a directory with the same name as the profile. Leave the other options unchecked and click **Next**.
7. If you want to enable security for this profile, enter the user name and password. Otherwise, clear the check box. Click **Next**.
8. Leave all the default port values (or change any required in your environment) and click **Next**.
9. Clear the check box to run the profile as a Windows service (on UNIX, clear the option to run as a daemon) and click **Next**.
10. Leave the web server definition option unchecked and click **Next**.
11. Click **Create**. The profile is created and available from the WebSphere Start menu.
12. Run the profile. For example, choose **Start > IBM WebSphere > Application Server v6.1 > Profiles > MuleProfile > Start the server**.

### **Deploying the Mule Application**

You are now ready to deploy your Mule application.

1. With the Mule profile running, open the WebSphere administrative console.
2. On the navigation bar on the left, click **Applications** and then click **Install New Application**.

3. For the path, browse to the location of the Mule WAR file you created above.
4. For the context root, type a unique root like "Mule".
5. Click **Next** on each of the screens and then **Finish**. Save your settings.
6. Go to Enterprise Applications, select `muleApp.war`, and then click **Start**.

Your Mule application is started as an embedded web application in WebSphere. To verify that it deployed correctly, check the `MuleProfile/logs/server1/SystemOut.log` file.

Your Rating:  5 stars

Results:  0 rates

## Deploying Mule as a Service to Tomcat

[ [Deploying Mule as a Service to Tomcat](#) ] [ [Installing Mule in Tomcat](#) ] [ [Copying the Mule Application Files](#) ] [ [Hot Deploying Mule Applications](#) ]

### Deploying Mule as a Service to Tomcat

This page describes how to install Mule as a service in Apache Tomcat and set up your Mule applications for hot deployment. For more information on hot deploying Mule applications, see [Application Server Based Hot Deployment](#).

#### Installing Mule in Tomcat

To install Mule in Tomcat so that you can hot deploy your Mule applications, you take the following steps:

1. Download and install Apache Tomcat 6 from the Apache web site following the standard installation instructions.
2. In the Tomcat home directory, add the following line to the `conf/server.xml` file:  
`<Listener className="org.mule.module.tomcat.MuleTomcatListener" />`
3. Copy the contents of the Mule `lib` folder with all its subdirectories **except** `/boot` to the `mule-libs/` directory under your Tomcat home directory (create one if necessary). You do not need to flatten the directories.
4. Copy the `mule-module-tomcat-<version>.jar` file to the `mule-libs/mule/` directory under your Tomcat home directory (if it is not there already).
5. Copy the following libraries from your Mule `lib/boot/` directory to your Tomcat `mule-libs/opt/` directory:
  - `jcl104-over-slf4j-1.5.0.jar`
  - `log4j-1.2.14.jar`
  - `slf4j-api-1.5.0.jar`
  - `slf4j-log4j12-1.5.0.jar`
6. In the Tomcat `conf/catalina.properties` file, add the following to `common.loader` (precede with a comma to separate it from existing values):
   
 `${catalina.home}/mule-libs/user/*.jar,${catalina.home}/mule-libs/mule/*.jar,${catalina.home}/mule-lib:`

#### Copying the Mule Application Files

After you package your configuration files and custom Java classes in a WAR file (see [Application Server Based Hot Deployment](#)), copy your Mule applications WAR files to the Tomcat `/webapps` directory.

#### Hot Deploying Mule Applications

After you have copied the WAR file for your Mule application to the Tomcat `/webapps` directory, Tomcat deploys it. If you need to make a change to a configuration or Java file in the webapp, simply modify the file in the exploded directory under the Tomcat `/webapps` directory, and then touch the `web.xml` file (or you can simply add and delete a space in the file and then save it) to trigger Tomcat to redeploy the application. Alternatively, you could modify the source files, repackage them as a WAR, and drop the new WAR into the `/webapps` directory to trigger Tomcat to redeploy the application.

Your Rating:  5 stars

Results:  3 rates

## Application Server Based Hot Deployment

### Deployment Model for Mule in Application Servers

[ [How it Works](#) ] [ [Preparing the Configuration File](#) ] [ [Deploying Mule as a Service](#) ] [ [Packaging the Mule Application](#) ] [ [Deploying the Mule Application](#) ]

If you are running Mule deployed in an Application Server, you have the option to configure Mule as a service of the container, which allows you to maintain a single instance of Mule but update your configuration files and/or transformers, filters, or component implementations without restarting Mule. When you hot deploy a Mule application, only that application will stop/start, and any other Mule applications that are deployed will continue to run as before. This is the same benefit you get when you run Mule standalone.

Deployment is currently tested and supported with Mule ESB 2.2.5 or later and the following application servers:

- JBoss 4.2.x (JBoss 5.x is not currently supported)
- Geronimo 2.1.x
- Tomcat 6.0.x
- MuleSoft Tcat Server 6

## How it Works

A Mule application is one or more configuration files packaged up with any dependencies that are not already provided by Mule. In a deployment scenario, each Mule application shares a *single Mule instance* running in an application server or web container. To hot deploy Mule applications, you do the following:

- Configure Mule to run as a service in the container (see instructions below)
- Package your Mule applications as WAR files so that you can deploy them as web applications

When you deploy the WAR file, Mule reads the Mule configuration file and creates and starts all the required objects just as when running Mule standalone. When you undeploy a Mule application, only the objects created and started for that Mule application are stopped and disposed of without affecting any other applications sharing the same Mule instance. In this way, multiple Mule applications can be deployed, undeployed, and redeployed independently with no downtime for any other services.

Two of the key advantages of using a single shared Mule instance running as a container server are:

- A greatly reduced memory footprint over using multiple Mule instances.
- The ability to share objects across applications. For example, you could have a single webapp that defines the connectors, filters, and transformers that are then used by the other webapps.

**Note:** If you share objects across applications, be sure to first deploy the webapp that creates the shared objects, and then deploy the remaining webapps that use those objects.

Below is a table listing the key differences in the two supported webapp deployment models for Mule ESB:

	Self-Contained Webapp	Container Service
Memory footprint	Larger	Smaller
Application isolation	More	Less
Shared resource management	None	Possible
Hot deployment	Entire Mule Application is deployed/undeployed	Only resources required for the application are deployed/undeployed

## Preparing the Configuration File

When defining the scope of your applications and their configurations, keep the following points in mind:

- Each Mule application uses an instance of Mule that is already started. Therefore, your application cannot modify anything in the `<configuration>` element, including attributes and child elements, which are required to be configured before Mule startup. If you need to modify these settings, you must modify them on the Mule instance and restart it.
- If you don't explicitly define the connector that should be used by an endpoint, and another web application that is already deployed defines a connector that supports the same protocol, that existing connector defined will be used.

## Deploying Mule as a Service

To deploy Mule as a service to Tomcat or Tcat Server, see [Deploying Mule as a Service to Tomcat](#).

## Packaging the Mule Application

Each Mule application you set up for hot deployment consists of one or more configuration files plus supporting custom classes, all packaged as a standard web application (WAR deployment archive file). You can create the WAR using your favorite IDE or build tool as you would create any standard web application.

Note: While these instructions describe WAR packaging, you could also deploy to JEE environments as an EAR.

To enable Mule to find and load your configuration, you must include a `web.xml` file that points to your configuration files and to the deployable listener, as shown below:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

    <display-name>MuleEchoExample</display-name>
    <description>Mule Echo Example</description>

    <context-param>
        <param-name>org.mule.config</param-name>
        <param-value>echo-cxf-config.xml</param-value>
    </context-param>

    <!--
        To use a Mule XML configuration file use this context listener
    -->
    <listener>
        <listener-class>org.mule.config.builders.DeployableMuleXmlContextListener</listener-class>
    </listener>

</web-app>

```

## Deploying the Mule Application

You deploy your Mule application WAR file in exactly the same way you deploy any web application on the application server or web container you are using. That is, you can use a web administration console, command-line deployment tool, Maven, or your IDE. For example, if you are deploying to Tomcat, you simply copy your WAR to the Tomcat `webapps` directory or use the Tcat console to upload and deploy it to your server. For more information, see the documentation for your application server or web container.

If your Mule applications share objects, be sure to first deploy the application that creates those objects, and then deploy the applications that use them.

Your Rating:  Results:  0 rates

## Classloader Control in Mule

### [Mule 3.2] Classloader Control in Mule

[ [Classloading in Mule](#) ] [ [Fine-Grained Classloading Control](#) ] [ [Mule Plugin System](#) ]

This topic introduces you to classloading in Mule and shows you how to override classloading in your applications and plugins.

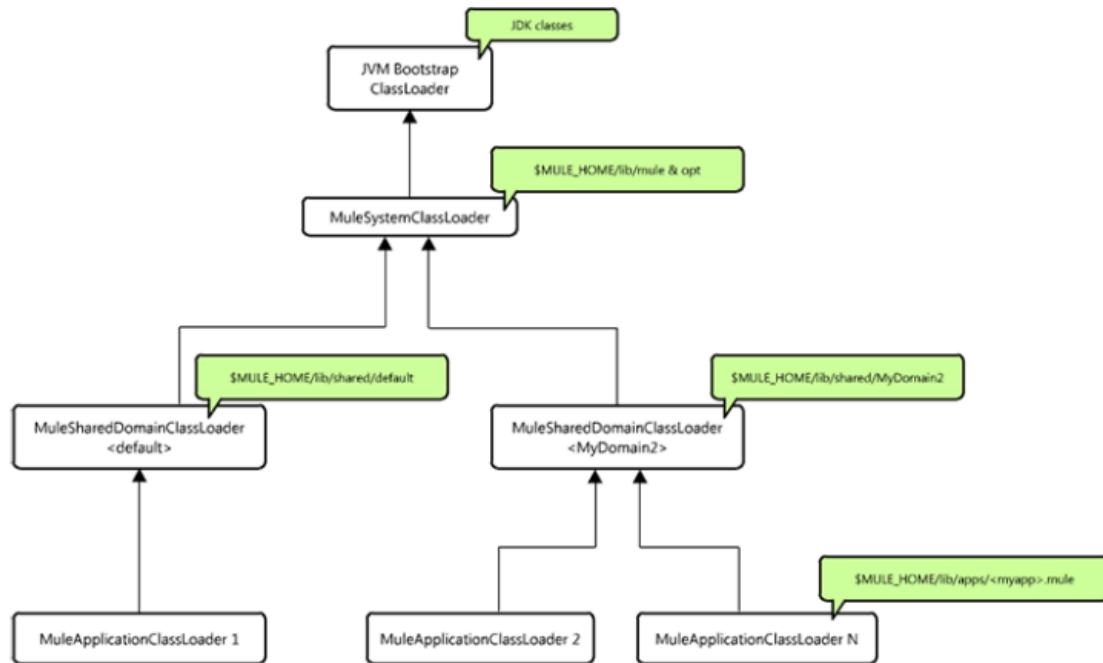
#### Classloading in Mule

Mule uses classloaders to find and load classes for execution. When Mule uses classloaders, it does so in the following order:

1. The bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine. This classloader loads the core Java libraries.
2. The Mule System class loader. This classloader loads standard Mule libraries, that is, libraries in the `<MULE_HOME>/lib` directory and subdirectories, where `<MULE_HOME>` is the directory where Mule ESB is installed.
3. One or more shared domain classloaders. These classloaders are used to share libraries between applications or allow an application to use a different version of a library. These libraries are in the `<MULE_HOME>/lib/shared/<domain>` library, where `<domain>` is the domain of the application. The shared domain classloader is optional, there is none by default.
4. One or more Mule application class loaders that load classes or libraries from a Mule application, that is, libraries in the `<MULE_HOME>/apps/<myapp>/lib` directory, where `<myapp>` is the name of the application.

The Mule classloader order is illustrated in the following diagram:

# Mule 3.x ClassLoader Architecture



Although this classloading architecture meets most classloading needs, there are times when you might need to override the default classloading scheme. For example, suppose an application requires a third-party library that is bundled with it. This might conflict with a library version that the shared domain classloader would load (that is, a version of the library already bundled with Mule). How do you ensure that the required version of the library is used with the application? To address requirements such as these, Mule 3.2.0 adds support for fine-grained class loading control that enables you to override default classloading.

## Fine-Grained Classloading Control

Mule introduces a new configuration property, `loader.override`, that enables you to override default classloading. You specify the property in the `mule-deploy.properties` file for the application, and identify a list of classes, packages, or both that will be used in overrides.

### mule-deploy.properties

```
loader.override=<comma-separated list of classes or packages>
```

Each class or package in the list is specified with its fully-qualified name. For example, an override list that includes a class (`com.example.MyProvider`) and a package (`com.sun.jersey`) would be specified as follows:

```
loader.override=com.example.MyProvider, com.sun.jersey
```

**⚠️** If a package is specified in the override list, it applies to all of its subpackages. For example, `com.sun.jersey` will also include `com.sun.jersey.impl`.

## Blocking

You can also block the loading of a class so that the class lookup is done in the application and not in Mule. You specify blocking similarly to the way you specify class overrides, that is, using `loader.override`. However, the fully qualified names of classes or packages in the list must be prefixed with a `-` (dash/minus sign). Here, is an example of a blocking specification:

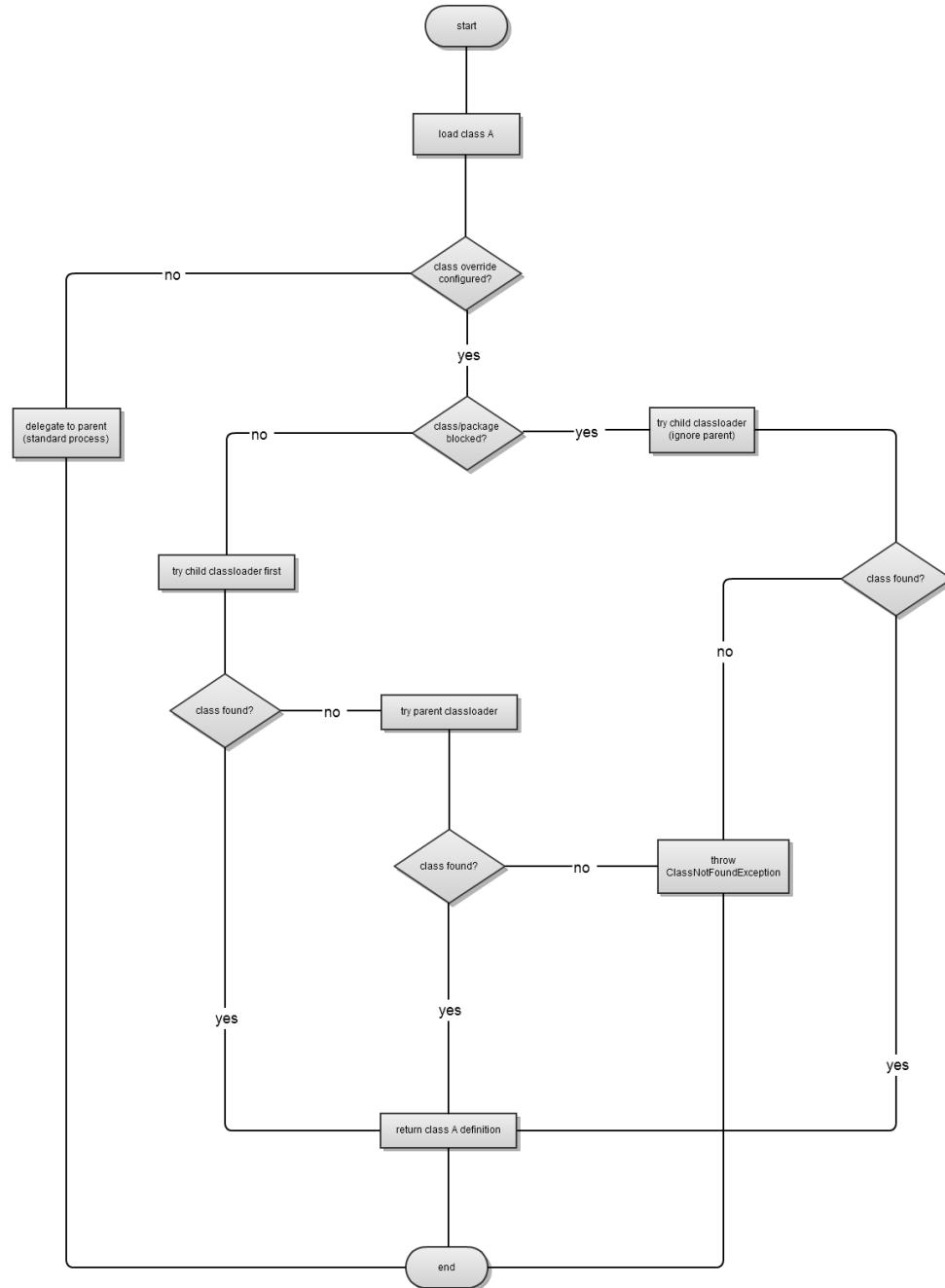
```
loader.override=-com.example.AnotherProvider
```

Blocking is a more advanced concept than class overriding, and many users will not need to consider it. But it is quite important for plugin

developers. The difference between blocking and class overriding is that if a class is specified in the blocking list, its lookup is performed within the application or plugin only. In other words, if the class is provided by Mule, it will not be visible in the application or plugin. A `ClassNotFoundException` is a valid outcome even if Mule does have such a class on a system level.

### **Classloading Override/Blocking Path Diagram**

Here is a diagram that illustrates how class loading is handled for class overrides and blocking. When you view the diagram note that every class loader has a parent class loader. Unless loader override is configured, a classloader first delegates the search for a class to its parent classloader before attempting to find the class itself (the classloader is the child of its parent classloader).



### **Mule Plugin System**

Mule now has a plugin system that allows applications to package the plugins they use, and optionally allows these plugins to control their class

loading in the same way that applications do.

Enabling a plugin to control classloading can be quite useful. For example, a cloud connector may bundle a third-party library that it's been tested with (that is, a library different from one that Mule provides) and declare a preference to load the third-party version. An application would just need to add a plugin that identifies the third-party library in a loader override. The loader override is specified as a property, `loader.override`, in the plugin structure (in [Plugin Layout](#) below).



Question for reviewers: In general, when/where are plugins useful? Also, where can users find information about developing plugins?

The module system also provides for a simple distribution format for Mule extensions with dependencies.

### **Application Layout**

Mule plugins are deployed as part of Mule applications. The Mule application structure has been extended to add a `plugins` top-level directory:

#### **Application Layout**

```
/-
  /classes
  /lib
  /plugins
  mule-deploy.properties (optional)
  mule-app.properties (optional)
```

Here are some things to consider regarding the plugins:

- Multiple plugins, such as cloud connectors, can be bundled in the application.
- The plugins are deployed in natural sort order by filename.
- Each plugin must be packaged as a zip file with a predefined structure (as shown in [Plugin Layout](#) below).
- Exploded plugins are not currently supported. The rationale for this is to encourage and maintain a simple plugin distribution model of just one file.
- The deployment lifecycle of the plugin is tied to the application. **[Reviewers: Is this true?]**

### **Plugin Layout**

The plugin layout is a simplified version of a Mule app.

#### **Plugin Layout**

```
/-
  /classes
  /lib
  plugin.properties (optional)
```

### **Plugin Properties**

The following plugin properties are currently recognized:

- **loader.override** - provides fine-grained control over class loading within a plugin. This is the same level of fine-grained class loader control that is available to an application (see [Fine-Grained Class Loading Control](#)). An application can indicate its classloading preference simply by including a plugin that specifies the `loader.override` property. The application does not need to do any extra classloader configuration. However, if necessary, the application can override any classloading preference the plugin declares.

Your Rating:

Results: 0 rates

## **Mule Deployment Model**

### **Mule Deployment Model**

Mule 3 introduces an application-level component model to Mule. With this new concept, there is now a well defined model for packaging your applications and deploying them into a Mule instance. In addition, Mule provides support for managing the lifecycle of applications, even supporting multiple applications running independently within the same Mule container. This allows additional advantages such as:

- Running multiple applications side-by-side
- Clear boundaries for operations on what a Mule application is
- New or modified applications can be deployed/undeployed with zero downtime
- Mule can monitor your applications and will reload configuration changes automatically
- Applications can depend on different library versions, even if they would conflict before
- Multiple versions of an application can exist side by side

Understanding the new model can be divided into several topics:

- Hot Deployment
- Application Deployment
- Application Format
- Deployment Descriptor

Your Rating: 

Results:  0 rates

## Hot Deployment

### Hot Deployment

[ [Outline of a Deployment](#) ] [ [How Hot Deployment works](#) ] [ [Hot Deployment Using Mule IDE](#) ] [ [Additional Features](#) ]

You can now modify your configuration files and custom classes and have them reloaded without having to restart Mule.

#### Outline of a Deployment

Here is a quick summary for deploying an app 'foo':

- Create a directory under \$MULE\_HOME/apps/foo
- Jar custom classes (if any), and put them under \$MULE\_HOME/apps/foo/lib
- Place the master Mule config file at \$MULE\_HOME/apps/foo/mule-config.xml (note that it has to be named `mule-config.xml`)
- Start your app with `mule -app foo`

As a bonus, application's master config file is monitored, so if there are any class changes you want to pick up or simply modify the config, save or touch `mule-config.xml` and Mule will hot-reload the application.

#### How Hot Deployment works

Mule checks every three seconds for updated configuration files under the \$MULE\_HOME/apps directory, and when it finds one, it reloads the configuration file and the JARs in that applications lib directory.

Therefore, if you want to change one of your custom classes, you modify and rejar it, copy the updated JAR to the lib directory, and then touch or save the configuration file. Currently, Mule checks only the first configuration file in your application's directory, so right now hot deployment works best with applications that have a single configuration file.

#### Hot Deployment Using Mule IDE

Starting with Mule IDE 2.1.1 hot deployment is automatically enabled when you create a project that is associated to a Mule 3 distribution. Read all about the hot deployment builder in the [Mule IDE user guide](#).

#### Additional Features

- Applications can depend on different library versions, even if they would conflict before
- Applications are now deployed with clear boundaries
- Multiple applications can be run side-by-side within a single instance of Mule

Your Rating: 

Results:  0 rates

## Application Deployment

### Application Deployment

#### Starting Mule

Start mule by running <MULE\_HOME>/bin/mule or starting Mule as a service. By default, applications in the <MULE\_HOME>/apps directory will

be deployed. You can also designate specific applications to start (separated by the colon - :) and Mule 3.0 will respect the order when starting the applications. In this scenario, only the applications specified will be started.

Start Mule by specifying an app to run:

```
mule -app foo
```

where `foo` is a Mule app at `$MULE_HOME/apps/foo`.

From this moment, Mule checks every three seconds for the `$MULE_HOME/apps/foo/mule-config.xml` updates. One can update the application jar contents and touch/modify this file to have Mule reload the config and class modifications.

## Deploying Applications

Mule applications, either zipped or exploded can be dropped into `$MULE_HOME/apps`. If Mule is already running, the application will be deployed dynamically.



All applications in Mule are unpacked at runtime and original zip removed. This means, e.g. that dropping a zip file into 'apps' dir will create a new folder with the same name (sans 'zip' extension) and delete the zip.

A successful app deployment is confirmed by:

- Having an unpacked application folder in the apps dir. E.g. for `stockTrader.zip`- `$MULE_HOME/apps/stockTrader`.
- An anchor file created for a running app, e.g. `$MULE_HOME/apps/stockTrader-anchor.txt`

If you wish to store your applications in a different location, you can do this on Unix-based systems by creating a symlink to your application directory from `$MULE_HOME/apps`

## Undeploying Applications

It is recommended one doesn't delete the application folder directly, but rather an app's anchor file only:

1. Prevents any interference from the hot-deployment layer and doesn't leave room for concurrent conflicting actions.
2. Avoids potential application jar locking issues on some operation systems and allows for clean shutdown and undeployment.

E.g. if the `stockTrader` app is running (app folder is there as well as the `$MULE_HOME/apps/stockTrader-anchor.txt` file, just delete the anchor file to have the app removed from the Mule instance at runtime. Application folder will be removed once the app terminates.

## Updating Applications at Runtime

Updating a Mule application at runtime can be a complex change involving class modifications, endpoint modifications (e.g. changing ports, etc), as well as reconfigured flows. As a result, any application update does a graceful app shutdown under the hood and reconfigures itself. In practice, this is pretty transparent to the user and happens within seconds.

There are several ways an application can be updated:

- By dropping the modifications over an existing exploded app folder and touching the 'master' configuration file (`mule-config.xml` in app root by default).
- By dropping a new **zipped** version of the app into `$MULE_HOME/apps` dir. Mule will detect this as an existing app update and will ensure a clean redeployment of the app. Note that any modifications to the old app folder are discarded - the new app folder is a clean exploded application from a zip.

As you see, both integrate pretty well with existing build tools, the preference for one over another really depends on established development practices only.

## Disabling the Mule Container Mode

If you want to run Mule 3 the legacy 2.x way, edit `$MULE_HOME/conf/wrapper.conf` file and replace the following line:

```
# Java Main class
wrapper.java.mainclass=org.mule.module.reboot.MuleContainerBootstrap
```

with this one:

```
# Java Main class  
wrapper.java.mainclass=org.mule.module.boot.MuleBootstrap
```

When run in this legacy mode, none of the new application deployment features apply

## Embedded Mule

When Mule is embedded in an application server, Java application, unit test, IDE, etc and started programmatically, the deployment functionality is disabled and Mule follows the legacy application model.

## Troubleshooting

If application fails to start (e.g. broken configuration file provided), Mule will **not** monitor the app for changes (as there technically is no application running). To update such an app, simply redeploy the app by dropping an updated archive into the apps folder.

Your Rating:  Results:  0 rates

## Application Format

### Application Format

[ General Definition ] [ Application structure ]

#### General Definition

A Mule application is either a:

- **Zip file.** Yes, that's a regular archive with a 'zip' extension.
- Unpacked version of the same zip (exploded app)

This deployment unit encapsulates everything an application would need to function, such as libraries, custom code, configuration, deployment descriptor and any environment properties accompanying the application. There is also a way to share libraries between applications via the **domain deployment descriptor** attribute, which provides for smaller application archives at the expense of a dependency on container being properly setup.

#### Application structure

A Mule *application* has a structured layout under \$MULE\_HOME/apps.

Mule will support both packaged and exploded deployment.

```
/  
  \- classes           // application-specific expanded resources (e.g. logging configuration  
  files, properties, etc  
    |- lib              // application-specific jars  
    |- mule-config.xml // Main Mule configuration file, also monitored for changes  
    |- mule-deploy.properties // Application deployment descriptor (optional)  
    |- mule-app.properties // custom properties to be added to the registry instance used by the  
    application (optional)
```

Your Rating:  Results:  0 rates

## Deployment Descriptor

### Deployment Descriptor

[ Options ] [ Classloading in Mule ]

Mule's deployment descriptor is a properties file (`mule-deploy.properties`) that controls how a Mule application should be deployed. A typical application, however, would rarely need to use many of the configuration properties available in the file, relying instead on defaults.

Mule checks if there is a deployment descriptor file in the application root and uses it if available (it's optional).

Here's the list of supported configuration properties in the deployment descriptor:

Name	Description	Default
domain	ClassLoader domain for this application. Typically used to share common libraries between applications and/or to allow use of different library version in applications. Maps directly to <code>\$MULE_HOME/lib/shared/&lt;domain&gt;</code> . For example, <code>stockDomain</code> maps to <code>\$MULE_HOME/lib/shared/stockDomain</code> .	"default"
config.resources	A Mule 2.x-style comma separated list of configuration files for this application. Typical use is to support split configurations declaratively. An alternative is to have a default <code>mule-config.xml</code> file import extra configuration pieces. Note that the first config piece is considered to be a 'master' and is monitored for redeployment, but not others.	<code>mule-config.xml</code> from the app root is used if nothing else is specified
redeployment.enabled	Allows explicitly disabling application hot-redeployment - configuration 'master' file will not be monitored for changes. Dropping a new version of the application archive in the <code>\$MULE_HOME/apps</code> will still redeploy the application. Any values other than <code>true</code> , <code>yes</code> , <code>on</code> are treated as <code>false</code> and disable the config change monitor.	Enabled by default
encoding	Default encoding, used throughout the application if none specified explicitly on, for example, a transformer	UTF-8
config.builder	Configuration builder to use for parsing the application config file.	AutoConfigurationBuilder
scan.packages (since Mule 3.1)	The application's classpath is no longer scanned for iBeans annotations ( <code>@Call</code> , <code>@Template</code> and <code>@IBeanGroup</code> ). The regular Mule annotations (for example, <code>@Transformer</code> still work). To reactivate classpath scanning, configure a comma-separated list of package names to scan for annotations at startup.	Empty by default
<b>[Mule 3.2]</b> loader.override	Overrides default class loading. The property value is specified as a comma-separated list of classes, packages, or both. Blocking can also be specified by preceding the classes or packages in the list with a - (dash/minus sign). If a class is specified in the blocking list, its lookup is performed within the application or plugin only, and not in Mule. For further details, see <a href="#">Classloader Control in Mule</a> .	Empty by default

Two things to note:

- Classloader domains can feature different versions of the same library for different sets of applications.
- Unless otherwise instructed, the default domain is used. You can specify a custom domain using the `domain` property in the application deployment descriptor.

## Options

An application may also contain a `mule-app.properties` file in the application root (right next to the `mule-deploy.properties` file). The `mule-app.properties` file is the place to put any custom properties for the application. Mule will make them available in the registry and they can be accessed in two ways:

- At application startup in the configuration file. Use a  `${foo}` placeholder (or any Mule expression which can get to the registry) to lookup a `foo` value.
- In the code (for example, by implementing `MuleContextAware`). A `registry` ref is then accessible through `muleContext.getRegistry()`. You can then use any suitable lookup method on this instance.

## Classloading in Mule

For details on classloading in Mule, including a diagram of the Mule classloading architecture, see [Classloader Control in Mule](#).

Your Rating:  Results:  0 rates

## Configuring Logging

### Configuring Logging

[ [Troubleshooting Logging](#) ] [ [Controlling Logging from JMX](#) ]

For logging, Mule ESB uses `slf4j`, which is a logging facade that discovers and uses a logging strategy from the classpath, such as `Log4J` or the JDK Logger. By default, Mule includes `Log4J`, which is configured with a file called `log4j.properties`.

The Mule server has a `log4j.properties` in its `conf` directory, which you can customize when running the server in standalone mode. Additionally, all the examples included with Mule have `log4j.properties` files in their `conf` directories.

## Troubleshooting Logging

### I don't see any logging output

A `log4j.properties` file must be at the root of your classpath. If you don't have a `log4j.properties` file, you can get a simple one [here](#). For more information about configuring Log4J, see their [website](#).

### I reconfigured Log4J, but nothing happened

This happens because there is another `log4j.properties` file on your classpath that is getting picked up before your modified one. To find out which configuration file Log4J is using, add the following switch when starting Mule (or container startup script if you are embedding Mule):

```
-M-Dlog4j.debug=true
```

This parameter will write the Log4J startup information, including the location of the configuration file being used, to `stdout`. You must remove that configuration file before your modified configuration will work.

### I don't want to use Log4J

You can remove Log4J by deleting `log4j-xx.jar` from your Mule classpath. You will then need to check that the logging system you are using is supported by [slf4j](#) and put the necessary JAR (unless using JDK Logger) and configuration files on the Mule classpath.

## Controlling Logging from JMX

You can expose a manager's logging configuration over JMX by configuring a Log4J Jmx agent in your Mule configuration file. See [JMX Management](#) for more information.

Your Rating:  Results:  0 rates

## Mule Server Notifications

### Mule Server Notifications

[ [Configuring Notifications](#) ] [ [Firing Custom Notifications](#) ] [ [Notification Interfaces](#) ] [ [Registering Listeners Programmatically](#) ] [ [Notification Payloads](#) ]

Mule ESB provides an internal notification mechanism that you can use to access changes that occur on the Mule Server, such as a service component being added, a Mule Model being initialized, or Mule being started. You can set up your agents or service components to react to these notifications.

### Configuring Notifications

Message notifications provide a snapshot of all information sent into and out of the Mule Server. These notifications are fired whenever a message is received or sent. These additional notifications have some impact on performance, so they are disabled by default. To enable message notifications, you set the type of messages you want to enable using the `<notifications>` element in your Mule configuration file. You also register the notification listeners and associate interfaces with specific notifications.

For example, first you create beans for the notification listeners, specifying the class of the type of notification you want to receive:

```
<spring:bean name="componentNotificationLogger" class="org.myfirm.ComponentMessageNotificationLogger"
/>
<spring:bean name="endpointNotificationLogger"
class="org.myfirm.EndpointMessageNotificationLogger"/>
```

Next, you specify the notifications you want to receive using the `<notification>` element, and then register the listeners using the `<notification-listener>` element:

```

<notifications>
  <notification event="COMPONENT-MESSAGE"/>
  <notification event="ENDPOINT-MESSAGE"/>
  <notification-listener ref="componentNotificationLogger"/>
  <notification-listener ref="endpointNotificationLogger"/>
</notifications>

```

When you specify the COMPONENT-MESSAGE notification, a notification is sent before and after a component is invoked. When you set ENDPOINT-MESSAGE, a notification is sent whenever a message is sent, dispatched, or received on an endpoint. Because the listeners implement the interface for the type of notification they want to receive (for example, the `ComponentMessageNotificationLogger` class would implement `org.mule.api.context.notification.ComponentMessageNotificationListener`), the listeners receive the correct notifications.

For a list of notification types, see [Notifications Configuration Reference](#). For a list of notification listener interfaces, see [Notification Interfaces](#) below.

## Specifying a Different Interface

If you want to change the interface that is associated with a notification, you specify the new interface with the `interface-class` and `interface` attributes:

```

<notifications>
  <notification event="COMPONENT-MESSAGE" interface-class="org.myfirm.MyMessageNotifications"
  interface="myComponentListener"/>

```

## Configuring a Custom Notification

If you create a custom notification, you also specify the `event-class` attribute:

```

<notifications>
  <notification event="CUSTOM-MESSAGE" event-class="org.myfirm.MyMessageNotificationsCustomMessage"
  interface-class="org.myfirm.MyMessageNotifications" interface="myCustomListener"/>
  ...

```

## Disabling Notifications

If you want to block a specific interface from receiving a notification, you specify it with the `<disable-notification>` element. You can specify the notification type (event), event class, interface, and/or interface class to block.

```

<notifications>
  <disable-notification interface="ComponentMessageNotificationListener"/>
  ...

```

## Using Subscriptions

When registering a listener, you can specify that it only receive notifications from a specific component using the `subscription` attribute. For example, to specify that the listener only receive notifications from a service component called "MyService1", you would configure the listener as follows:

```

<notification-listener ref="endpointNotificationLogger" subscription="MyService1"/>

```

You can also register listeners and filter the subscriptions from your Java code:

```
muleContext.registerListener(listener, "MyService1");
```

To register interest in notifications from all service components with "Service" in the name, you would use a wildcard string as follows:

```
muleContext.registerListener(listener, "*Service*");
```

For more information, see [Registering Listeners Programmatically](#) below.

## Firing Custom Notifications

Custom notifications can be fired by objects in Mule to notify custom listeners. For example, a discovery agent might fire a Client Found notification when a client connects.

You fire a custom notification as follows:

```
CustomNotification n = new CustomNotification("Hello");
muleContext.fireNotification(n);
```

Any objects implementing `CustomNotificationListener` will receive this notification. It's a good idea to extend `CustomNotification` and define actions for your custom notification type. For example:

```
DiscoveryNotification n = new DiscoveryNotification(client, DiscoveryNotification.CLIENT_ADDED);
muleContext.fireNotification(n);
```

Note that non-system objects in Mule can only fire custom notifications through the manager. Attempting to fire other notifications such as `ModelNotification` will cause an `UnsupportedOperationException`.

## Notification Interfaces

The following table describes the Mule server notifications and the interfaces in the `org.mule.api.context.notification` class an object can implement to become a listener for that notification. All listeners extend the `ServerNotificationListener` interface.

Notification	Description	Interface
Component Message Notification	A message was processed by a service component. These notifications are very good for tracing, but they are not enabled by default because they have an impact on performance.	ComponentMessageNotificationListener
Connection Notification	A connector connected to its underlying resource or released the connection, or the connection attempt failed.	ConnectionNotificationListener
Custom Notification	Can be fired by objects themselves to custom notification listeners and can be used to customize notifications on agents, service components, connectors, and more.	CustomNotificationListener
Endpoint Message Notification	A message was sent or received from an endpoint. These notifications are very good for tracing, but they are not enabled by default because they have an impact on performance.	EndpointMessageNotificationListener
Exception Notification	An exception was thrown.	ExceptionNotificationListener
Management Notification	The state of the Mule instance or its resources have changed.	ManagementNotificationListener
Model Notification	The state is changing on a model, such as initializing, starting and stopping, or service components within the model are being registered or unregistered.	ModelNotificationListener

Mule Context Notification	An event occurred on the Mule Manager.	MuleContextNotificationListener
Registry Notification	An event occurred on the registry.	RegistryNotificationListener
Routing Notification	A routing event such as an async-reply miss occurred.	RoutingNotificationListener
Security Notification	A request was denied security access.	SecurityNotificationListener
Server Notification	Fired when the server, models, and components stop, start, or initialize.	ServerNotificationListener
Service Notification	An event occurred on a service.	ServiceNotificationListener
Transaction Notification	During transaction life cycle after a transaction has begun, was committed, or was rolled back.	TransactionNotificationListener

The listener interfaces all have a single method:

```
public void onNotification(T notification);
```

where T is a notification class (listener class without the 'Listener' at the end).

Depending on the listener implemented, only certain notifications will be received. For example, if the object implements ManagerNotificationListener, only notifications of type ManagerNotification will be received. Objects can implement more than one listener to receive more types of notifications.

## Registering Listeners Programmatically

You can register listeners on the Mule Context as follows:

```
muleContext.registerListener(listener);
```

## Registering Listeners Dynamically

By default, you cannot register listeners in the Mule context after Mule has started. Therefore, you would register your listeners in your code before starting Mule. For example:

```
MuleContext context = new DefaultMuleContextFactory().createMuleContext
(new SpringXmlConfigurationBuilder("foo-config.xml"));
context.registerListener(listener, "*Service*");
context.start();
```

To change this behavior so that you can add listeners dynamically at run time, you can set the dynamic attribute on the <notifications> element. If you just want to enable dynamic notifications for a specific connector, you can set the dynamicNotification attribute on the connector.



Depending on the nature of your app you may need to call context.unregisterListener() to prevent memory leaks.

## Notification Action Codes

Each notification has an action code that determines the notification type. The action code can be queried to determine its type. For example:

### MyObject.java

```
public class MyObject implements ConnectionNotificationListener<ConnectionNotification>,  
MuleContextAware  
{  
  
    // muleContext injection and field omitted for brevity  
  
    public void onNotification(ConnectionNotification notification)  
    {  
        if (notification.getAction() == ConnectionNotification.CONNECTION_FAILED)  
        {  
            System.out.println("Connection failed");  
        }  
    }  
}
```

For a list of the action codes available with each notification type, see the Javadocs for the [org.mule.context.notification](#) package and click on the class of the notification type you want.

## Notification Payloads

All notifications extend `java.util.EventObject`, and the payload of the object can be accessed using the `getSource()` method. The following table describes the payloads for each type of notification.

Notification	Payload Type	Resource ID	Description
Component Message Notification	Component	Component name	The service component that triggered this notification
Connection Notification	Connectable	<connector-name>.receiver(<endpoint-uri>)	The message receiver or message dispatcher that was connected
Custom Notification	Any object	Any String	The object type is custom to the object firing the notification
Endpoint Message Notification	ImmutableEndpoint	Endpoint URI	The endpoint that triggered this notification
Exception Notification	Throwable	Component Name	The service component that triggered this notification
Management Notification	Object	The object ID	The monitored object that triggered this notification
Model Notification	Model	Model Name	The Model instance on the Mule Context. Equivalent to calling <code>MuleContext.getRegistry().lookupModel()</code>
Mule Context Notification	MuleContext	Mule context ID	The Mule context instance. Equivalent to calling <code>getMuleContext()</code> .
Registry Notification	Registry	Mule registry ID	The Mule registry. Equivalent to calling <code>MuleContext.getRegistry()</code> .
Routing Notification	MuleMessage	Message ID	The message sent or received
Security Notification	SecurityException	The exception message	The security exception that occurred
Service Notification	Service	Service ID	The service that triggered this notification
Transaction Notification	Transaction	Component name	The component that triggered this notification

Your Rating: 

Results:  0 rates

# Profiling Mule

## Profiling Mule

[ [Installing the Profiler Pack](#) ] [ [Enabling the Profiler Agent](#) ] [ [Running the Profiler](#) ] [ [Embedded Mule](#) ]

The Mule Profiler Pack uses YourKit 9.0 to provide CPU and memory profiling, helping you identify memory leaks in your custom Mule ESB extensions. To use a different version of YourKit, instead of using the Profiler Pack see the YourKit documentation for the appropriate version for instructions on how to profile a Java application. When doing this with standalone Mule, note that any JVM flags need to be prefaced with -M so that they affect the Mule process rather than the wrapper process.

### Installing the Profiler Pack

The Profiler Pack is contained in the Mule Enterprise Edition download. If you are installing Mule Enterprise Edition using the graphical installer, simply select the Profiler check box when installing the product.

If you are installing Mule Community Edition, go to the [downloads page](#), and under the latest stable community release, expand the **Downloads** section. You can then click the link to the .zip, .tar, or .gz version of the Profiler pack. After downloading, unpack it on top of the Mule installation.

Set the following environment variable:

```
LD_LIBRARY_PATH=$MULE_HOME/lib/native/profiler
```

### Enabling the Profiler Agent

The Profiler agent exposes the YourKit Profiler to JMX to provide CPU and memory profiling. You configure the Profiler agent with the `<management:yourkit-profiler/>` element. For more information, see [JMX Management](#).

### Running the Profiler

To run the profiler, you run Mule with the **-profile** switch plus any extra YourKit startup options with multiple parameters separated by commas, e.g. **-profile onlylocal,onexit=memory**. This integration pack will automatically take care of configuration differences for Java 1.4.x and 5.x/6.x.

For example:

```
./mule -profile
```

### Embedded Mule

If you are running Mule embedded in a webapp, the Profiler configuration is completely delegated to the owning container. Launch YourKit Profiler, **Tools -> Integrate with J2EE server...** and follow the instructions. Typically, a server's launch script is modified to support profiling, and you then use this modified start script instead of the original.

Your Rating: 

Results:  1 rates

## Hardening your Mule Installation

[ [Hardening your Mule Installation](#) ] [ [Hardening Checklist](#) ] [ [Hardening in Layers](#) ] [ [More Information](#) ]

### Hardening your Mule Installation

As distinct from Security, hardening refers to the steps one must perform in order to bring an application from development into production. These are the hoops one must jump through in order to get something deployed in the IT space after development is completed.

Normally, an IT organization already has control of keeping open ports to a minimum, restricting access to administrative applications, minimizing the number of applications, and other housekeeping. Of course, the files used to configure Mule should be as secure as any configuration files.

### Hardening Checklist

Here is a list of simple steps you can take to begin harden the Mule installation itself:

- Run Mule as a Non-privileged User
- Install Mule as a Service

- Make sure to configure Mule to write logs or temporary files within appropriate locations. Configure logs, passwords, and keystore files
- In some situations, you will need to configure usernames, passwords and keystores on Mule. Usually, these settings are made available externally, so that dev/ops can change these settings.
- Manage certificates in a keystore file
- Use a separate property file to store usernames and passwords and secure it using file system permissions

### **About Running Mule as Non-Privileged User**

On Xnix, you can run mule as any user with the following caveats:

- You will need write perms to logs directory
- Without root you will not be able to use ports below 1024

On Windows, you need to be in admin user's group to run Mule

### **Hardening in Layers**

By its nature, Mule can be situated in a variety of configurations. That said, the suggested approach to hardening involves hardening in layers beginning with the operating system, then working up the stack. The Center for Internet Security (CIS) publishes configuration benchmarks that are widely used in whole or in part as system hardening guides. Mule TCat Server also offers added [security options](#).

Commercial configuration management and integrity management tools can help you automate management to the CIS benchmarks. Also, Mule documentation includes a good deal of information on [configuring security](#). If your application deals in sensitive data, consider using SSL (HTTPS) to protect it.

On the network security side, security experts recommend using a good stateful inspection network firewall with a default-deny rule set and exceptions only for justified business needs. Also, any internet facing server belongs in a DMZ with strong default-deny egress rules on the firewall to prevent data exfiltration. Furthermore, you can use a network IDS/IPS to monitor and prevent known attacks. Putting the database on an internal network - not the DMZ - will also help harden your installation.

Be sure your software developers are familiar with secure web application coding techniques. At the very least, they should be familiar with best practices to avoid common web app pitfalls, such as those listed in [OWASP's top 10](#).

### **More Information**

Keep track of the latest releases of Mule by subscribing to [announce@mule.codehaus.org](mailto:announce@mule.codehaus.org).

Your Rating: 

Results:  0 rates

## **Mule High Availability**

### **Mule High Availability**

[ [Supported Architecture](#) ] [ [Enabling and Configuring Mule High Availability](#) ] [ [Running Mule High Availability](#) ] [ [Example Application](#) ] [ [Limitations to Mule High Availability for Mule ESB Version 3.x](#) ]



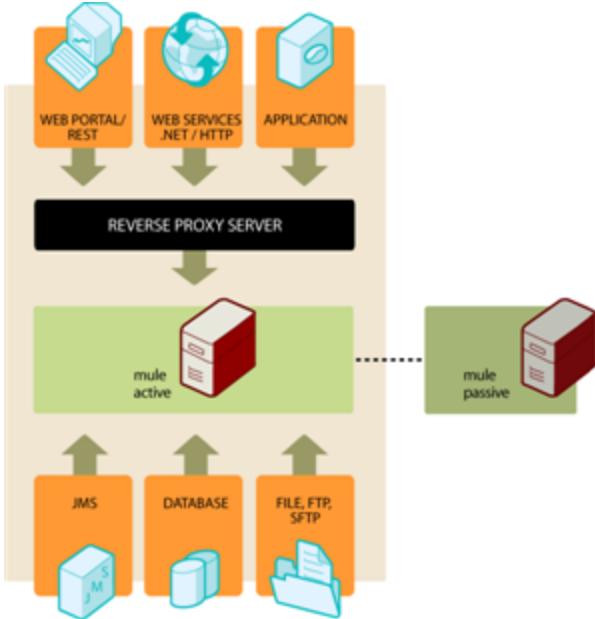
The material on this page applies to Mule 3.1.x. For information related to high availability (as well as high reliability, performance, and scalability) in Mule 3.2.0, see Clusters.

Mule High Availability provides basic failover capability for Mule ESB. When the primary Mule instance becomes unavailable (e.g., because of a fatal JVM or hardware failure or it's taken offline for maintenance), a backup Mule instance immediately becomes the primary node and resumes processing where the failed instance left off. After a system administrator has recovered the failed Mule instance and brought it back online, it automatically becomes the backup node.

Seamless failover is made possible by a distributed memory store that shares all transient state information among clustered Mule instances. This can include information in SEDA service event queues and in-memory message queues.

Mule High Availability is currently available for the following transports:

- HTTP (including CXF Web Services)
- JMS
- WebSphere MQ
- JDBC
- File
- FTP
- VM



## Supported Architecture

Mule High Availability supports the following architecture:

- Two Mule instances running standalone
- Active-Passive topology: one primary node, one backup
- Reverse proxy server (such as [Apache](#)) required for socket-based transports (HTTP, Web services, and TCP) to forward requests to the currently active node
- Multicasting must be enabled on each server where Mule is installed
- High Availability is **not** a replacement for [transactions](#) and does not guarantee reliability. If your message flow is not transactional, you will likely suffer from lost, partial, or duplicated messages in the case a failover should occur.

For details on specific features that are not supported in the current release, see [Limitations](#) below.

## Enabling and Configuring Mule High Availability

You must have two identical Mule instances installed and configured: a primary instance and a backup instance. Although these instances could be on the same machine, they should ideally be separated (different machines in different physical locations) to avoid having a single point of failure.

Be sure to follow the steps in this section for both instances of Mule.

### Configuring the System Properties

High Availability needs a few system properties set in order to start up. By default, these properties are commented out in your `conf/wrapper.conf` file. In order to enable High Availability, you can simply uncomment the properties in this file before starting Mule (it is not necessary to change their values). Note that the last one, `mule.clusterNodeBackupId` needs to be uncommented for all backup instances, but *not* the primary instance.

```
#####
# High Availability settings
#####
wrapper.java.additional.4=-Dmule.clusterId=DEFAULT
wrapper.java.additional.5=-Dmule.clusterNodeId=1
wrapper.java.additional.6=-Dmule.clusterSchema=partitioned-sync2backup
# Uncomment for all but one node in the cluster
# wrapper.java.additional.7=-Dmule.clusterNodeBackupId=1
#####
```



Alternatively, you could set these properties from the command line when you start Mule, or set them in Java code if you start Mule programmatically. For details, see "System Properties" on the [Configuring Properties page](#).

## Modifying Your Configuration Files

To use the Mule High Availability capabilities, you must make some changes to your Mule XML configuration file(s).



Future versions of Mule will make High Availability more transparent, and the following changes to your configuration may no longer be necessary. The Mule Enterprise Migration Tool will help with any migration of configuration when this occurs.

- Import the "cluster" Namespace

```
<mule xmlns="...cut...
       xmlns:cluster="http://www.mulesoft.org/schema/mule/ee/cluster"
       xsi:schemaLocation="
           ...cut...
           http://www.mulesoft.org/schema/mule/ee/cluster
           http://www.mulesoft.org/schema/mule/ee/cluster/3.1/mule-cluster-ee.xsd">
```

- Replace `<service>` elements with `<cluster:service>`

**Before**

```
<model name="ClusterExample">
  <service name="Service1">
    ...cut...
  </service>
  <service name="Service2">
    ...cut...
  </service>
</model>
```

**After**

```
<model name="ClusterExample">
  <cluster:service name="Service1">
    ...cut...
  </cluster:service>
  <cluster:service name="Service2">
    ...cut...
  </cluster:service>
</model>
```



A mixture of `<service>` and `<cluster:service>` elements in a configuration file is not supported. All services must be `<cluster:service>`.

- Replace the VM transport (connector and endpoints) with the Cluster transport

**Before**

```
<vm:connector name="queueConnector" />
...cut...
<inbound>
  <vm:inbound-endpoint name="InQueue" path="queue.in" />
</inbound>
<outbound>
  <pass-through-router>
    <vm:outbound-endpoint name="OutQueue" path="queue.out" />
  </pass-through-router>
</outbound>
```

```

<cluster:connector name="queueConnector" />
...cut...
<inbound>
    <cluster:inbound-endpoint name="InQueue" path="queue.in" />
</inbound>
<outbound>
    <pass-through-router>
        <cluster:outbound-endpoint name="OutQueue" path="queue.out" />
    </pass-through-router>
</outbound>

```

- Replace VM transactions with Cluster transactions

```

<vm:inbound-endpoint queue="InQueue">
    <vm:transaction action="BEGIN_OR_JOIN" />
</vm:inbound-endpoint>

```

```

<cluster:inbound-endpoint queue="InQueue">
    <cluster:transaction action="BEGIN_OR_JOIN" />
</cluster:inbound-endpoint>

```

 Only local transactions are supported with the Cluster transport at this time. XA transactions are not yet supported with the Cluster transport.

## Stateful Components



This feature will be available in a coming release of Mule

If your custom components are stateful (that is, they use variables to store information *between* messages/events) you must ensure that these variables are stored using the provided mechanism for distributed storage. Otherwise, this state information will be lost in the case of failover.

## Running Mule High Availability

After you have enabled and configured High Availability as described above, start your Mule instances. They should now be running in high-availability mode, as you can verify from the splash screen at startup.

```
*****
* Application: default
* OS encoding: MacRoman, Mule encoding: UTF-8
*
* Agents Running:
*   High Availability mode is: PRIMARY
*   JMX Agent
*****
```

The primary instance will be fully started. The backup instance will be running, but its services will be stopped and therefore will not receive messages on any inbound endpoints. If the primary node should become unavailable, the backup node will become the primary node, causing its services to start and begin receiving messages from inbound endpoints. After a systems administrator is able to bring the failed instance back online, it becomes the new backup node (running but with its services stopped).

## Example Application

A simple example application, [Widget Factory](#), is provided with Mule Enterprise to illustrate the use of High Availability. This example is located in the directory `examples/widget` under your Mule home directory. Refer to the `README.txt` file in that directory for information on running the example.

## Limitations to Mule High Availability for Mule ESB Version 3.x

- Mule High Availability is designed to work with two identical Mule ESB instances. These instances must be configured identically.
- Mule High Availability does not currently support Flows or Mule Configuration Patterns. However high availability can be achieved in flows by using transactions and a clustered JMS server. Please contact MuleSoft if you need further assistance.
- Mule HA currently works only with single-application deployments. Note that this application can contain multiple Mule services.
- The only transports currently supported by Mule High Availability are: JMS, JDBC, HTTP, File, and FTP. Asynchronous HTTP, Mail transports, and Streaming have some known compatibility issues and should not be used.
- If a custom component is stateful (i.e., if variables are used to store information *between* messages/events), it will currently lose its state information after failover. Failover of stateful components will be supported in a future release of Mule.
- Stateful routers such as the following may cause lost, partial, or duplicated messages after failover:

XML Element	Class
<code>&lt;correlation-resequencer-router&gt;</code>	<code>org.mule.routing.inbound.CorrelationEventResequencer</code>
<code>&lt;idempotent-receiver-router&gt;</code>	<code>org.mule.routing.inbound.IdempotentReceiver</code>
<code>&lt;idempotent-secure-hash-receiver-router&gt;</code>	<code>org.mule.routing.inbound.IdempotentSecureHashReceiver</code>
<code>&lt;message-chunking-aggregator-router&gt;</code>	<code>org.mule.routing.inbound.MessageChunkingAggregator</code>
<code>&lt;collection-aggregator-router&gt;</code>	<code>org.mule.routing.inbound.SimpleCollectionAggregator</code>
<code>&lt;custom-correlation-aggregator-router&gt;</code>	custom class

Failover of stateful routers will be supported in a future release of Mule.

- Unexpected behavior may occur with inbound JMS topics. JMS queues as well as outbound topics should work as normal.
- Session-based information such as Security Context or Servlet Context and possibly HTTPS certificates will be lost after failover.
- XA transactions are not yet supported with the Cluster transport. (Local transactions are fully supported.)
- Lifecycle changes such as pausing or stopping services, connectors, or agents via Mule MMC will be lost after failover, and all entities will return to their default state, usually "started".
- Schedules based on Quartz, such as receiver polling intervals and the Quartz transport, will restart after failover. Therefore, a single interval at less than the specified time period may occur.
- Statistics collection such as message throughput is per Mule instance. Aggregate statistics will not be available after failover.



### Transactions

Mule High Availability is **not** a replacement for transactions. If your message flow is not transactional, you will likely suffer from lost, partial, or duplicated messages in the case a failover should occur. Transports that are not transactional for a single Mule instance (File system, FTP, HTTP) are still not transactional for Mule High Availability.

Your Rating:

Results: 1 rates

## Configuring Mule for Different Deployment Scenarios

### Configuring Mule for Different Deployment Scenarios

This section describes configuring Mule to run as a Linux or Unix daemon, a Windows service, or from a script.

- Configuring Mule as a Linux or Unix Daemon
- Configuring Mule as a Windows Service
- Configuring Mule to Run From a Script
- Configuring Mule to Run From Maven

## Configuring Mule as a Linux or Unix Daemon

### ***Running Mule as a Daemon***

By default, the `mule` command runs Mule in the foreground. To run Mule in the background as a daemon, enter the following command instead, using `start`, `stop`, or `restart` as the first parameter as needed:

```
mule [start|stop|restart]
```

Your Rating:  Results:  1 rates

## Configuring Mule as a Windows Service

### ***Configuring Mule as a Windows NT Service***

If you want to install Mule as a Windows NT Service, type

```
mule install
```

Likewise, to remove Mule from your services, type:

```
mule remove
```

Once you have installed Mule as a service, you can invoke the service exactly as you did before, but with an additional parameter:

```
mule [start|stop|restart]
```

You can also use the Windows `net` utility:

```
net [start|stop] mule
```

Your Rating:  Results:  0 rates

## Configuring Mule to Run From a Script

### ***Configuring Mule to Run From a Script***

To start Mule from a script or from your IDE without using the Java Service Wrapper, you can use the `org.mule.MuleServer` class. This class accepts a couple of parameters.

```
org.mule.MuleServer -config mule-config.xml
```

or

```
org.mule.MuleServer -builder <fully qualified classname> -config appContext.xml
```

- **-config** specifies one or more configuration files to use. If this argument is omitted, it will look for and use `mule-config.xml` if it exists.
- **-builder** is a fully qualified classname of the configuration builder to use. If this is not set, the default `org.mule.config.builders.AutoConfigurationBuilder` is used, which will try to auto-detect configuration files based on available builders. In the most common scenario, this will resolve to `org.mule.config.spring.SpringXmlConfigurationBuilder`.

The easiest way to set the classpath is to include all JARs in the `./lib/mule` and `./lib/opt` directories of the distribution. You can look at the dependency report for the server and each of the modules to see exactly which JARs are required for a particular module.

Your Rating: 

Results:  0 rates

## Configuring Mule to Run From Maven

### Configuring Mule to Run From Maven

Let's say you are building and installing a Java web service with Maven, and you want to run SoapUI tests for it. You could use an automated test framework such as Hudson to run the SoapUI tests after it building the Java WS.

In this case, you may want a Mule instance running for the SoapUI tests to run. You can add a task in Maven so that it starts up Mule (possibly passing the correct config file) and then runs the SoapUI tests.

Use the `exec-maven-plugin`'s `java` goal to run the class `org.mule.MuleServer` and pass it a '`-config configname.xml`' parameter. Maven will work out the classpath for you.

Your Rating: 

Results:  0 rates

## Testing With Mule ESB 3

### Testing With Mule ESB 3

This section describes how to test your Mule application.

- [Introduction to Testing Mule](#)
- [Logger Element for Flows](#)
- [Unit Testing](#)
- [Functional Testing](#)
- [Running Benchmark Tests](#)
- [Using Dynamic Ports in Mule Test Cases](#)
- [Profiling Mule](#)

Your Rating: 

Results:  0 rates

## Introduction to Testing Mule

### Introduction to Testing Mule

This page describes the types of testing you can perform on Mule ESB. In general, you can always look at examples of existing tests for guidance. Every Mule ESB maven project should have a `src/test/` directory which contain some unit/functional tests. There is also a maven 'tests' project which contains some useful functional and integration tests.

### Types of Testing

When you configure and customize Mule, you perform the following types of tests:

- Unit testing of your simple extensions and customizations

- Functional testing of your Mule configuration and setup
- Functional and unit testing of your custom modules and transports
- Integration testing of multiple modules/transports/transformers/etc
- System testing of your transports which require connections to embedded/external services
- Stress/Performance testing (see below)

Mule provides functional test classes in the `org.mule.tck` and `org.mule.tck.functional` packages that allow you to test your configuration as well as your custom modules and transports. For more information, see [Functional Testing](#). Additionally, the Mule test JAR file contains abstract test cases that you can use for unit testing your simple extensions (e.g., `AbstractTransformerTestCase` and `AbstractOutboundRouterTestCase`) as well as your custom modules and transports (e.g., `AbstractConnectorTestCase`, `AbstractReceiverTestCase`, `AbstractDispatcherTestCase`, and `AbstractEndpointTestCase`). For more information, see [Unit Testing](#).

## Performance Tests

After you have ensured that your setup and configuration are correct and that your customizations are working, you should ensure that your system is performing correctly. You can run [Japex benchmark tests](#) to test individual packages. Additionally, the [Mule Profiler Pack](#) helps you identify memory leaks in your customizations.

## Using MuleForge for Continuous Integration Testing

If you host your Mule project on MuleForge, you can take advantage of continuous integration testing. MuleForge projects are configured to be automatically built using Bamboo, a Continuous Integration Build Server from Atlassian. The source code build frequency is set to 30 minutes, while the snapshot build frequency is set to 1 day. You can request that these frequencies be changed for your project.

For more information on hosting your project on MuleForge, see the [Despot's Guide](#).

Your Rating:  Results:  1 rates

## Using IDEs

### Using IDEs

You can use an integrated development environment (IDE) such as Eclipse, IntelliJ, and Mule IDE to rapidly develop Mule ESB applications. For more information on the Mule IDE, see the [Mule IDE User Guide](#).

Usually, you simply attach the `src.zip` file that comes with the Mule distribution to the Mule JARs in your project so you can browse the source code while developing your classes. If you want to build Mule from source, see the following topics in the Mule Developer's Guide:

- [Setting Up the Development Environment](#)
- [Working with an IDE](#)
- [Building from Source](#)

Your Rating:  Results:  0 rates

## Unit Testing

### Unit Testing

Mule ESB provides a Test Compatibility Kit (TCK) of unit tests that you can use to test your simple extensions as well as your custom modules and transports. The unit tests are located in the `-tests.jar` file, such as `mule-core-3.0.0-tests.jar` for Mule version 3.0.0. All unit tests inherit from `org.mule.tck.AbstractMuleTestCase`.

These unit tests are beneficial for the following reasons:

- Components tested with a TCK test case ensure that the common behavior of the component is compatible with the Mule framework.
- Using a TCK test case allows the developer to concentrate on writing tests for specific behavior of their component.
- Where testing of a method in the Service Component API cannot be tested by the TCK test case, the test cases provides an abstract method for the test, ensuring the developer tests all areas of the component.
- The TCK provides a default test model that is a simple set of test classes. The developer doesn't need to worry about writing new test classes for their test cases each time.
- The abstract test cases in the TCK use JUnit's `TestCase`, so they are compatible with other test cases.

Following is a description of some of the unit tests in the Mule TCK:

Testing Component	Description
AbstractMuleTestCase	A helper test case providing methods for creating test and mock object types. This is the base class for all other abstract TCK classes.
AbstractConnectorTestCase	Used to test the common behavior of a connector. This tests dispatching and sending events using mock objects.
AbstractMuleMessageFactoryTestCase	Provides tests for all the standard methods defined in the MuleMessageFactory interface. Add specific tests for converting your transport message to a MuleMessage in your subclass.
AbstractMessageReceiverTestCase	Used to test the common behavior of a MessageReceiver. This tests receiving messages using mock objects.
AbstractComponentTestCase	This is the base class for unit tests that test custom component implementations. Concrete subclasses of this base class include DefaultJavaComponentTestCase, PooledJavaComponentTestCase, and SimpleCallableJavaComponentTestCase, each of which contains methods for testing that component type. For example, the DefaultJavaComponentTestCase includes methods for testing the creation, lifecycle, and disposal of a basic Java component.
AbstractTransformerTestCase	Used to test transformers. This class defines a number of tests that ensures that the transformer works in single scenarios as well as in round trip scenarios. There are many concrete sub-classes of this abstract class that test specific types of transformers, such as StringByteArrayTransformersTestCase.
DefaultMuleContextTestCase	Tests the creation and disposal of the Mule context.
AbstractServiceTestCase	An abstract test case that provides methods for testing the starting, stopping, pausing, resuming, and disposing of services.

Your Rating: 

Results:  1 rates

## Functional Testing

### Functional Testing

[ FunctionalTestCase ] [ FunctionalTestComponent ] [ Additional Features ] [ Additional Example: Event Callback With a Spring Component ] [ Test Component Configuration Reference ] [ Component ] [ Web Service Component ]

Because Mule ESB is light-weight and embeddable, it is easy to run a Mule Server inside a test case. Mule provides an abstract JUnit test case called `org.mule.tck.FunctionalTestCase` that runs Mule inside a test case and manages the lifecycle of the server. The `org.mule.tck.functional` package contains a number of supporting classes for functionally testing Mule code, including `FunctionalTestComponent`. These classes are described in more detail in the following sections.

#### FunctionalTestCase

`FunctionalTestCase` is a base test case for Mule functional tests. Your test cases can extend `FunctionalTestCase` to use its functionality.

`FunctionalTestCase` fires up a Mule server using a configuration you specify by overriding the `getConfigResources()`:

```
protected String getConfigResources()
{
    return "mule-conf.xml";
}
```



You can use the method `getConfigResources` to specify a configuration file or comma-separated list of configuration files to use. All configuration files must exist in your classpath.

You then create tests that interact with the Mule server. `FunctionalTestCase` extends `junit.framework.TestCase`, so JUnit is the framework for creating and running your test cases. For example, this simple test would send a message to a vm endpoint.

```

public void testSend() throws Exception
{
    MuleClient client = new MuleClient(muleContext);
    String payload = "foo";
    Map<String, Object> properties = null;
    MuleMessage result = client.send("vm://test", payload, properties);
    assertEquals("foo Received", result.getPayloadAsString());
}

```

Notice the use of `MuleClient` to interact with the running Mule server. `MuleClient` is used to send messages to and receive messages from endpoints you specify in your Mule configuration file (`mule-conf.xml` in this case). The example `mule-conf.xml` file used in this example is shown below:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd
          http://www.mulesoft.org/schema/mule/test http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd">

    <model name="TestComponentModel">
        <service name="TestComponentService">
            <inbound>
                <inbound-endpoint address="vm://test"/>
            </inbound>
            <test:component appendString=" Received"/>
        </service>
    </model>
</mule>

```

## Watchdog Timeout

The base test case class includes a watchdog timeout feature that times out your functional test after 60 seconds. To change this setting, add `-Dmule.test.timeoutSecs=XX` either to the `mvn` command you use to run Mule or to the JUnit test runner in your IDE. As of Mule 3.0-M2 you can also set the environment variable `MULE_TEST_TIMEOUTSECS`. If both the system property and the environment variable are set, the system property takes precedence.

## FunctionalTestComponent

The previous example of `FunctionalTestCase` covers many common (synchronous) test scenarios, where the service responds directly to the caller. `FunctionalTestComponent` can help support richer tests, such as:

1. Simulating asynchronous communication
2. Returning mock data to the caller
3. Common scenarios such as forced exceptions, storing message history, appending text to responses, and delayed responses.

The component includes two methods: the `onCall` method and the `onReceive` method that basically do the same thing.

- **onCall:** receives a `MuleEventContext` as input and returns an `Object`.
- **onReceive:** receives an `Object` as input and returns an `Object`.

In both methods, `FunctionalTestComponent` takes the message that is passed to it (either from the `MuleEventContext` or from the `Object`) and transform it into a `String`. It then creates a message and sends it back to the caller. It also checks whether any of its properties are set and acts accordingly.

## Asynchronous Tests with FunctionalTestComponent

The `FunctionalTestComponent` supports two event mechanisms for responding to a caller asynchronously: event callbacks and notifications.

Both event callbacks and notifications fire events that get handled by registered listeners. During functional testing, the listener will typically be a class accessible in the `FunctionalTestCase`.

## Event Callbacks

User-defined event callbacks get called when the test component is invoked. Following is an example of a test case and Mule configuration that uses callbacks:

```
public void testEventCallback() throws Exception
{
    EventCallback callback = new EventCallback()
    {
        public void eventReceived(MuleEventContext context, Object component)
            throws Exception
        {
            System.out.println("Thanks for calling me back");
        }
    };

    getFunctionalTestComponent("TestComponentService").setEventCallback(callback);

    MuleClient client = new MuleClient();

    client.send("vm://test", new DefaultMuleMessage("foo"));
}
```

In this example, the `eventReceived` callback method is invoked as soon as the `FunctionalTestComponent` receives the message, and a message is printed to the console. Test assertions could be made in this method.

The corresponding Mule configuration used in this example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:test="http://www.mulesoft.org/schema/mule/test"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/test http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd">

    <model name="TestComponentModel">
        <service name="TestComponentService">
            <inbound>
                <inbound-endpoint address="vm://test"/>
            </inbound>
            <component>
                <singleton-object class="org.mule.tck.functional.FunctionalTestComponent"/>
            </component>
        </service>
    </model>
</mule>
```

Notice that in this configuration, we did not use the "`<test:component>`" element, since we need `FunctionalTestComponent` to be singleton for the callback to work properly.

For an example of an event callback on a Spring component, see the [additional example](#) below.

## Notifications

Notifications are an alternative to event callbacks. When an event is received, the `FunctionalTestComponent` fires a notification informing us that the event has been received. It is up to us to set up a listener (the `FunctionalTestNotificationListener`) on our test to capture this

notification.

To do this, we must first make our test case implement the `FunctionalTestNotificationListener` interface. Then, we must implement the method exposed by this listener, which is `onNotification`. In the example below, we check `notification.getAction` to see whether it is the `FunctionalTestNotification` fired by the `FunctionalTestComponent`. If it is, we print it out to the console.

```
public void onNotification(ServerNotification notification)
{
    if (notification.getAction() == FunctionalTestNotification.EVENT_RECEIVED)
    {
        System.out.println("Event Received");
    }
}
```

Now, in order for our listener to start listening for notifications, we must register it:

```
muleContext.registerListener(this, "myComponent");
```

### Returning Mock Data from `FunctionalTestComponent`

`FunctionalTestComponent` can return mock data specified either in a file or embedded in the Mule configuration. For example, to have the `FunctionalTestComponent` return the message "donkey", you would configure the component as follows:

```
<test:component>
    <test:return-data>donkey</test:return-data>
</test:component>
```

To return contents from a file, you could use:

```
<test:component>
    <test:return-data file="abc.txt" />
</test:component>
```

The file referenced should exist on the Mule classpath.

### Other Useful Features of `FunctionalTestComponent`

#### *Forcing Exceptions*

You can use `throwException` to always return the exception specified by `exceptionToThrow`, as follows:

```
<test:component throwException="true" exceptionToThrow="your.service.exception"/>
```

#### *Storing Message History*

By default, every message that is received by the `FunctionalTestComponent` is stored and can be retrieved. If you do not want this information stored, you can set `enableMessageHistory` to false. For example, if you are running millions of messages through the component, an out-of-memory error would probably occur eventually if this feature were enabled.

To enable:

```
<test:component enableMessageHistory="true" />
```

Messages are stored in an `ArrayList`. To retrieve a stored message, you use the `getReceivedMessage` method to retrieve it by number (e.g.,

`getReceivedMessage(1)` to retrieve the first message stored), or use `getLastReceivedMessage` to retrieve the last message that was received. You can use `getReceivedMessages` to return the total number of messages stored.

## Appending Text to Responses

You can use `appendString` to append text to the response message, as follows:

```
<test:component appendString="Received" />
```

## Delayed Responses

You can set `waitTime` to delay responses from this `FunctionalTestComponent`. In this example, responses are delayed five seconds:

```
<test:component waitTime="5000" />
```

## Disable Inbound Transformer

You can set `doInboundTransform` to `false` to disable the inbound transformer. For example:

```
<test:component doInboundTransform="false" />
```

## Additional Features

The functional package includes several additional classes, such as `CounterCallback`, a test callback that counts the number of messages received. For complete information, see the [org.mule.tck.functional](#) Javadoc.

## Additional Example: Event Callback With a Spring Component

This example is similar to the "Event Callbacks" example above, except the component used here is a Spring component. In this case, we can look up the component using the Spring registry.

```
public void testEventCallback() throws Exception
{
    EventCallback callback = new EventCallback()
    {
        public void eventReceived(MuleEventContext context, Object component)
            throws Exception
        {
            System.out.println("Thanks for calling me back");
        }
    };

    ApplicationContext ac =
(ApplicationContext)muleContext.getRegistry().lookupObject(SpringRegistry.SPRING_APPLICATION_CONTEXT);
    FunctionalTestComponent testComponent = (FunctionalTestComponent) ac.getBean("FTC");
    testComponent.setEventCallback(callback);

    MuleClient client = new MuleClient();

    client.send("vm://test", new DefaultMuleMessage("foo"));
}
```

The corresponding Mule configuration would be as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:test="http://www.mulesoft.org/schema/mule/test"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd
        http://www.mulesoft.org/schema/mule/test http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd">

    <spring:bean id="FTC" class="org.mule.tck.functional.FunctionalTestComponent" />

    <model name="TestComponentModel">
        <service name="TestComponentService">
            <inbound>
                <inbound-endpoint address="vm://test" />
            </inbound>
            <component>
                <spring-object bean="FTC" />
            </component>
        </service>
    </model>
</mule>

```

## Test Component Configuration Reference

Following is detailed information about the test components provided in the test framework (mule-test.xsd).

### Component

A component that can be used for testing message flows. It is a configurable component. The return data for the component can be set so that users can simulate a call to a real service. This component can also track invocation history and fire notifications when messages are received.

#### Attributes of <component...>

Name	Type	Required	Default	Description
throwException	boolean	no		Whether the component should throw an exception before any processing takes place.
logMessageDetails	boolean	no		Whether to output all message details to the log. This includes all headers and the full payload. The information will be logged at INFO level.
dolnboundTransform	boolean	no		Whether the message will be transformed using the transformer(s) set on the inbound endpoint before it gets processed. The default is true.
exceptionToThrow	name (no spaces)	no		A fully qualified classname of the exception object to throw. Used in conjunction with throwException. If this is not specified, a FunctionalTestException will be thrown by default.
enableMessageHistory	boolean	no		Every message that is received by the test component is stored and can be retrieved. If you do not want this information stored, such as if you are running millions of messages through the component, you can disable this feature to avoid a potential out of memory error.
enableNotifications	boolean	no		Whether to fire a FunctionalTestNotification when a message is received by the component. Test cases can register to receive these notifications and make assertions on the current message.

appendString	string	no		A string value that will be appended to every message payload that passes through the component. Note that by setting this property you implicitly select that the message payload will be converted to a string and that a string payload will be returned. The inbound transformer (if any) will get applied first, but if that does not return a string, <code>MuleEventContext.getMessageAsString()</code> will be called directly after.
waitTime	long	no		The time in milliseconds to wait before returning a result. All processing happens in the component before the wait begins.

#### Child Elements of <component...>

Name	Cardinality	Description
return-data	0..1	Defines the data to return from the service once it has been invoked. The return data can be located in a file, which you specify using the <code>file</code> attribute (specify a resource on the classpath or on disk), or the return data can be embedded directly in the XML.
callback	0..1	A user-defined callback that is invoked when the test component is invoked. This can be useful for capturing information such as message counts. Use the <code>class</code> attribute to specify the callback class name, which must be an object that implements <code>org.mule.tck.functional.EventCallback</code> .

## Web Service Component

A component that can be used for testing web services. This component has the same properties as `component` element, but in addition to implementing `org.mule.api.lifecycle.Callable`, it also implements `org.mule.api.component.simple.EchoService`, `org.mule.tck.testmodels.services.DateService`, and `org.mule.tck.testmodels.services.PeopleService`. When using this with WS endpoints such as CXF, be sure to set the `serviceClass` property of the endpoint to the type of service you are using.

#### Attributes of <web-service-component...>

Name	Type	Required	Default	Description
throwException	boolean	no		Whether the component should throw an exception before any processing takes place.
logMessageDetails	boolean	no		Whether to output all message details to the log. This includes all headers and the full payload. The information will be logged at INFO level.
doInboundTransform	boolean	no		Whether the message will be transformed using the transformer(s) set on the inbound endpoint before it gets processed. The default is true.
exceptionToThrow	name (no spaces)	no		A fully qualified classname of the exception object to throw. Used in conjunction with <code>throwException</code> . If this is not specified, a <code>FunctionalTestException</code> will be thrown by default.
enableMessageHistory	boolean	no		Every message that is received by the test component is stored and can be retrieved. If you do not want this information stored, such as if you are running millions of messages through the component, you can disable this feature to avoid a potential out of memory error.
enableNotifications	boolean	no		Whether to fire a <code>FunctionalTestNotification</code> when a message is received by the component. Test cases can register to receive these notifications and make assertions on the current message.
appendString	string	no		A string value that will be appended to every message payload that passes through the component. Note that by setting this property you implicitly select that the message payload will be converted to a string and that a string payload will be returned. The inbound transformer (if any) will get applied first, but if that does not return a string, <code>MuleEventContext.getMessageAsString()</code> will be called directly after.
waitTime	long	no		The time in milliseconds to wait before returning a result. All processing happens in the component before the wait begins.

#### Child Elements of <web-service-component...>

Name	Cardinality	Description

return-data	0..1	Defines the data to return from the service once it has been invoked. The return data can be located in a file, which you specify using the <code>file</code> attribute (specify a resource on the classpath or on disk), or the return data can be embedded directly in the XML.
callback	0..1	A user-defined callback that is invoked when the test component is invoked. This can be useful for capturing information such as message counts. Use the <code>class</code> attribute to specify the callback class name, which must be an object that implements <code>org.mule.tck.functional.EventCallback</code> .

Your Rating: 

Results:  0 rates

## Using Dynamic Ports in Mule Test Cases

### Using Dynamic Ports in Mule Test Cases

If you find yourself hitting test failures due to ports not being available during testing, using the dynamic ports testing framework should fix your issue.

#### Pre-requisites:

1. You need to know how many ports you test will use. Count the number of unique ports in your mule-config file(s) for a given test class.
2. Your test class extends `FunctionalTestCase`.
3. You are ok with the ports being between 5000-6000

#### Enabling dynamic ports

Adding dynamic ports to your tests is pretty simple:

1. Wherever your test extends `FunctionalTestCase`, extend `DynamicPortTestCase` instead
2. Define a method `getNumPortsToFind` and return an int with the number of ports you need for testing:

```
protected int getNumPortsToFind()
{
    return 1;
}
```

3. Replace the hard-coded ports in your mule config files with incremental parameters, starting with  `${port1}`, i.e.

```
<http:endpoint name="clientEndpoint1" host="localhost" port="${port1}"
    path="test1/?foo=boo&far=bar" exchange-pattern="request-response" />
```

4. If your test class uses hardcoded port numbers, you have a couple of ways of updating it:

- a. If you are using a Mule api call like `client.send`, add the name parameter to your endpoint and use that to refer to the endpoint, i.e.

```
MuleMessage result = client.send(((InboundEndpoint) client.getMuleContext().getRegistry()
    .lookupObject("inMain")).getAddress(),
    msg, null);
```

- b. If you need access to the dynamic port number, use the `AbstractMuleTestCase.getPorts()` method, where index 0 maps to  `${port1}`, index 1 maps to  `${port2}`, etc:

```
getPorts().get(1)
```

Your Rating: 

Results:  1 rates

## Testing Strategies

### Testing Strategies

Building a comprehensive suite of automated tests for your Mule project is the primary factor that will ensure its longevity: you'll gain the security of a safety net catching any regression or incompatible change in your applications before they even leave your workstation.

We'll look at testing under three different aspects:

\*Unit testing: these tests are designed to be fast, with a very narrow system under test. Mule is typically not run for unit tests.

\*Functional testing: these tests usually involve running Mule, though with a limited configuration, and should run fast enough to be executed on each build.

\*Integration testing: these tests exercise a full Mule application with settings that are as close to production as possible. They are usually slower to run and not part of the regular build.

In practice, unit and functional testing are often merged and executed together.

## Unit Testing

In a Mule application, unit testing is limited to the code that can be realistically exercised without the need to run it inside Mule itself. As a rule of thumb, code that is Mule aware (for example, code that relies on the registry), will better be exercised with a functional test

With this in mind, the following are good candidates for unit testing:

\*Custom transformers

\*Custom components

\*Custom expression evaluators

\*All the Spring beans that your Mule application will use. Typically, these beans come as part of a dependency JAR and are tested while being built, alleviating the need for re-retesting them in your Mule application project

Mule provides abstract parent classes to help with unit testing. Turn to the following URL for more information about them:

<http://www.mulesoft.org/documentation/display/MULE3USER/Unit+Testing>

## Functional Testing

Functional tests are those that most extensively exercise your application configuration. In these tests, you'll have the freedom and tools for simulating happy and unhappy paths.

The "paths" that you will be interested to cover include:

\*Message flows

\*Rule-based routing, including validation handling within these flows

\*Error handling

If you've modularized your configuration files as explained in section 2, you've put yourself in a great position for starting functional testing.

Let's see why:

\*Imported configurations can be tested in isolation. This means that you will be able to create one functional test suite for each of the different imported configuration. This reduces the size of the system under test, making it easier to write tests for each of the different cases that need to be covered

\*Side-by-side transport configuration allows transport switching and failure injection. This means you'll not need to use real transports (say HTTP, JMS or JDBC) when running your functional but will be able to run everything through VM in-memory queues. You will also have the possibility to create stubs for target services and make them fail to easily simulate unhappy paths

Real transports or not? That is the question you're maybe asking and it is a valid one, as many in-memory alternatives exist for the different infrastructures your Mule application will connect to (for example: ActiveMQ for JMS, HSQLDB for JDBC). The real question is: what are you really testing? Is it relevant for your functional tests to exercise the actual transports, knowing that they're already tested by MuleSoft and that the integration tests will take care of exercising them.

Unable to render embedded object: File (functional\_tests\_sut.png) not found.

Mule provides a lot of supporting features for implementing functional tests. Let's look into an example and discover them as we go. The following diagram illustrates the flow we will be testing:

This flow accepts incoming messages over HTTP, validates them and dispatches them to JMS if they are acceptable. For the actual implementation, we will be using the Validator configuration pattern and check that the incoming message payload is XML. Keep in mind that the same testing principles and tools apply if you're testing a flow (or even a service).

## Testing with side-by-side configurations

Let's look at the configuration files for this application. First, we have the configuration file that contains the Validator:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule-xml="http://www.mulesoft.org/schema/mule/xml"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/xml
          http://www.mulesoft.org/schema/mule/xml/3.1/mule-xml.xsd">

    <validator name="WorkAcceptor"
               inboundEndpoint-ref="NewWorkEndpoint"
               ackExpression="#{string:OK:#[message:id]}"
               nackExpression="#{string:NOT_XML}"
               outboundEndpoint-ref="AcceptedWorkEndpoint">
        <mule-xml:is-xml-filter/>
    </validator>
</mule>

```

Note how the inbound and outbound endpoints are actually references to global ones. These global endpoints are configured in a separate configuration file designed to be loaded side-by-side with the above one. Here is its content, without the JMS connector configuration omitted for brevity:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd">

    <http:endpoint name="NewWorkEndpoint"
                   host="${web.host}"
                   port="8080"
                   path="api/work">
        <object-to-string-transformer/>
    </http:endpoint>

    <!-- JMS connector configuration omitted -->

    <jms:endpoint name="AcceptedWorkEndpoint"
                  queue="work"
                  connector-ref="WorkQueueJmsConnector" />
</mule>

```

Note how this configuration provides the actual configuration of the global endpoints used by the other configuration. In order to functional test this, we will have to create an alternative configuration that provides global endpoints with the same name but use the VM transport. Here it is:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/vm"
      http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <vm:endpoint name="NewWorkEndpoint"
                  path="work.new"
                  exchange-pattern="request-response" />

    <vm:endpoint name="AcceptedWorkEndpoint"
                  path="work.ok"
                  exchange-pattern="one-way" />
</mule>

```

Now let's write two tests: one for each possible path (message is XML or not). We will subclass Mule's `FunctionalTestCase`, an abstract class designed to be the parent of all your functional tests!

The `FunctionalTestCase` class is a descendant of JUnit's `TestCase` class.

Here is the test class, without the Java import declarations:

```

public class WorkManagerFunctionalTestCase extends FunctionalTestCase
{
    @Override
    protected String getConfigResources()
    {
        return "mule-workmanager-config.xml,mule-test-transports-config.xml";
    }

    public void testValidJob() throws Exception
    {
        MuleClient client = new MuleClient(muleContext);
        MuleMessage result = client.send("vm://work.new", "<valid_xml />", null);
        assertTrue(result.getPayloadAsString().startsWith("OK:"));

        MuleMessage dispatched = client.request("vm://work.ok", 5000L);
        assertEquals("<valid_xml />", dispatched.getPayloadAsString());
    }

    public void testInvalidJob() throws Exception
    {
        MuleClient client = new MuleClient(muleContext);
        MuleMessage result = client.send("vm://work.new", "not_xml", null);
        assertTrue(result.getPayloadAsString().startsWith("NOT_XML"));

        MuleMessage dispatched = client.request("vm://work.ok", 5000L);
        assertNull(dispatched);
    }
}

```

Notice in `testValidJob()` how we ensure we receive the expected synchronous response to our valid call (starting with "OK:") but also how we check that the message has been correctly dispatched to the expected destination by requesting it from the target VM queue. Conversely in `testInvalidJob()` we verify that nothing has been sent to the valid work endpoint.

As standard JUnit tests, you can now run these tests either from Eclipse or the command line with Maven.

Using a VM queue to accumulate messages and subsequently requesting them (as we did with `vm://work.ok`) can only work with the one-way exchange pattern. Using a request-response pattern would make Mule look for a consumer of the VM queue, as a synchronous response is expected. So what do we do when we have to test request-response endpoints? We use the Functional Test Component!

Stubbing out with the Functional Test Component

The Functional Test Component (FTC) is a programmable stub that can be used to consume messages from endpoints, accumulate these messages, respond to them and even throw exceptions. Let's revisit our example and see how the FTC can help us, as our requirements are

changing.

We have decided to use a Validator's feature that wasn't used previously, which ensures that the message has been successfully dispatched to the accepted job endpoint and otherwise returns a failure message to the caller. Here is it's new configuration:

```
<validator name="WorkAcceptor"
    inboundEndpoint-ref="NewWorkEndpoint"
    ackExpression="#[string:OK:#[message:id]]"
    nackExpression="#[string:NOT_XML]"
    errorExpression="#[string:SERVER_ERROR]"
    outboundEndpoint-ref="AcceptedWorkEndpoint">
    <mule-xml:is-xml-filter/>
</validator>
```

The only difference is that an error expression has been added. This addition yields the following changes:

\*The Validator will now behave fully synchronously, preventing us from using an outbound VM queue as an accumulator of dispatched messages: we will have to use the FTC to play the role of accumulator,

\*A new path will have to be tested as we will want to check the behavior of the system when dispatching fails. We will also use the FTC here, configuring it to throw an exception upon message consumption.

Let's see how introducing the FTC has changed our test transports configuration:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
xmlns:test="http://www.mulesoft.org/schema/mule/test"
xsi:schemaLocation="
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd
    http://www.mulesoft.org/schema/mule/test http://www.mulesoft.org/schema/mule/test/3.1/mule-test.xsd">

    <vm:endpoint name="NewWorkEndpoint"
        path="work.new"
        exchange-pattern="request-response" />

    <vm:endpoint name="AcceptedWorkEndpoint"
        path="work.ok"
        exchange-pattern="request-response" />

    <simple-service name="WorkQueueProcessorStub"
        endpoint-ref="AcceptedWorkEndpoint">
        <test:component />
    </simple-service>
</mule>
```

As you can see, the FTC manifests itself as a `<test:component />` element. We used the convenience of the Simple Service pattern to make it consume the messages sent to the AcceptedWorkEndpoint.

The FTC supports plenty of configuration options. Read more about it there:  
<http://www.mulesoft.org/documentation/display/MULE3USER/Functional+Testing>

Now that we have this in place, let's see first how we can test the new failure path. Here is the source code of the new test method added to our previously existing functional test case:

```

public void testDispatchError() throws Exception
{
    FunctionalTestComponent ftc =
        getFunctionalTestComponent("WorkQueueProcessorStub");
    ftc.setThrowException(true);

    MuleClient client = new MuleClient(muleContext);
    MuleMessage result = client.send("vm://work.new", "<valid_xml />", null);
    assertTrue(result.getPayloadAsString().startsWith("SERVER_ERROR"));
}

```

Note how we get hold of the particular FTC we're interested in: we use `getFunctionalTestComponent`, a protected method provided by the parent class, to locate the component that sits at the heart of our Simple Service (located by its name).

Once we have gained a reference to the FTC, we configure it for this particular test so it will throw an exception anytime it is called. With this in place, our test works: the exception that is raised makes the Validator use our provided error expression to build its response message.

Now lets look at how we've refactored the existing test methods to use the FTC:

```

public void testValidJob() throws Exception
{
    MuleClient client = new MuleClient(muleContext);
    MuleMessage result = client.send("vm://work.new", "<valid_xml />", null);
    assertTrue(result.getPayloadAsString().startsWith("OK"));

    FunctionalTestComponent ftc =
        getFunctionalTestComponent("WorkQueueProcessorStub");
    assertEquals("<valid_xml />", ftc.getLastReceivedMessage());
}

public void testInvalidJob() throws Exception
{
    FunctionalTestComponent ftc =
        getFunctionalTestComponent("WorkQueueProcessorStub");
    ftc.setThrowException(true);

    MuleClient client = new MuleClient(muleContext);
    MuleMessage result = client.send("vm://work.new", "not_xml", null);
    assertTrue(result.getPayloadAsString().startsWith("NOT_XML"));
}

```

In `testValidJob()`, the main difference is that we now query the FTC for the dispatched message instead of requesting it from the outbound VM queue.

In `testInvalidJob()`, the main difference is that we configured the FTC to fail if a message gets dispatched despite the fact it is invalid. This approach actually leads to a better performance of the test because, previously, requesting a nonexistent message from the dispatch queue was blocking until the 5 seconds time-out was kicking in.

### 3.1.3. Integration Testing

Integration tests is the last layer of tests we'll be adding to be fully covered. These tests will actually be run against Mule running with your full configuration in place. We'll be limited to testing the paths that we can explore when exercising the system as a whole, from the outside. This means that some failure paths, like the one above that simulates a failure of the outbound JMS endpoint, will not be tested.

Though it is possible to use Maven to start Mule before running the integration tests, we recommend that you deploy your application on the container it will be running in in production (either Mule standalone or a Java EE container).

Since integration tests exercise the application as a whole with actual transports enabled, external systems will be affected when these tests will run. For example, in our case a JMS queue will receive a message: we will need to ensure this message has been received, which implies that no other system will consume it (or else we would have to check in these systems that they have received the expected message).

In shared environments, this is tricky to achieve and usually requires the agreement of all systems about the notion of test messages. These test messages exhibit certain characteristics (properties or content) so other systems realize they should not consume or process them.

To learn more about test messages and for more testing strategies and approaches, we suggest reading this excellent paper about "Test-Driven Development in Enterprise Integration Projects", from Hohpe and Istvanick <http://www.eaipatterns.com/docs/TestDrivenEAI.pdf>

Another very important aspect is the capacity to trace a message as it progresses through Mule services and reaches external systems: this is achieved by using unique correlation IDs on each message and consistently writing these IDs to log files. As you'll see it later on, we also rely on unique correlation IDs for integration testing. For now, here is our inbound HTTP endpoint refactored to ensure that the Mule correlation ID is set to the same message ID value that is returned in the OK acknowledgement message:

```
<http:endpoint name="NewWorkEndpoint"
    host="${web.host}"
    port="8080"
    path="api/work">
    <object-to-string-transformer/>
    <message-properties-transformer>
        <add-message-property key="MULE_CORRELATION_ID"
            value="#[message:id]" />
    </message-properties-transformer>
</http:endpoint>
```

Mule will do the rest: it will ensure that the correlation ID that is been set with the message properties transformer shown above, gets propagated to any internal flow or external system receiving the message.

#### Maven Failsafe to feel safe

In order to keep our example simple, we'll assume that no other system will attempt to consume the messages dispatched on the target JMS queue: they will be sitting there until we consume them.

To show that no specific tooling is needed to build integration tests, we'll build them in Java, as JUnit test cases, and will run them with Maven's failsafe plug-in . Feel free to use instead any tool you're more familiar with.

For our current needs, soapUI used in conjunction with HermesJMS would give us a nice graphical environment for creating and running integration tests. See <http://www.soapui.org/JMS/getting-started.html> for more information. Also note that soapUI can be run from Maven too: <http://www.soapui.org/Test-Automation/maven-2x.html>

Since the main entry point of our application is exposed over HTTP, we'll use HttpUnit in our tests. Let's look at our test case for invalid work submissions:

```
@Test
public void rejectInvalidWork() throws Exception
{
    String testPayload = "not_xml";
    ByteArrayInputStream payloadAsStream = new ByteArrayInputStream(testPayload.getBytes());

    WebConversation wc = new WebConversation();
    WebRequest request = new PostMethodWebRequest(WORK_API_URI, payloadAsStream, "text/plain");
    WebResponse response = wc.getResponse(request);

    assertEquals(200, response.getResponseCode());
    String responseText = response.getText();
    assertTrue(responseText.startsWith("NOT_XML"));
}
```

In this test, which is a Junit 4 annotated test, we send a bad payload to our work manager and ensure that it gets rejected as expected. The WORK\_API\_URI constant is of course pointing to the Mule instance that is tested.

The test for valid submissions is slightly more involved:

```

@Test
public void acceptValidWork() throws Exception
{
    String testPayload = "<valid_xml />";
    ByteArrayInputStream payloadAsStream = new ByteArrayInputStream(testPayload.getBytes());

    WebConversation wc = new WebConversation();
    WebRequest request = new PostMethodWebRequest(WORK_API_URI, payloadAsStream, "application/xml");
    WebResponse response = wc.getResponse(request);

    assertEquals(200, response.getResponseCode());
    String responseText = response.getText();
    assertTrue(responseText.startsWith("OK:"));

    String correlationId = responseText.substring(3);
    Message jmsMessage = consumeQueueMessageWithSelector("work", "JMSCorrelationID=" + correlationId +
    "", 5000L);

    assertTrue(jmsMessage instanceof TextMessage);
    assertEquals(testPayload, ((TextMessage) jmsMessage).getText());
}

private Message consumeQueueMessageWithSelector(String queueName,
                                                String selector,
                                                long timeout) throws JMSException
{
    ConnectionFactory connectionFactory = getConnectionFactory();
    Connection connection = connectionFactory.createConnection();
    connection.start();

    Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    MessageConsumer createConsumer = session.createConsumer(session.createQueue(queueName),
        selector);
    Message result = createConsumer.receive(timeout);
    connection.close();
    return result;
}

```

Note that `getConnectionFactory()` is specific to the JMS implementation in use and, as such, hasn't been included in the above code snippet.

The important take away is that we use the correlation ID returned by the Validator as a mean to select and retrieve the dispatched message from the target JMS queue. As you can see, Mule has propagated its internal correlation ID to the JMS-specific one, opening the door to this kind of characterization and tracking of test messages.

It's time to run these two tests with the Failsafe plug-in. By convention integration test classes are named `IT*` or `*IT` or `*ITCase` and are located under `src/it/java`. This path is not by default on a standard Maven project build path, so we will need a little bit of jiggery-pokery to make sure they're compiled and loaded. Because we do not want to always add the integration test source path to all builds, we create a Maven profile (named `it`) and store all the necessary configuration in it:

```

<profile>
  <id>it</id>
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>build-helper-maven-plugin</artifactId>
        <executions>
          <execution>
            <id>add-test-source</id>
            <phase>generate-test-sources</phase>
            <goals>
              <goal>add-test-source</goal>
            </goals>
            <configuration>
              <sources>
                <source>src/it/java</source>
              </sources>
            </configuration>
          </execution>
        </executions>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>failsafe-maven-plugin</artifactId>
        <executions>
          <execution>
            <id>integration-test</id>
            <goals>
              <goal>integration-test</goal>
            </goals>
          </execution>
          <execution>
            <id>verify</id>
            <goals>
              <goal>verify</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>httpunit</groupId>
      <artifactId>httpunit</artifactId>
      <version>1.7</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</profile>

```

With this configuration in place in your pom.xml, you can run:

```
mvn -Pit verify
```

to execute your first automated Mule integration tests.

Your Rating: 

Results:  0 rates

## Troubleshooting

# Troubleshooting

This section demonstrates how to troubleshoot Mule.

- Configuring Mule Stacktraces
- Logging
- Step Debugging

## Configuring Mule Stacktraces

### Configuring Mule Stacktraces

By default Mule will filter out some internal class references from stacktraces to produce a more readable output. This behavior can be controlled one of two ways:

#### Command-Line Arguments

Two command-line properties enable you to control this behavior using `-M-DpropertyName`:

- `mule.stacktrace.full` - when present, Mule will not filter stacktraces. Intended for Mule developers only, end user would probably never need to enable this.
- `mule.stacktrace.filter` - a comma-separated list of packages and/or classes to remove from the stacktraces (matched via `String.startsWith()`)

#### Using JMX

The Configuration MBean now exposes two new options.

- `FullStackTrace` - same as above
- `StackTraceFilter` - same as above

Note that these settings are shared between apps even though each app has its own Configuration MBean. That means that modifying it in one app will affect others.

Your Rating: 

Results:  0 rates

## Logging

### Logging

The less high-tech and most popular of all debugging techniques is the usage of log statements in order to follow the evolution of an application's state. In Mule, the state you're interested in resides in the messages that are flowing through your configuration and, possibly, custom code.

If you're running your Mule configuration from Eclipse, the log outputs will be visible right in Eclipse console window. If you're running Mule from the command line, the logs will then be visible in your OS console.

Mule's standalone logging configuration is stored in `<Mule Installation Directory>/conf/log4j.properties`: edit this file if you need to change the verbosity of the log output.

The log component is a quick and easy way to log the payload of an in-flight message. Add it anywhere in a message flow you want to probe your message:

```
<flow name="FlowWithLoggers">
    <http:inbound-endpoint address="http://localhost:8383/flowlog" />
    <log-component />
    <base64-encoder-transformer/>
    <log-component />
    <vm:outbound-endpoint path="next.in.line" />
</flow>
```

If you need more details about the message, a simple scripted logging component like the following can come handy:

```
<scripting:script name="Logger" engine="groovy">
<scripting:text>log.info(message); log.info(payload); message</scripting:text>
</scripting:script>
```

You can refer to it from anywhere in your flows:

```
<flow name="FlowWithLoggers">
<http:inbound-endpoint address="http://localhost:8383/flowlog" />
<scripting:component script-ref="Logger" />
<base64-encoder-transformer/>
<scripting:component script-ref="Logger" />
<vm:outbound-endpoint path="next.in.line" />
</flow>
```

Your Rating: 

Results:  0 rates

## Logging With Mule ESB 3.x

### Logging With Mule ESB 3.x

Mule 3.1.2 introduced new logging features:

1. Log file per-application
2. Applications can override default logging configuration
3. Logging configurations can be reloaded on the fly without restarting an app or Mule.

These features are supported in **standalone** Mule only, no embedded support possible.

#### Details:

1. Both log4j.xml and log4j.properties formats are supported. See <http://wiki.apache.org/logging-log4j/Log4jXmFormat>
2. log4j.xml takes precedence over log4j.properties if both found
3. Switching log configuration formats on the fly isn't supported. I.e. dropping a log4j.xml in when the system was configured using log4j.properties won't trigger the re-configuration. However, individual app re-deployment can change logging format (new config will be monitored once the app is done redeploying).
4. \$MULE\_HOME/conf/log4j.xml/properties is a top-level (container) log configuration.
5. Mule monitors the log config file for changes every 10 secs
6. For on-the-fly log config changes to happen, the config file must be a physical file on a disk, not a resource in a jar. In case of a jарped configuration file, logging will be configured once on startup without support for dynamic reconfiguration. In practice, this simply means that one has to put log4j config files in the app's 'classes' directory.
7. If there's an error in the logging config file, the logging subsystem will become unavailable and produce no output. Simply correct the error and save changes, Mule will pick them up in 10 seconds and reconfigure logging.
8. By default, dedicated log file is created for each app. All log files are located in \$MULE\_HOME/logs
9. Default filename pattern for a per-app log is 'mule-app-myapp.log', where 'myapp' is the name of the app.
10. Default appender is a DailyRollingFileAppender, for past dates other than today a timestamp is added to the filename in the 'yyyy-MM-dd' format
11. An app may **optionally** override logging configuration
12. Same rules about xml- vs properties-based config preference apply
13. Same rules about on-the-fly changes to logging configuration file apply
14. When an app overrides logging configuration, it should configure full logging including the root logger, appenders, etc. This also means that full power of log4j can be leveraged.
15. For convenience \${mule.home} can be used in log4j config settings for file paths.
16. It is possible to use the old-style single-file logging (note that dynamic log re-configuration won't be supported either). Specify -Dmule.simpleLog JVM startup property (actual value of the property doesn't matter, only its presence)

Your Rating: 

Results:  3 rates

## Step Debugging

### Step Debugging

There is only that much information that a log output capture: when more run-time context is required, your best option is to step debug into your running Mule development instance.

## When Running From Eclipse

Select the "Debug As > Mule Server" start-up mode from your configuration file. Any breakpoint you will have set in your custom code or Mule's source code will suspend the execution and will take you to Eclipse's Debug perspective.

## When Running Mule Standalone

You need to start Mule with the -debug to activate remote debugging, which gives (on Linux):

```
$MULE_HOME/bin/mule -debug
```

With this option, Mule will start normally with the only difference being the following message logged in the console:

```
Listening for transport dt_socket at address: 5005
```

Mule is now remote debuggable on port 5005. Should you want to use another port or configure the JVM to ponder until a remote debugger gets attached, edit the mule start-up script that is relevant for your operating system and change the parameters found in the JPDA\_OPTS property.

Switch to Eclipse and go the the "Run > Debug Configurations" menu. From there, create a "Remote Java Application" configuration. Ensure the port is 5005.

Also, in order to step debug with the source code attached, you need to add the source code of your Mule local instance as an external archive on the second tab of the configuration panel.

You are now ready: click the "Debug" button at the lower right corner of the configuration screen: the local Mule instance will resume its start-up sequence and soon you'll be ready to step debug.

Your Rating:  Results:  0 rates

## Team Development with Mule

### Team Development with Mule

Topics in these books have addressed your local workstation, and a single project with a single configuration file. When working on a team, the Mule project will increase in size, will increase in its number of developers, and must run in other environments, such as test and production. Here are some practices that will make such growth possible.

The different approaches to modularizing Mule configurations and applications are all opportunities for splitting work across teams, whether these teams work on the same overarching project or on different projects with an accent put on reuse.

Download a PDF version of this section [here](#).

- Modularizing Your Configuration Files for Team Development
- Using Side-by-Side Configuration Files
- Using Parameters in Your Configuration Files
- Using Modules In Your Application
- Sharing Custom Code
- Sharing Custom Configuration Fragments
- Sharing Custom Configuration Patterns
- Sharing Applications

Your Rating:  Results:  1 rates

## Modularizing Your Configuration Files for Team Development

### Modularizing Your Configuration Files for Team Development

Though it may seem convenient to keep all your Mule configuration in one place, the reality is that a gigantic XML file quickly becomes unmanageable. This is why it is recommended to split monolithic configurations into several files and leverage Mule's capacity to load multiple configuration files at application start-up time. Moreover, splitting configurations into multiple fragments encourages re-use across teams.

Mule offers two options for loading several configuration files:

- \*side-by-side: provide a list of independent configuration files to load,
- \*imported: have one configuration file import several others, which in-turn can import other files.

In practice, it is common to use both approaches simultaneously.

Don't forget that all the configuration files end up loaded in the same context; therefore you should be careful and use unique names for all your configuration elements. Mule will refuse to load an application whose configuration files contain name conflicts.

How can you determine what constitutes good separation lines between configuration fragments? Here are a few rules of thumb:

- \*Business domains usually form a natural border that can be used to separate configuration elements
- \*Keeping together elements that have similar reasons for change reduces the risk of impacting unrelated aspects of your application
- \*Technical aspects, like administrative components, security or Spring beans configuration, define good lines of demarcation
- \*Extracting a side-by-side transport configuration (connectors and endpoints) facilitates functional testing (discussed in section 3). Note that it is not intended to take care of environment specific transport configuration, which is dealt with properties files (discussed later on)
- \*And, last but not least, re-use across teams and projects (also discussed later on)

## Imported configuration files

Mule relies on Spring XML configuration for importing configuration files into each other.

Here is the main configuration file illustrated above, which takes care of importing the three other configuration elements:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <spring:beans>
        <spring:import resource="domain-A-config.xml" />
        <spring:import resource="domain-B-config.xml" />
        <spring:import resource="admin-config.xml" />
    </spring:beans>
</mule>
```

Your Rating: 

Results:  0 rates

## Using Side-by-Side Configuration Files

### Using Side-by-Side Configuration Files

Side by side configurations are independent and require nothing specific to work except to let Mule know it should load them.

To do so, create a file named mule-deploy.properties in the same directory that contains your configuration files and add configuration similar to the following one, but of course with your configuration file names:

```
config.resources=mule-main-config.xml,mule-transports-config.xml
```

With this in place, Mule will know which configuration files it should load when deploying your application.

If you're starting your application from Eclipse, go to the parameters screen of the "Run" configuration you use and select your files there.

Your Rating: 

Results:  0 rates

# Using Parameters in Your Configuration Files

## Using Parameters in Your Configuration Files

When an application gets deployed in different environments, like QA, pre-production or production, it usually needs to be configured differently as server names, credentials and other similar parameters will vary.

As a developer facing this kind of variability, your goal is to produce a single Mule application for all your environments and to externalize all the environment-specific configuration parameters. This is the key to reproducible deployments.

Consider externalizing other aspects of your configuration, like time-out values, polling frequencies, etc... even if they don't vary between environments. This will facilitate tuning and experimenting as the whole Mule application would become configurable through a single properties file.

In Mule, you achieve this by using Spring's property placeholder resolution mechanism. Consider the following Mule configuration fragment that defines an HTTP endpoint pointing to a password protected web resource:

```
<http:endpoint name="ProtectedWebResource"
    user="${web.rsc.user}"
    password="${web.rsc.password}"
    host="${web.rsc.host}"
    port="80"
    path="path/to/resource" />
```

The variables bits are clearly visible: the user, password and host can vary for each environment where this endpoint gets deployed in. To provide values for these variables, we use a standard Java properties file:

```
web.rsc.user=alice
web.rsc.password=s3cr3t
web.rsc.host=www.acme.com
```

Use a consistent naming strategy for your properties and make them unique across applications: this will greatly facilitate re-use across teams.

You also need to configure Spring's property placeholder configurer. Instead of configuring Spring to load a single properties file, follow this approach:

- Configure Spring to load a default properties file and another file containing overrides.
- Ship a default properties file with values applicable for developers' workstations inside your Mule application deployable.
- Create the properties override file only in the environments where it's needed and with only the properties that actually need to be overridden.

The advantages of this approach are:

- Developers don't need to deploy and run the application locally.
- The ops team only needs to work with the set of properties they have to configure for a particular environment.

Here is a method of accomplishing this:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <spring:beans>
        <context:property-placeholder
            location="classpath:my-mule-app.properties,
                      classpath:my-mule-app-override.properties" />
    </spring:beans>
</mule>

```

With this in place, add a `my-mule-app.properties` file in your application resources directory (`src/main/app` for a Mule application Maven project) and put default and development environment values in it. To override some values, create a `my-mule-app-override.properties` file and drop it in `$MULE_HOME/conf`.

If your ops team can't drop files in Mule's directory hierarchy, the alternative is to configure the placeholder configurer to pick up the override file from a well-known location, as shown here:

```

<context:property-placeholder
    location="classpath:my-mule-app.properties,
              file:///etc/mule/conf/my-mule-app-override.properties" />

```

Use unique file names for your properties files to ease the burden on sysadmins. A good strategy is to use the application name or ID in the default and override properties file names.

Should you need to encrypt passwords in your properties file, consider using Jasypt's subclass of Spring's placeholder configurer. For more information, check the section named "Encryption-aware Spring's property configurers" on this page:  
<http://www.jasypt.org/encrypting-configuration.html>

Your Rating:  Results:  0 rates

## Using Modules In Your Application

### Using Modules In Your Application

Mule applications themselves can provide an additional way to modularize your project.

Thanks to Mule's capacity to run and hot-redeploy applications side-by-side in the same instance, such a coarse-grained approach to modularity is useful when keeping elements of your application running while others could go through some maintenance operations.

For optimum modularity:

- Consider what services are tightly interrelated and keep them together in the same Mule application: they will form sub-systems of your whole solution.
- Establish communication channels between the different Mule applications: the VM transport will not be an option here, as it can't be used across different applications. Prefer the TCP or HTTP transports for synchronous channels and JMS for asynchronous ones.
- Watch out for port conflicts: two applications cannot deploy inbound endpoints bound to the same ports.

Your Rating:  Results:  0 rates

## Sharing Custom Code

### Sharing Custom Code

Besides all the common code that exists in a company, there are Mule specific programmatic artifacts that are worth considering sharing.

Let's name a few:

\*Custom transformers - performing operations that none of the Mule stock transformers can perform (see: <http://www.mulesoft.org/documentation/display/MULE3USER/Creating+Custom+Transformers>),

\*Custom components - typically Mule-aware or non-business oriented components, as business components are usually simple POJOs coming from pre-existing projects (see: <http://www.mulesoft.org/documentation/display/MULE3USER/Developing+Components>),

\*Custom expression evaluators - for the cases when a specific message extraction capacity is needed in several places of an application (see: <http://www.mulesoft.org/documentation/display/MULE3USER/Creating+Expression+Evaluators>).

The most convenient way to share custom code across team is to rely on Maven's dependency management mechanism. Here is an extract of a pom.xml referring to common code stored in a shared Maven library:

```
<dependency>
  <groupId>com.acme.mulings</groupId>
  <artifactId>common-mule-project</artifactId>
  <version>1.3</version>
</dependency>
```

The Mule build plug-in will automatically bundle these extra dependencies in the /lib directory of the deployable application. In this case, the common-mule-project-1.3.jar will be added to this directory at build time, ready to be deployed and made available to the application running on Mule.

Your Rating:  Results:  0 rates

## Sharing Custom Configuration Fragments

### Sharing Custom Configuration Fragments

Thanks to its element naming and referencing strategy, the Mule configuration configuration mechanism supports re-using fragments of configuration between teams. This is very convenient for sharing complex, repetitive or critical bits of configuration.

Concretely, here are some type of configuration elements that can be shared across projects:

\*Connector configurations - a connector that need a complex configuration, say with specific transport level details, is a great candidate for re-use  
\*Endpoint definitions - defining global endpoints facilitates testing (as discussed in section 3) but also promotes re-use as they become sharable  
\*Pre-configured transformers - some transformers, like the XSL-T one, can require a fair bit of configuration, making them likely to be shared  
\*Sub-flows - a particular chain of message processors can form a bit of configuration that is worth re-using (we'll look at an example right after this)  
\*Flows - it can make sense for a service to exist in several Mule applications: in that case sharing a flow is the best way to go. If this flow is configured to rely on global endpoints, the application using this flow retains full control on what protocol will actually be used

Let's take a look at a configuration file containing a sub-flow that defines a standard chain of transformer we want to share with all our applications so they can use it too:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

  <flow name="DefaultTransformers">
    <append-string-transformer message=">>" />
    <base64-encoder-transformer />
  </flow>
</mule>
```

As you can see, nothing makes this flow a sub-flow except that it has no inbound router. It's when if gets re-used that it truly becomes a sub-flow:

```

<flow name="FlowUsingSubflow">
    ...
    <log-component />
    <flow-ref name="DefaultTransformers" />
    ...
</flow>

```

Now, how does an application refer to a common flow like this? In truth, what we're sharing are complete configuration files. We do so with the standard Maven dependency mechanism described before: the Maven project containing common configuration files (containing common configuration fragments) simply needs to export them as resources in a JAR.

The only (simple) trick is that the Mule application that wants to use a shared configuration file should import it by locating it on the classpath, as shown here:

```
<spring:import resource="classpath:mule-common-config.xml" />
```

Your Rating:  Results:  1 rates

## Sharing Custom Configuration Patterns

### Sharing Custom Configuration Patterns

Mule supports the notion of configuration patterns that are specialized flow tailored to serve a very specific purpose in the least amount of XML.

The list of configuration patterns that ship with Mule are listed here:

<http://www.mulesoft.org/documentation/display/MULE3USER/Configuration+Patterns>

Mule also provides the necessary tooling for creating a pattern catalog and patterns within it, allowing teams to encapsulate domain specific knowledge in custom configuration elements and share it across teams.

In essence, a pattern catalog is not different than a standard module: teams sharing pattern catalogs will only need to add a new dependency in their pom.xml. Of course, unlike what it's done with Mule standard modules, the dependency scope will have to be "compile" and not "provided."

Your Rating:  Results:  0 rates

## Sharing Applications

### Sharing Applications

Applications are self-contained archives and can therefore be shared via a simple file or web server.

You must document the name and content of the override properties file that each application is expecting.

## Sustainable Software Development Practices with Mule

### Sustainable Software Development Practices with Mule

The following describe sustainable software development practices with Mule.

- Reproducible Builds
- Continuous Integration
- Repeatable Deploys

## Reproducible Builds

## Reproducible Builds

You should be able to build a particular version of your Mule project at any point of time. This facilitates maintenance as the versions of your projects in production will certainly not be the latest ones.

The following will help you achieve this goal:

- Source control all your Mule project, like any other project (remember not to check-in Eclipse specific files). Branching, merging and tagging are all applicable practices to Mule projects
- Manage dependencies strictly: using Maven and an in-house repository manager (like Nexus) will get you a long way there.

Your Rating:  Results:  1 rates

## Continuous Integration

### Continuous Integration

With all the previous emphasis on testing, setting up continuous integration for your project should look like a no-brainer. By using Maven as your build tool, you'll be able to set-up a build that gets triggered on every project change and run all its unit and functional tests automatically.\

There are plenty of continuous integration tools out there. To name a few: Hudson, TeamCity and Bamboo are popular choices.

If you've opted for using real transports in your functional test cases, you will have to pay attention to potential port conflicts that could occur in your continuous build server. Mule provides a convenient utility that can locate available ports and make them available as properties that your test specific transport configuration file can use. Read more about this tool here:

<http://www.mulesoft.org/documentation/display/MULE3USER/Using+Dynamic+Ports+in+Mule+Test+Cases>

If your target deployable is a web application and not a Mule application, consider using Jitr (<http://jitr.org>) for running your functional tests and avoiding port conflicts.

Your Rating:  Results:  1 rates

## Repeatable Deploys

### Repeatable Deploys

As mentioned in 2.2, it is highly desirable that your build produces the same deployable unit (ie. Mule or Web application depending on your deployment model) for all target environments. This is made possible by the externalization of all the configuration variables in properties files.

If the use of external overrides is not an option for you and you have to create different deployable units for your different environments, avoid manual operations at all cost. A good approach is to use a different Maven profile for each of your environments in order to control the build in a strict and reproducible manner.

Your Rating:  Results:  0 rates

## Reference Materials for Mule ESB 3

### Reference Materials for Mule ESB 3

- Configuration Reference
- Transport Reference
- Modules Reference
- API Reference 3.1.1
- API Reference 3.1.0
- Schemadocs Reference 3.1.1
- Schemadocs Reference 3.1.0
- Test API Reference 3.1.1
- Test API Reference 3.1.0

## Documentation for Previous Releases

- Mule 1.x Getting Started
- Mule 1.x User Guide
- Mule 2.x Getting Started
- Mule 2.x User Guide
- Using the Management Console (Previous release, EE only)

## Configuration Reference

### Configuration Reference

This topic relates to the most recent version of Mule ESB

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

NOTE: With Mule 3.0 comes a release of new Javadoc-style schema documentation, making it easier to familiarize yourself with all the elements and attributes permissible in xml configuration files.

- BPM Configuration Reference
- Global Settings Configuration Reference
- Model Configuration Reference
- Service Configuration Reference
- Endpoint Configuration Reference
- Routers
  - Inbound Router Configuration Reference
  - Outbound Router Configuration Reference
  - Asynchronous Reply Router Configuration Reference
  - Catch-all Strategy Configuration Reference
- Filters Configuration Reference
- Transformers Configuration Reference
- Component Configuration Reference
- Entry Point Resolver Configuration Reference
- Exception Strategy Configuration Reference
- Properties Configuration Reference
- Notifications Configuration Reference
- Transactions Configuration Reference
- Expressions Configuration Reference
- Security Manager Configuration Reference

For configuration reference on transports, see [Transports Reference](#). For modules, see [Modules Reference](#).

For transports and modules contributed by the community, see [MuleForge Active Projects](#).

## Asynchronous Reply Router Configuration Reference

### Asynchronous Reply Router Configuration Reference

This page provides details on the elements you configure for [asynchronous reply routers](#). This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page.

cache: Unexpected program error: java.lang.NullPointerException

#### Single async reply router

Configures a Single Response Router. This will return the first message it receives on a reply endpoint and will discard the rest.

#### Attributes of `<single-async-reply-router...>`

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of `<single-async-reply-router...>`

Name	Cardinality	Description
------	-------------	-------------

abstract-inbound-endpoint	0..*	The endpoint used to receive the response(s) on. A placeholder for inbound endpoint elements. Inbound endpoints receive messages from the underlying transport. The message payload is then delivered to the component for processing.
---------------------------	------	--

cache: Unexpected program error: java.lang.NullPointerException

### Collection async reply router

Configures a Collection Response Router. This will return a MuleMessageCollection message type that will contain all messages received for the current correlation.

#### Attributes of <collection-async-reply-router...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of <collection-async-reply-router...>

Name	Cardinality	Description
abstract-inbound-endpoint	0..*	The endpoint used to receive the response(s) on. A placeholder for inbound endpoint elements. Inbound endpoints receive messages from the underlying transport. The message payload is then delivered to the component for processing.

cache: Unexpected program error: java.lang.NullPointerException

### Custom async reply router

#### Attributes of <custom-async-reply-router...>

Name	Type	Required	Default	Description
class	class name	yes		A fully qualified Java class name of the router to use. The router should either extend org.mule.routing.response.AbstractResponseRouter or org.mule.routing.response.AbstractResponseAggregator.

#### Child Elements of <custom-async-reply-router...>

Name	Cardinality	Description
abstract-inbound-endpoint	0..*	The endpoint used to receive the response(s) on. A placeholder for inbound endpoint elements. Inbound endpoints receive messages from the underlying transport. The message payload is then delivered to the component for processing.
spring:property	0..*	Spring-style property elements so that custom configuration can be configured on the custom router.

Your Rating: 

Results:  1 rates

## Catch-all Strategy Configuration Reference

### Catch-all Strategy Configuration Reference

This page provides details on the elements you configure for catch-all strategies. This information is pulled directly from mule.xsd and is cached. If the information appears to be out of date, refresh the page.

cache: Unexpected program error: java.lang.NullPointerException

#### Logging catch all strategy

Does nothing with the message but simply logs (using the WARN log level) the fact that the message was not dispatched because no routing path was defined.

#### Attributes of <logging-catch-all-strategy...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### ***Child Elements of <logging-catch-all-strategy...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Custom catch all strategy**

##### ***Attributes of <custom-catch-all-strategy...>***

Name	Type	Required	Default	Description
class	class name	yes		Fully qualified class name of the custom catch-all strategy to be used.

#### ***Child Elements of <custom-catch-all-strategy...>***

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

#### **Forwarding catch all strategy**

Forwards the message to the specified endpoint if no outbound routers match.

##### ***Attributes of <forwarding-catch-all-strategy...>***

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### ***Child Elements of <forwarding-catch-all-strategy...>***

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.

cache: Unexpected program error: java.lang.NullPointerException

#### **Custom forwarding catch all strategy**

##### ***Attributes of <custom-forwarding-catch-all-strategy...>***

Name	Type	Required	Default	Description
class	class name	yes		Fully qualified class name of the custom forwarding catch-all strategy to be used.

#### ***Child Elements of <custom-forwarding-catch-all-strategy...>***

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
spring:property	0..*	Spring-style property element for custom configuration.

Your Rating: 

Results:  0 rates

## **Component Configuration Reference**

## Component Configuration Reference

This page provides details on the elements you configure for components. This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on components, see [Configuring Components](#).

cache: Unexpected program error: java.lang.NullPointerException

### Component

A simple POJO (Plain Old Java Object) component that will be invoked by Mule when a message is received. The class or object instance to be used can be specified using a child object factory element, or via the 'class' attribute. If the 'class' attribute is used, an object factory cannot be configured as well. Using the 'class' attribute is equivalent to using the prototype object factory ('prototype-object' child element).

#### Attributes of <component...>

Name	Type	Required	Default	Description
class	class name	no		Specifies a component class. This is a shortcut that is equivalent to providing a 'prototype-object' element.

#### Child Elements of <component...>

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.
abstract-object-factory	0..1	Object factory used to obtain the object instance that will be used for the component implementation. The object factory is responsible for object creation and may implement different patterns, such as singleton or prototype, or look up an instance from other object containers.
abstract-lifecycle-adapter-factory	0..1	
binding	0..*	A binding associates a Mule endpoint with an injected Java interface. This is like using Spring to inject a bean, but instead of calling a method on the bean, a message is sent to an endpoint.
abstract-entry-point-resolver-set	0..1	A placeholder for entry point resolver set elements. These combine a group of entry point resolvers, trying them in turn until one succeeds.
abstract-entry-point-resolver	0..1	A placeholder for an entry point resolver element. Entry point resolvers define how payloads are delivered to Java code by choosing the method to call.

cache: Unexpected program error: java.lang.NullPointerException

### Pooled component

A pooled POJO (Plain Old Java Object) component that will be invoked by Mule when a message is received. The instance can be specified via a factory or a class.

#### Attributes of <pooled-component...>

Name	Type	Required	Default	Description
class	class name	no		Specifies a component class. This is a shortcut that is equivalent to providing a 'prototype-object' element.

#### Child Elements of <pooled-component...>

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

abstract-object-factory	0..1	Object factory used to obtain the object instance that will be used for the component implementation. The object factory is responsible for object creation and may implement different patterns, such as singleton or prototype, or look up an instance from other object containers.
abstract-lifecycle-adapter-factory	0..1	
binding	0..*	A binding associates a Mule endpoint with an injected Java interface. This is like using Spring to inject a bean, but instead of calling a method on the bean, a message is sent to an endpoint.
abstract-entry-point-resolver-set	0..1	A placeholder for entry point resolver set elements. These combine a group of entry point resolvers, trying them in turn until one succeeds.
abstract-entry-point-resolver	0..1	A placeholder for an entry point resolver element. Entry point resolvers define how payloads are delivered to Java code by choosing the method to call.
abstract-pooling-profile	0..1	Characteristics of the object pool.

cache: Unexpected program error: java.lang.NullPointerException

## Pooling profile

### Attributes of <pooling-profile...>

Name	Type	Required	Default
maxActive	string	no	
maxIdle	string	no	
initialisationPolicy	INITIALISE_NONE/INITIALISE_ONE/INITIALISE_ALL	no	INITIALISE_ONE

exhaustedAction	WHEN_EXHAUSTED_GROW/WHEN_EXHAUSTED_WAIT/WHEN_EXHAUSTED_FAIL	no	WHEN_EXHAUSTED
maxWait	string	no	

**Child Elements of <pooling-profile...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException  
 cache: Unexpected program error: java.lang.NullPointerException

**Echo component**

Logs the message and returns the payload as the result.

**Attributes of <echo-component...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

**Child Elements of <echo-component...>**

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

cache: Unexpected program error: java.lang.NullPointerException

**Log component**

Logs the message content (or content length if it is a large message).

**Attributes of <log-component...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

**Child Elements of <log-component...>**

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

cache: Unexpected program error: java.lang.NullPointerException

## Null component

Throws an exception if it receives a message.

### Attributes of <null-component...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <null-component...>

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

cache: Unexpected program error: java.lang.NullPointerException

## Spring object

### Attributes of <spring-object...>

Name	Type	Required	Default	Description
bean	name (no spaces)	no		Name of Spring bean to look up.

### Child Elements of <spring-object...>

Name	Cardinality	Description
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

## Singleton object

### Attributes of <singleton-object...>

Name	Type	Required	Default	Description
class	class name	no		Class name

### Child Elements of <singleton-object...>

Name	Cardinality	Description
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

## Prototype object

### Attributes of <prototype-object...>

Name	Type	Required	Default	Description
class	class name	no		Class name

### Child Elements of <prototype-object...>

Name	Cardinality	Description
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

## Custom lifecycle adapter factory

### Attributes of <custom-lifecycle-adapter-factory...>

Name	Type	Required	Default	Description
class	class name	yes		An implementation of the LifecycleAdapter interface.

### Child Elements of <custom-lifecycle-adapter-factory...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

## Binding

A binding associates a Mule endpoint with an injected Java interface. This is like using Spring to inject a bean, but instead of calling a method on the bean, a message is sent to an endpoint.

### Attributes of <binding...>

Name	Type	Required	Default	Description
interface	class name	yes		The interface to be injected. A proxy will be created that implements this interface by calling out to the endpoint.
method		no		The method on the interface that should be used. This can be omitted if the interface has a single method.

### Child Elements of <binding...>

Name	Cardinality	Description
abstract-outbound-endpoint	1..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.

## Interceptors

See [Using Interceptors](#).

## Entry Point Resolvers

See [Entry Point Resolver Configuration Reference](#).

Your Rating:  5 stars

Results:  0 rates

## Endpoint Configuration Reference

### Endpoint Configuration Reference

This page provides details on the elements you configure for endpoints. This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on endpoints, see [Configuring Endpoints](#).

cache: Unexpected program error: java.lang.NullPointerException

#### Inbound endpoint

An inbound endpoint receives messages via the associated transport. As with global endpoints, each transport implements its own inbound endpoint element.

##### **Attributes of <inbound-endpoint...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mime-type	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		

##### **Child Elements of <inbound-endpoint...>**

Name	Cardinality	Description
------	-------------	-------------

response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

## Outbound endpoint

An outbound endpoint sends messages via the associated transport. As with global endpoints, each transport implements its own outbound endpoint element.

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.

responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		

#### **Child Elements of <outbound-endpoint...>**

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

#### **Endpoint**

A global endpoint, which acts as a template that can be used to construct an inbound or outbound endpoint elsewhere in the configuration by referencing the global endpoint name. Each transport implements its own endpoint element, with a more friendly syntax, but this generic element can be used with any transport by supplying the correct address URI. For example, "vm://foo" describes a VM transport endpoint.

#### **Attributes of <endpoint...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).

address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		

#### Child Elements of <endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

Your Rating: 

Results:  1 rates

## Exception Strategy Configuration Reference

## Exception Strategy Configuration Reference

This page provides details on the elements you configure for exception strategies. This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on exception strategies, see [Error Handling](#).

cache: Unexpected program error: java.lang.NullPointerException

### Default exception strategy

Provides default exception handling.

#### Attributes of <default-exception-strategy...>

Name	Type	Required	Default	Description
enableNotifications	boolean	no		Determines whether ExceptionNotifications will be fired from this strategy when an exception occurs. Default is true.
stopMessageProcessing	boolean	no	false	Stop the flow/service when an exception occurs. You will need to restart the flow/service manually after this (e.g, using JMX).

#### Child Elements of <default-exception-strategy...>

Name	Cardinality	Description
commit-transaction	0..1	Defines when a current transaction gets committed based on the name of the exception caught. You can set a comma-separated list of wildcard patterns that will be matched against the fully qualified classname of the current exception. Patterns defined for this element will leave the current transaction (if any) untouched and allow it to be committed.
rollback-transaction	0..1	Defines when a current transaction gets rolled back based on the name of the exception caught. You can set a comma-separated list of wildcard patterns that will be matched against the fully qualified classname of the current exception. Patterns defined for this element will roll back the current transaction (if any).
abstract-message-processor	0..1	A message processor A placeholder for message processor elements.
abstract-outbound-endpoint	0..1	An outbound endpoint A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.

cache: Unexpected program error: java.lang.NullPointerException

### Custom exception strategy

A user-defined exception strategy.

#### Attributes of <custom-exception-strategy...>

Name	Type	Required	Default	Description
enableNotifications	boolean	no		Determines whether ExceptionNotifications will be fired from this strategy when an exception occurs. Default is true.
class	class name	yes		A class that implements the ExceptionListener interface. In addition, if an 'outbound-endpoint' element is specified, it is set as an "endpoint" bean property.

#### Child Elements of <custom-exception-strategy...>

Name	Cardinality	Description
commit-transaction	0..1	Defines when a current transaction gets committed based on the name of the exception caught. You can set a comma-separated list of wildcard patterns that will be matched against the fully qualified classname of the current exception. Patterns defined for this element will leave the current transaction (if any) untouched and allow it to be committed.
rollback-transaction	0..1	Defines when a current transaction gets rolled back based on the name of the exception caught. You can set a comma-separated list of wildcard patterns that will be matched against the fully qualified classname of the current exception. Patterns defined for this element will roll back the current transaction (if any).

abstract-message-processor	0..1	A message processor A placeholder for message processor elements.
abstract-outbound-endpoint	0..1	An outbound endpoint A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
spring:property	0..*	Spring-style property element for custom configuration.

Your Rating: 

Results:  2 rates

## Filters Configuration Reference

### Filters Configuration Reference

[ Filter ] [ Not filter ] [ And filter ] [ Or filter ] [ Wildcard filter ] [ Expression filter ] [ Regex filter ] [ Message property filter ] [ Exception type filter ] [ Payload type filter ] [ Custom filter ] [ Encryption security filter ] [ Jxpath filter ] [ Jaxen filter ] [ Xpath filter ] [ Schema validation filter ]

For more information on filters, see [Using Filters](#).

cache: Unexpected program error: java.lang.NullPointerException

#### Filter

A filter that is defined elsewhere (at the global level, or as a Spring bean).

##### Attributes of <filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
ref	string	yes		The name of the filter to use.

##### Child Elements of <filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Not filter

Inverts the enclosed filter. For example, if the filter would normally return true for a specific message, it will now return false, and vice versa.

##### Attributes of <not-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.

##### Child Elements of <not-filter...>

Name	Cardinality	Description
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.

cache: Unexpected program error: java.lang.NullPointerException

#### And filter

Returns true only if all the enclosed filters return true.

##### Attributes of <and-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.

#### ***Child Elements of <and-filter...>***

Name	Cardinality	Description
abstract-filter	2..*	A placeholder for filter elements, which control which messages are handled.

cache: Unexpected program error: java.lang.NullPointerException

#### **Or filter**

Returns true if any of the enclosed filters returns true.

#### ***Attributes of <or-filter...>***

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.

#### ***Child Elements of <or-filter...>***

Name	Cardinality	Description
abstract-filter	2..*	A placeholder for filter elements, which control which messages are handled.

cache: Unexpected program error: java.lang.NullPointerException

#### **Wildcard filter**

A filter that matches string messages against wildcards. It performs matches with "", **for example**, "jms.events." would catch "jms.events.customer" and "jms.events.receipts". This filter accepts a comma-separated list of patterns, so more than one filter pattern can be matched for a given argument: "jms.events., jms.actions." will match "jms.events.system" and "jms.actions" but not "jms.queue".

#### ***Attributes of <wildcard-filter...>***

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
pattern	string	yes		The property name and optionally a value to use when matching. If the expression is just a property name, the filter will check that the property exists. Users can also use '=' and '!=' to determine a specific value for a property.
caseSensitive	boolean	no	true	If false, the comparison ignores case.

#### ***Child Elements of <wildcard-filter...>***

Name	Cardinality	Description

cache: Unexpected program error: java.lang.NullPointerException

#### **Expression filter**

A filter that can evaluate a range of expressions. It supports some base expression types such as header, payload (payload type), regex, and wildcard.

#### ***Attributes of <expression-filter...>***

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
evaluator	header/payload-type/exception-type/wildcard/regex/ognl>xpath/groovy/bean/custom/registry	yes		The evaluator type. Used to build the expression. All the evaluators support header, payload, exception, wildcard, regular expression, OGNL, XPath, Groovy, Bean, Custom and Registry.
expression	string	yes		The expression to evaluate. Can be a simple string or an expression using properties.
customEvaluator	name (no spaces)	no		Must set evaluator with expression if it is not defined at the global level.
nullReturnsTrue	boolean	no		Whether null returns true or false.

#### Child Elements of <expression-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Regex filter

A filter that matches string messages against a regular expression. The Java regular expression engine (java.util.regex.Pattern) is used.

#### Attributes of <regex-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
pattern	string	yes		The property name and optionally a value to use when matching. If the expression is just a property name, the filter will check that the property exists. Users can also use '=' and '!=' to determine a specific value for a property.
flags	string	no		Comma-separated list of flags for compiling the pattern. Valid values are CASE_INSENSITIVE, MULTILINE, DOTALL, UNICODE_CASE and CANON_EQ.

#### Child Elements of <regex-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Message property filter

A filter that matches properties on a message. This can be very useful, as the message properties represent all the meta information about the message from the underlying transport, so for a message received over HTTP, you can check for HTTP headers and so forth. The pattern should be expressed as a key/value pair, such as "propertyName=value". If you want to compare more than one property, you can use the logic filters for And, Or, and Not expressions. By default, the comparison is case sensitive, which you can override with the 'caseSensitive' property.

#### Attributes of <message-property-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
pattern	string	yes		The property name and optionally a value to use when matching. If the expression is just a property name, the filter will check that the property exists. Users can also use '=' and '!=' to determine a specific value for a property.
caseSensitive	boolean	no	true	If false, the comparison ignores case.
scope	inbound/invocation/outbound/session/application	no	outbound	Property scope to lookup the value from (default: outbound)

#### Child Elements of <message-property-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Exception type filter

A filter that matches the type of an exception.

#### Attributes of <exception-type-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
expectedType	class name	yes		The expected class used in the comparison.

#### Child Elements of <exception-type-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Payload type filter

A filter that matches the type of the payload.

#### Attributes of <payload-type-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
expectedType	class name	yes		The expected class used in the comparison.

#### Child Elements of <payload-type-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Custom filter

A user-implemented filter.

#### Attributes of <custom-filter...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the filter so that other elements can reference it. Required if the filter is defined at the global level.
class	class name	no		An implementation of the Filter interface.

#### Child Elements of <custom-filter...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### Encryption security filter

A filter that provides password-based encryption.

#### Attributes of <encryption-security-filter...>

Name	Type	Required	Default	Description
strategy-ref	string	no		The name of the encryption strategy to use. This should be configured using the 'password-encryption-strategy' element, inside a 'security-manager' element at the top level.

#### Child Elements of <encryption-security-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

### Jxpath filter

Filters messages based on XPath expressions using JXPath.

#### Attributes of <jxpath-filter...>

Name	Type	Required	Default	Description
lenient	boolean	no	true	Whether or not errors are thrown if the XPath expression doesn't exist.
expectedValue	string	no		The expected value of the XPath expression evaluation. If the expected value matches the evaluation, the filter returns true.

#### Child Elements of <jxpath-filter...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as prefix and uri attributes. The prefix can then be used inside the expression.
context-property	0..*	A property that will be made available to the filter context. Expression Evaluators can be used to grab these properties from the message at runtime.

cache: Unexpected program error: java.lang.NullPointerException

## Jaxen filter

The Jaxen filter allows you to route messages based on XPath expressions. The Jaxen filter is generally faster than the JXPath filter and should be considered the first choice when using an XPath filter.

### Attributes of <jaxen-filter...>

Name	Type	Required	Default	Description
expectedValue	string	no		The expected value of the XPath expression evaluation. If the expected value matches the evaluation, the filter returns true.

### Child Elements of <jaxen-filter...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as prefix and uri attributes. The prefix can then be used inside the expression.
context-property	0..*	A property that will be made available to the filter context. Expression Evaluators can be used to grab these properties from the message at runtime.

cache: Unexpected program error: java.lang.NullPointerException

## Xpath filter

The XPath filter uses the JAXP libraries to filter XPath expressions. Available as of Mule 2.2.

### Attributes of <xpath-filter...>

Name	Type	Required	Default	Description
expectedValue	string	no		The expected value of the XPath expression evaluation. If the expected value matches the evaluation, the filter returns true.

### Child Elements of <xpath-filter...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as prefix and uri attributes. The prefix can then be used inside the expression.

cache: Unexpected program error: java.lang.NullPointerException

## Schema validation filter

The schema validation filter uses the JAXP libraries to validate your message against a schema. Available as of Mule 2.2.

### Attributes of <schema-validation-filter...>

Name	Type	Required	Default	Description
schemaLocations	string	no		The path to the schema file. You can specify multiple schema locations for validation.
schemaLanguage	string	no		The schema language to use. The default is "http://www.w3.org/2001/XMLSchema".
returnResult	boolean	no	true	Whether the filter should cache the result of the XML. If this is false, the filter will be more efficient, but it won't allow you to read the XML again.

### Child Elements of <schema-validation-filter...>

Name	Cardinality	Description

Your Rating: Results:  1 rates

## Global Settings Configuration Reference

### Global Settings Configuration Reference

This page provides details on the global settings you configure at the root level of a Mule configuration. Some of this information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on configuration, see [About Mule Configuration](#). For information on threading profiles, see [Tuning Performance](#).

cache: Unexpected program error: java.lang.NullPointerException

#### Configuration

Specifies defaults and general settings for the Mule instance.

#### Attributes of <configuration...>

Name	Type	Required	Default	Description
defaultResponseTimeout	string	no	10000	The default period (ms) to wait for a synchronous response.
defaultTransactionTimeout	string	no	30000	The default timeout (ms) for transactions. This can also be configured on transactions, in which case the transaction configuration is used instead of this default.
shutdownTimeout	integer	no	5000	(As of Mule 2.2.2) The time in milliseconds to wait for any in-progress messages to finish processing before Mule shuts down. After this threshold has been reached, Mule starts interrupting threads, and messages can be lost. If you have a very large number of services in the same Mule instance, if you have components that take more than a couple seconds to process, or if you are using large payloads and/or slower transports, you should increase this value to allow more time for graceful shutdown. The value you specify is applied to services and separately to dispatchers, so the default value of 5000 milliseconds specifies that Mule has ten seconds to process and dispatch messages gracefully after shutdown is initiated.

#### Child Elements of <configuration...>

Name	Cardinality	Description
default-threading-profile	0..1	The default threading profile, used by components and by endpoints for dispatching and receiving if no more specific configuration is given.
default-dispatcher-threading-profile	0..1	The default dispatching threading profile, which modifies the default-threading-profile values and is used by endpoints for dispatching messages. This can also be configured on connectors, in which case the connector configuration is used instead of this default.
default-receiver-threading-profile	0..1	The default receiving threading profile, which modifies the default-threading-profile values and is used by endpoints for receiving messages. This can also be configured on connectors, in which case the connector configuration is used instead of this default.
default-service-threading-profile	0..1	The default service threading profile, which modifies the default-threading-profile and is used by services for processing messages. This can also be configured on models or services, in which case these configurations will be used instead of this default.
abstract-reconnection-strategy	0..1	The default reconnection strategy, used by connectors and endpoints. This can also be configured on connectors, in which case the connector configuration is used instead of this default. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.

Your Rating: Results:  0 rates

## Inbound Router Configuration Reference

### Inbound Router Configuration Reference

This page provides details on the elements you configure for [inbound routers](#). This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page.

cache: Unexpected program error: java.lang.NullPointerException

### **Idempotent receiver router**

Ensures that only unique messages are received by a service by checking the unique ID of the incoming message. Note that the ID used can be generated from the message using an expression defined in the 'idExpression' attribute. By default, the expression used is '#[message:id]', which means the underlying endpoint must support unique message IDs for this to work. Otherwise, a `UniquelIdNotSupportedException` is thrown.

#### **Attributes of <idempotent-receiver-router...>**

Name	Type	Required	Default	Description
idExpression	string	no		Defines one or more expressions to use when extracting the ID from the message. For example, it would be possible to combine two headers as the ID of the message to provide idempotency: '#[headers:foo,bar]'. Or, you could combine the message ID with a header: '#[message:id]-#[header:foo]'. If this property is not set, '#[message:id]' will be used by default.

#### **Child Elements of <idempotent-receiver-router...>**

Name	Cardinality	Description
abstract-object-store	0..1	A placeholder for an object store that can be used by routers to maintain state.

cache: Unexpected program error: java.lang.NullPointerException

### **Idempotent secure hash receiver router**

Ensures that only unique messages are received by a service by calculating the hash of the message itself using a message digest algorithm. This provides a value with an infinitesimally small chance of a collision. This can be used to filter message duplicates. Keep in mind that the hash is calculated over the entire byte array representing the message, so any leading or trailing spaces or extraneous bytes (like padding) can produce different hash values for the same semantic message content. Care should be taken to ensure that messages do not contain extraneous bytes. This class is useful when the message does not support unique identifiers.

#### **Attributes of <idempotent-secure-hash-receiver-router...>**

Name	Type	Required	Default	Description
messageDigestAlgorithm	string	no		The secure hashing algorithm to use. If not set, the default is SHA-256.

#### **Child Elements of <idempotent-secure-hash-receiver-router...>**

Name	Cardinality	Description
abstract-object-store	0..1	A placeholder for an object store that can be used by routers to maintain state.

cache: Unexpected program error: java.lang.NullPointerException

### **Wire tap router**

The WireTap inbound router allows you to route certain messages to a different endpoint as well as to the component.

#### **Attributes of <wire-tap-router...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### **Child Elements of <wire-tap-router...>**

Name	Cardinality	Description
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-outbound-endpoint	1..1	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.

cache: Unexpected program error: java.lang.NullPointerException

## Forwarding router

Allows messages to be forwarded to the outbound routers without first being processed by a component.

### Attributes of <forwarding-router...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <forwarding-router...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Selective consumer router

Applies one or more filters to the incoming message. If the filters match, the message is forwarded to the component. Otherwise, the message is forwarded to the catch-all strategy on the router. If no catch-all strategy is configured, the message is ignored and a warning is logged.

### Attributes of <selective-consumer-router...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <selective-consumer-router...>

Name	Cardinality	Description
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.

cache: Unexpected program error: java.lang.NullPointerException

## Correlation resequencer router

Holds back a group of messages and resequences them using each message's correlation sequence property.

### Attributes of <correlation-resequencer-router...>

Name	Type	Required	Default	Description
timeout	integer	no		Defines a timeout in Milliseconds to wait for events to be aggregated. By default the router will throw an exception if the router is waiting for a correlation group and times out before all group entries are received.
failOnTimeout	boolean	no		When false, incomplete aggregation groups will be forwarded to a component on timeout as a java.util.List. When true (default), a CorrelationTimeoutException is thrown and RoutingNotification.CORRELATION_TIMEOUT is fired. The component doesn't receive any messages in this case.

### Child Elements of <correlation-resequencer-router...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Message chunking aggregator router

Combines two or more messages into a single message by matching messages with a given Correlation ID. Correlation IDs are set on messages when they are dispatched by certain outbound routers, such as the Recipient List and Message Splitter routers. These messages can be aggregated back together again using this router.

### Attributes of <message-chunking-aggregator-router...>

Name	Type	Required	Default	Description
timeout	integer	no		Defines a timeout in Milliseconds to wait for events to be aggregated. By default the router will throw an exception if the router is waiting for a correlation group and times out before all group enties are received.
failOnTimeout	boolean	no		When false, incomplete aggregation groups will be forwarded to a component on timeout as a java.util.List. When true (default), a CorrelationTimeoutException is thrown and RoutingNotification.CORRELATION_TIMEOUT is fired. The component doesn't receive any messages in this case.

#### Child Elements of <message-chunking-aggregator-router...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Custom correlation aggregator router

Configures a custom message aggregator. Mule provides an abstract implementation that has a template method that performs the message aggregation. A common use of the aggregator router is to combine the results of multiple requests such as "ask this set of vendors for the best price of X".

#### Attributes of <custom-correlation-aggregator-router...>

Name	Type	Required	Default	Description
timeout	integer	no		Defines a timeout in Milliseconds to wait for events to be aggregated. By default the router will throw an exception if the router is waiting for a correlation group and times out before all group enties are received.
failOnTimeout	boolean	no		When false, incomplete aggregation groups will be forwarded to a component on timeout as a java.util.List. When true (default), a CorrelationTimeoutException is thrown and RoutingNotification.CORRELATION_TIMEOUT is fired. The component doesn't receive any messages in this case.
class	class name	yes		Fully qualified class name of the custom correlation aggregator router to be used.

#### Child Elements of <custom-correlation-aggregator-router...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

#### Collection aggregator router

Configures a Collection Response Router. This will return a MuleMessageCollection message type that will contain all messages received for a each correlation group.

#### Attributes of <collection-aggregator-router...>

Name	Type	Required	Default	Description
timeout	integer	no		Defines a timeout in Milliseconds to wait for events to be aggregated. By default the router will throw an exception if the router is waiting for a correlation group and times out before all group enties are received.
failOnTimeout	boolean	no		When false, incomplete aggregation groups will be forwarded to a component on timeout as a java.util.List. When true (default), a CorrelationTimeoutException is thrown and RoutingNotification.CORRELATION_TIMEOUT is fired. The component doesn't receive any messages in this case.

#### Child Elements of <collection-aggregator-router...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Custom inbound router

Allows for custom inbound routers to be configured.

#### Attributes of <custom-inbound-router...>

Name	Type	Required	Default	Description
class	class name	yes		An implementation of InboundRouter (fully qualified Java class name)

#### Child Elements of <custom-inbound-router...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property elements so that custom configuration can be configured on the custom router.

Your Rating: 

Results:  0 rates

## Model Configuration Reference

### Model Configuration Reference

This page provides details on the elements you configure for models. Some of this information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on models, see [Models](#).

[Schema Reference documentation on the Model element.](#)

cache: Unexpected program error: java.lang.NullPointerException

#### Model

The container for a set of services, providing basic settings and processing for all the services it contains.

#### Attributes of <model...>

Name	Type	Required	Default	Description
name	name	no		The name used to identify this model.
inherit	boolean	no		If true, this model element is an extension of a previous model element with the same name.

#### Child Elements of <model...>

Name	Cardinality	Description
abstract-exception-strategy	0..1	A placeholder for an exception strategy element. Exception strategies define how Mule should react to errors.
abstract-service	0..*	A placeholder for a service element. Services combine message routing with a component (typically a POJO).
abstract-entry-point-resolver-set	0..1	A placeholder for entry point resolver set elements. These combine a group of entry point resolvers, trying them in turn until one succeeds.
abstract-entry-point-resolver	0..1	A placeholder for an entry point resolver element. Entry point resolvers define how payloads are delivered to Java code by choosing the method to call.
abstract-queue-profile	0..1	A placeholder for a queue profile, which controls how messages are queued.

cache: Unexpected program error: java.lang.NullPointerException

### Queue profile

Specifies the properties of an internal Mule queue. Internal queues are used to queue messages for each component managed by Mule.

#### Attributes of <queue-profile...>

Name	Type	Required	Default	Description
maxOutstandingMessages	integer	no		Defines the maximum number of messages that can be queued.
persistent	boolean	no	false	Whether Mule messages are persisted to a store. Primarily, this is used for persisting queued messages to disk so that the internal state of the server is mirrored on disk in case the server fails and needs to be restarted. Default is false.

#### Child Elements of <queue-profile...>

Name	Cardinality	Description
------	-------------	-------------

#### Exception Strategy

See [Exception Strategy Configuration Reference](#).

#### Service

See [Service Configuration Reference](#).

#### Entry Point Resolver

See [Entry Point Resolver Configuration Reference](#).

Your Rating:

Results: ★★★★★ 1 rates

## Notifications Configuration Reference

### Notifications Configuration Reference

This page provides details on the elements you configure for notifications. Some of this information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on notifications, see [Mule Server Notifications](#).

cache: Unexpected program error: java.lang.NullPointerException

#### Notifications

Registers listeners for notifications and associates interfaces with particular events.

#### Attributes of <notifications...>

Name	Type	Required	Default	Description
dynamic	boolean	no		If the notification manager is dynamic, listeners can be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. Otherwise, some parts of Mule will cache notification configuration for efficiency and will not generate events for newly enabled notifications or listeners. The default value is false.

#### Child Elements of <notifications...>

Name	Cardinality	Description
notification	0..*	Associates an event with an interface. Listeners that implement the interface will receive instances of the event.
disable-notification	0..*	Blocks the association of an event with a particular interface. This filters events after the association with a particular interface (and so takes precedence).

notification-listener	0..*	Registers a bean as a listener with the notification system. Events are dispatched by reflection - the listener will receive all events associated with any interfaces it implements. The relationship between interfaces and events is configured by the notification and disable-notification elements.
-----------------------	------	---

cache: Unexpected program error: java.lang.NullPointerException

## Notification

Associates an event with an interface. Listeners that implement the interface will receive instances of the event.

### Attributes of <notification...>

Name	Type	Required	Default	Description
event-class	class name	no		The class associated with a notification event that will be delivered to the interface. This can be used instead of the 'event' attribute to specify a custom class.
event	notificationTypes	no		The notification event to deliver.
interface-class	class name	no		The interface (class name) that will receive the notification event.
interface	notificationTypes	no		The interface that will receive the notification event.

### Child Elements of <notification...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Disable notification

Blocks the association of an event with a particular interface. This filters events after the association with a particular interface (and so takes precedence).

### Attributes of <disable-notification...>

Name	Type	Required	Default	Description
event-class	class name	no		The class associated with an event that will no longer be delivered to any interface. This can be used instead of the 'event' attribute to specify a custom class.
event	notificationTypes	no		The event you no longer want to deliver.
interface-class	class name	no		The interface (class name) that will no longer receive the event.
interface	notificationTypes	no		The interface that will no longer receive the event.

### Child Elements of <disable-notification...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Notification listener

Registers a bean as a listener with the notification system. Events are dispatched by reflection - the listener will receive all events associated with any interfaces it implements. The relationship between interfaces and events is configured by the notification and disable-notification elements.

## Notification Types

You can specify the following types of notifications using the event attribute of the <notification> and <disable-notification> element:

- "CONTEXT"
- "MODEL"
- "SERVICE"
- "SECURITY"
- "ENDPOINT-MESSAGE"
- "COMPONENT-MESSAGE"

"MANAGEMENT"  
"CONNECTION"  
"REGISTRY"  
"CUSTOM"  
"EXCEPTION"  
"TRANSACTION"  
"ROUTING"

Your Rating:  0 rates

Results:  0 rates

## Outbound Router Configuration Reference

### Outbound Router Configuration Reference

This page provides details on the elements you configure for **outbound routers**. This information is pulled directly from the schema files and is cached. If the information appears to be out of date, refresh the page.

cache: Unexpected program error: java.lang.NullPointerException

#### Pass through router

This router always matches and simply sends or dispatches message via the endpoint that is configured.

##### Attributes of <pass-through-router...>

Name	Type	Required	Default	Description
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.

##### Child Elements of <pass-through-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..1	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

#### Filtering router

Uses filters to determine whether the message matches a particular criteria and if so will route the message to the endpoint configured on the router.

##### Attributes of <filtering-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.

enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
-------------------	-------------------------	----	------------	--

#### Child Elements of <filtering-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..1	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException  
cache: Unexpected program error: java.lang.NullPointerException

#### Chaining router

Sends the message through multiple endpoints using the result of the first invocation as the input for the next.

#### Attributes of <chaining-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.

#### Child Elements of <chaining-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.

abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
----------------------	------	---

cache: Unexpected program error: java.lang.NullPointerException

### Exception based router

Sends a message over an endpoint by selecting the first endpoint that can connect to the transport. Endpoints are listed statically in the router configuration.

#### Attributes of <exception-based-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.

#### Child Elements of <exception-based-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

### Multicasting router

Sends the same message over multiple endpoints.

#### Attributes of <multicasting-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.

enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
-------------------	-------------------------	----	------------	--

#### Child Elements of <multicasting-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

#### Endpoint selector router

Selects the outgoing endpoint based on an expression evaluator ("header:endpoint" by default). It will first try to match the endpoint by name and then by address. The endpoints to use can be set on the router itself or be global endpoint definitions.

#### Attributes of <endpoint-selector-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
default	string	no		The name of the default endpoint to use if the expression returns null. This can be used as an 'else' condition to route messages that don't contain the expected routing information.
evaluator	standardExpressionEvaluators	yes		The expression evaluator to use. Expression evaluators must be registered with the ExpressionEvaluatorManager before they can be used. Using the custom evaluator allows you to define your own evaluator with the 'custom-evaluator' attribute. Note that some evaluators such as xpath, groovy, and bean are loaded from other Mule modules (XML and Scripting, respectively). These modules must be on your classpath before the evaluator can be used.
expression	string	yes		The expression to evaluate. The syntax of this attribute changes depending on the evaluator being used.

custom-evaluator	name (no spaces)	no		The name of the custom evaluator to use. This attribute is only used when the 'evaluator' attribute is set to "custom". You can plug in your own expression evaluators by registering them with the ExpressionEvaluatorManager.
------------------	------------------	----	--	---

#### Child Elements of <endpoint-selector-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

#### List message splitter router

The Filtering List Message Splitter accepts a list of objects that is split each object being routed to different endpoints.

#### Attributes of <list-message-splitter-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
deterministic	boolean	no		If 'disableRoundRobin' is false and this option is true (the default) then the first message part will be routed to the first endpoint, the second part to the second endpoint, etc, with the nth part going to the (n modulo number of endpoints) endpoint. If false then the messages will be distributed equally amongst all endpoints.
disableRoundRobin	boolean	no		If filters are being used on endpoints then round robin behaviour is probably not desirable. This flag switches round robin behaviour off, it is on by default.
failIfNoMatch	boolean	no		If 'disableRoundRobin' is true, there may be situations where the current split message does not match any endpoints. this flag controls whether an exception should be thrown when a match is not found.

#### Child Elements of <list-message-splitter-router...>

Name	Cardinality	Description
------	-------------	-------------

abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

## Expression splitter router

Splits the message based on an expression. The expression must return one or more message parts in order to be effective.

### Attributes of <expression-splitter-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
deterministic	boolean	no		If 'disableRoundRobin' is false and this option is true (the default) then the first message part will be routed to the first endpoint, the second part to the second endpoint, etc, with the nth part going to the (n modulo number of endpoints) endpoint. If false then the messages will be distributed equally amongst all endpoints.
disableRoundRobin	boolean	no		If filters are being used on endpoints then round robin behaviour is probably not desirable. This flag switches round robin behaviour off, it is on by default.
failIfNoMatch	boolean	no		If 'disableRoundRobin' is true, there may be situations where the current split message does not match any endpoints. this flag controls whether an exception should be thrown when a match is not found.
evaluator	standardExpressionEvaluators	yes		The expression evaluator to use. Expression evaluators must be registered with the ExpressionEvaluatorManager before they can be used. Using the custom evaluator allows you to define your own evaluator with the 'custom-evaluator' attribute. Note that some evaluators such as xpath, groovy, and bean are loaded from other Mule modules (XML and Scripting, respectively). These modules must be on your classpath before the evaluator can be used.
expression	string	yes		The expression to evaluate. The syntax of this attribute changes depending on the evaluator being used.

custom-evaluator	name (no spaces)	no		The name of the custom evaluator to use. This attribute is only used when the 'evaluator' attribute is set to "custom". You can plug in your own expression evaluators by registering them with the ExpressionEvaluatorManager.
------------------	------------------	----	--	---

#### Child Elements of <expression-splitter-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

#### Filter based splitter

The filter-based splitter router will select the endpoint where you want to send part of a message by filtering parts using the endpoint filters.

#### Attributes of <filter-based-splitter...>

Name	Type	Required	Default	Description
splitExpression	string	yes		The XPath expression used to split the message.
externalSchemaLocation	string	no		The location of a schema that should be used to validate the current message. This is not required if the message contains the location of the schema.
validateSchema	boolean	no		Whether to enable schema validation when processing the XML message. Note that this can have a serious performance hit on high-throughput systems.
failIfNoMatch	boolean	no		Whether this router should fail if none of the endpoint filters match the payload. The default is true.

#### Child Elements of <filter-based-splitter...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as prefix and uri attributes. The prefix can then be used inside the expression.

cache: Unexpected program error: java.lang.NullPointerException

#### Round robin splitter

The round robin message splitter will split a DOM4J document into nodes based on the `splitExpression` property. It will then send these document fragments to the list of specified endpoints in a round-robin fashion. Optionally, you can specify a namespaces property map that contain prefix/namespace mappings.

#### Attributes of <round-robin-splitter...>

Name	Type	Required	Default	Description

splitExpression	string	yes		The XPath expression used to split the message.
externalSchemaLocation	string	no		The location of a schema that should be used to validate the current message. This is not required if the message contains the location of the schema.
validateSchema	boolean	no		Whether to enable schema validation when processing the XML message. Note that this can have a serious performance hit on high-throughput systems.
deterministic	boolean	no		If there is no endpoint filtering and this attribute is true (the default), the first message part is routed to the first endpoint, the second part routes to the second endpoint, and so on with the nth part going to the (n modulo number of endpoints) endpoint. If false, the messages will be distributed equally among all endpoints.

#### Child Elements of <round-robin-splitter...>

Name	Cardinality	Description
namespace	0..*	A namespace declaration, expressed as <code>prefix</code> and <code>uri</code> attributes. The prefix can then be used inside the expression.

cache: Unexpected program error: java.lang.NullPointerException

#### Message chunking router

Allows you to split a single message into a number of fixed-length messages that will all be routed to the same endpoint.

#### Attributes of <message-chunking-router...>

Name	Type	Required	Default	Description
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
messageSize	integer	no		The message chunk size (in bytes) that the current message will be split into. Note that this is mutually exclusive to the 'numberOfMessages' property.
numberOfMessages	integer	no		The number of message pieces to break the current message into. This property is less useful than the 'message' size property since, usually messages are constricted by size. Note that this is mutually exclusive to the 'messageSize' property.

#### Child Elements of <message-chunking-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.

abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
----------------------	------	---

cache: Unexpected program error: java.lang.NullPointerException

### Static recipient list router

Sends the same message to multiple endpoints over the same endpoint, or implements routing-slip behavior where the next destination for the message is determined from message properties or the payload. It uses a static list of recipient endpoints.

#### Attributes of <static-recipient-list-router...>

Name	Type	Required	Default	Description
recipientsProperty	string	no		Defines a property name on the current message where a list of endpoint names (or URLs) can be obtained. This property can return a <code>java.util.List</code> of values or a delimited <code>java.lang.String</code> . If the 'recipientsProperty' returns a string then the 'recipientsDelimiter' property is used to split the string. If the entries in the String or List define endpoint names, these will be looked up at runtime. If the entries define endpoint URLs these endpoints will be created at runtime.
recipientsDelimiter	string	no		The delimiter to use when splitting a String list of recipients. the default is ','. This property is only used with the 'recipientsProperty'.
synchronous	boolean	no		This flag controls whether the message will be sent to the recipients synchronously. Unlike other routers the recipient list router doesn't have pre-configured endpoints so the synchronicity of the endpoint cannot be honoured.
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.

#### Child Elements of <static-recipient-list-router...>

Name	Cardinality	Description
recipients	0..1	Static list of recipients that the outgoing message is sent to. The default delimiter is ','.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

cache: Unexpected program error: java.lang.NullPointerException

## Expression recipient list router

Sends the same message to multiple endpoints over the same endpoint, or implements routing-slip behavior where the next destination for the message is determined from message properties or the payload. The recipients can be extracted from the message using an expression, or you can specify a static list of recipient endpoints. (As of version 2.1)

### Attributes of <expression-recipient-list-router...>

Name	Type	Required	Default	Description
synchronous	boolean	no		This flag controls whether the message will be sent to the recipients synchronously. Unlike other routers the recipient list router doesn't have pre-configured endpoints so the synchronicity of the endpoint cannot be honoured.
useTemplates	boolean	no	true	Determines if placeholders with expressions can be used with the form [ ] in endpoint uri's.
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.
evaluator	standardExpressionEvaluators	yes		The expression evaluator to use. Expression evaluators must be registered with the ExpressionEvaluatorManager before they can be used. Using the custom evaluator allows you to define your own evaluator with the 'custom-evaluator' attribute. Note that some evaluators such as xpath, groovy, and bean are loaded from other Mule modules (XML and Scripting, respectively). These modules must be on your classpath before the evaluator can be used.
expression	string	yes		The expression to evaluate. The syntax of this attribute changes depending on the evaluator being used.
custom-evaluator	name (no spaces)	no		The name of the custom evaluator to use. This attribute is only used when the 'evaluator' attribute is set to "custom". You can plug in your own expression evaluators by registering them with the ExpressionEvaluatorManager.

### Child Elements of <expression-recipient-list-router...>

Name	Cardinality	Description
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
recipients	0..1	A static list of endpoint names or URIs that will be used as recipients of the current message. If the expression on this router returns a list of endpoint names, the endpoints here will be checked as well as any global endpoints.

cache: Unexpected program error: java.lang.NullPointerException

## Custom outbound router

Allows you to configure a custom outbound router by specifying the custom router class and by using Spring properties.

### Attributes of <custom-outbound-router...>

Name	Type	Required	Default	Description
class	class name	yes		An implementation of OutboundRouter (fully qualified Java class name)
transformer-refs	list of names	no		A list of the transformers that will be applied to the message in order before it is delivered to the component.
enableCorrelation	ALWAYS/NEVER/IF_NOT_SET	no	IF_NOT_SET	Specifies whether Mule should give outgoing messages a correlation ID. The default behavior is to give messages a correlation ID only if they don't already have one, so that existing correlation IDs are maintained.

### Child Elements of <custom-outbound-router...>

Name	Cardinality	Description
abstract-outbound-endpoint	0..*	A placeholder for outbound endpoint elements. Outbound endpoints dispatch messages to the underlying transport.
spring:property	0..*	Spring-style property elements so that custom configuration can be configured on the custom router.
abstract-filter	0..1	Filters the messages to be processed by this router. @Deprecated since 2.2. Configure the filter on the endpoint instead of the router. A placeholder for filter elements, which control which messages are handled.
abstract-transformer	0..*	Filters are applied before message transformations. A transformer can be configured here to transform messages before they are filtered. A placeholder for transformer elements. Transformers convert message payloads.
reply-to	0..1	Defines where the message should be routed after the recipient of the message to which this service dispatches has finished with it.
abstract-transaction	0..1	Defines an overall transaction that will be used for all endpoints on this router. This is only useful when you want to define an outbound only transaction that will commit all of the transactions defined on the outbound endpoints for this router. Note that you must still define a transaction on each of the endpoints that should take part in the transaction. These transactions should always be configured to JOIN the existing transaction. A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

Your Rating: 

Results:  1 rates

## Properties Configuration Reference

### Properties Configuration Reference

This page provides detailed reference information for property elements in Mule. The information below is pulled directly from the source code, so it reflects the latest data since you loaded the page. If something appears to be out of date, you can refresh the page to reload the information.

cache: Unexpected program error: java.lang.NullPointerException

#### Global property

A global property is a named string. It can be inserted in most attribute values using standard (ant-style) Spring placeholders.

### Attributes of <global-property...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		The name of the property. This is used inside Spring placeholders.

value	string	yes		The value of the property. This replaces each occurrence of a Spring placeholder.
-------	--------	-----	--	---

#### ***Child Elements of <global-property...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Property**

Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.

#### ***Attributes of <property...>***

Name	Type	Required	Default	Description
key	string	no		
value	string	no		
value-ref	string	no		

#### ***Child Elements of <property...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Properties**

A map of Mule properties.

#### ***Attributes of <properties...>***

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### ***Child Elements of <properties...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Jndi provider property**

Direct setting of a JNDI property.

#### ***Attributes of <jndi-provider-property...>***

Name	Type	Required	Default	Description
key	string	no		
value	string	no		
value-ref	string	no		

#### ***Child Elements of <jndi-provider-property...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Jndi provider properties

Direct setting of JNDI properties (allows access to the full Spring map entry).

### Attributes of <jndi-provider-properties...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <jndi-provider-properties...>

Name	Cardinality	Description
------	-------------	-------------

Your Rating:  Results:  2 rates

## Security Manager Configuration Reference

### Security Manager Configuration Reference

This page provides details on the elements you configure for the security manager. For more information, see [Configuring Security](#).

cache: Unexpected program error: java.lang.NullPointerException

#### Security manager

The default security manager.

### Attributes of <security-manager...>

Name	Type	Required	Default	Description
id		no		
name		no		

### Child Elements of <security-manager...>

Name	Cardinality	Description
custom-security-provider	0..1	A custom implementation of SecurityProvider.
custom-encryption-strategy	0..1	A custom implementation of EncryptionStrategy.
secret-key-encryption-strategy	0..1	Provides secret key-based encryption using JCE.
password-encryption-strategy	0..1	Provides password-based encryption using JCE. Users must specify a password and optionally a salt and iteration count as well. The default algorithm is PBEWithMD5AndDES, but users can specify any valid algorithm supported by JCE.

cache: Unexpected program error: java.lang.NullPointerException

#### Custom security provider

A custom implementation of SecurityProvider.

### Attributes of <custom-security-provider...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		
provider-ref	string	yes		The name of the security provider to use.

### Child Elements of <custom-security-provider...>

Name	Cardinality	Description
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### Custom encryption strategy

A custom implementation of EncryptionStrategy.

#### Attributes of <custom-encryption-strategy...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		
strategy-ref	string	yes		A reference to the encryption strategy (which may be a Spring bean that implements the EncryptionStrategy interface).

#### Child Elements of <custom-encryption-strategy...>

Name	Cardinality	Description
spring:property	0..*	

cache: Unexpected program error: java.lang.NullPointerException

### Secret key encryption strategy

Provides secret key-based encryption using JCE.

#### Attributes of <secret-key-encryption-strategy...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		
key	string	no		The key to use. This and the 'keyFactory-ref' attribute are mutually exclusive.
keyFactory-ref	string	no		The name of the key factory to use. This should implement the ObjectFactory interface and return a byte array. This and the 'key' attribute are mutually exclusive.

#### Child Elements of <secret-key-encryption-strategy...>

Name	Cardinality	Description
spring:property	0..*	

cache: Unexpected program error: java.lang.NullPointerException

### Password encryption strategy

Provides password-based encryption using JCE. Users must specify a password and optionally a salt and iteration count as well. The default algorithm is PBKDF2WithHmacSHA1, but users can specify any valid algorithm supported by JCE.

#### Attributes of <password-encryption-strategy...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		
password	string	yes		The password to use.
salt	string	no		The salt to use (this helps prevent dictionary attacks).
iterationCount	integer	no		The number of iterations to use.

## **Child Elements of <password-encryption-strategy...>**

Name	Cardinality	Description
------	-------------	-------------

Your Rating: Results:  0 rates

# Service Configuration Reference

## Service Configuration Reference

This page provides details on the elements you configure for services. This information is pulled directly from `mule.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on services, see [Configuring the Service](#).

cache: Unexpected program error: java.lang.NullPointerException

### Service

Describes how to receive messages, deliver them to a component, and handle the results (if any).

#### Attributes of <service...>

Name	Type	Required	Default	Description
name	name	yes		The name used to identify this service.
initialState	started/stopped/paused	no	started	The initial state of the service. Usually a service is started automatically ("started"), but this attribute can be used to disable initial startup ("stopped") or start the service in a paused state ("paused").
queueTimeout	integer	no		The timeout used when taking messages from the service queue.

#### Child Elements of <service...>

Name	Cardinality	Description
description	0..1	This can hold any kind of documentation related to the service. It is intended to be "human readable" only and is not used by the system.
inbound	0..1	The elements within 'inbound' describe how a service receives messages.
abstract-component	0..1	The service component that is invoked when incoming messages are received. If this element is not present, the service simply bridges the inbound and outbound using a pass-through component. A placeholder for a component element. A component is invoked when inbound messages are received by the service.
outbound	0..1	The elements within 'outbound' describe how a service sends or dispatches messages.
async-reply	0..1	The elements within 'async-reply' describe how asynchronous replies are handled.
abstract-exception-strategy	0..1	A placeholder for an exception strategy element. Exception strategies define how Mule should react to errors.
abstract-message-info-mapping	0..1	The message info mapper used to extract key bits of the message information, such as Message ID or Correlation ID. These properties are used by some routers and this mapping information tells Mule where to get the information from in the current message. Maps the attributes of the current message to known message elements in Mule, namely Message ID and CorrelationID.
abstract-service-threading-profile	0..1	A placeholder for the service threading profile element. Threading profiles define how thread pools are used by a service.
abstract-queue-profile	0..1	A placeholder for a queue profile, which controls how messages are queued.

cache: Unexpected program error: java.lang.NullPointerException

### Custom service

A user-implemented service (typically used only in testing).

### Attributes of <custom-service...>

Name	Type	Required	Default	Description
name	name	yes		The name used to identify this service.
initialState	started/stopped/paused	no	started	The initial state of the service. Usually a service is started automatically ("started"), but this attribute can be used to disable initial startup ("stopped") or start the service in a paused state ("paused").
class	class name	yes		The class to use for the service.

### Child Elements of <custom-service...>

Name	Cardinality	Description
description	0..1	This can hold any kind of documentation related to the service. It is intended to be "human readable" only and is not used by the system.
inbound	0..1	The elements within 'inbound' describe how a service receives messages.
abstract-component	0..1	The service component that is invoked when incoming messages are received. If this element is not present, the service simply bridges the inbound and outbound using a pass-through component. A placeholder for a component element. A component is invoked when inbound messages are received by the service.
outbound	0..1	The elements within 'outbound' describe how a service sends or dispatches messages.
async-reply	0..1	The elements within 'async-reply' describe how asynchronous replies are handled.
abstract-exception-strategy	0..1	A placeholder for an exception strategy element. Exception strategies define how Mule should react to errors.
abstract-message-info-mapping	0..1	The message info mapper used to extract key bits of the message information, such as Message ID or Correlation ID. These properties are used by some routers and this mapping information tells Mule where to get the information from in the current message. Maps the attributes of the current message to known message elements in Mule, namely Message ID and CorrelationID.
spring:property	0..*	Spring-style property element for custom configuration.

cache: Unexpected program error: java.lang.NullPointerException

### Inbound

The elements within 'inbound' describe how a service receives messages.

### Attributes of <inbound...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <inbound...>

Name	Cardinality	Description
abstract-inbound-endpoint	0..*	A placeholder for inbound endpoint elements. Inbound endpoints receive messages from the underlying transport. The message payload is then delivered to the component for processing.
abstract-catch-all-strategy	0..1	A placeholder for catch-all strategy elements.
abstract-inbound-router	0..1	A placeholder for inbound router elements, which control how incoming messages are handled.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
splitter	0..1	The simple splitter splits incoming message into parts using the configured expression passing on each part in turn to the next message processor
collection-splitter	0..1	The collection splitter accepts a collection of objects and splits the collection invoking the next message processor with each item in the collection in sequence.
processor	0..1	A reference to a message processor defined elsewhere.

custom-processor	0..1	
------------------	------	--

cache: Unexpected program error: java.lang.NullPointerException

## Outbound

The elements within 'outbound' describe how a service sends or dispatches messages.

### Attributes of <outbound...>

Name	Type	Required	Default	Description
matchAll	boolean	no	false	If true, the output message will be sent to all routers. Otherwise, only the first matching router is used.

### Child Elements of <outbound...>

Name	Cardinality	Description
abstract-outbound-router	0..*	A placeholder for outbound router elements, which control how outgoing messages are delivered to the outbound endpoints.
abstract-catch-all-strategy	0..1	A placeholder for catch-all strategy elements.

cache: Unexpected program error: java.lang.NullPointerException

## Async reply

The elements within 'async-reply' describe how asynchronous replies are handled.

### Attributes of <async-reply...>

Name	Type	Required	Default	Description
timeout	integer	no		The timeout (ms) to wait for a reply.
failOnTimeout	boolean	no		If the router times out before all expected events have been received, specifies whether an exception should be thrown (true) or the current events should be returned for processing (false). The default is true.

### Child Elements of <async-reply...>

Name	Cardinality	Description
abstract-inbound-endpoint	1..*	A placeholder for inbound endpoint elements. Inbound endpoints receive messages from the underlying transport. The message payload is then delivered to the component for processing.
abstract-async-reply-router	0..1	A placeholder for an async reply router element. Asynchronous replies are handled via this router.
abstract-inbound-router	0..1	A placeholder for inbound router elements, which control how incoming messages are handled.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	

cache: Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

## Queue profile

Specifies the properties of an internal Mule queue. Internal queues are used to queue messages for each component managed by Mule.

### Attributes of <queue-profile...>

Name	Type	Required	Default	Description
maxOutstandingMessages	integer	no		Defines the maximum number of messages that can be queued.

<code>persistent</code>	boolean	no	false	Whether Mule messages are persisted to a store. Primarily, this is used for persisting queued messages to disk so that the internal state of the server is mirrored on disk in case the server fails and needs to be restarted. Default is false.
-------------------------	---------	----	-------	---

#### ***Child Elements of <queue-profile...>***

Name	Cardinality	Description
------	-------------	-------------

#### **Exception Strategy**

See [Exception Strategy Configuration Reference](#).

#### **Catch All Strategy**

See [Catch-all Strategy Configuration Reference](#).

#### **Component**

See [Component Configuration Reference](#).

Your Rating: Results: 0 rates

## **Transactions Configuration Reference**

### **Transactions Configuration Reference**

This page provides reference information on the elements you configure for transactions. For more information on transactions, see [Transaction Management](#).

cache: Unexpected program error: java.lang.NullPointerException

#### **Abstract transaction**

A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

#### ***Attributes of <abstract-transaction...>***

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

action	NONE/ALWAYS_BEGIN/BEGIN_OR_JOIN/ALWAYS_JOIN/JOIN_IF_POSSIBLE	yes		The type of action the transaction should take, one of the following: NONE - Never participate in a transaction. ALWAYS_BEGIN - Always start a new transaction when receiving a message. An exception will be thrown if a transaction already exists. BEGIN_OR_JOIN - If a transaction is already in progress when a message is received, join the transaction if possible. Otherwise, start a new transaction. ALWAYS_JOIN - Always expects a transaction to be in progress when a message is received. If there is no transaction, an exception is thrown. JOIN_IF_POSSIBLE - Join the current transaction if one is available. Otherwise, no transaction is created.
timeout	integer	no		Timeout for the transaction (ms).

#### ***Child Elements of <abstract-transaction...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Custom transaction**

A user-defined or otherwise unsupported third-party transactions.

#### ***Attributes of <custom-transaction...>***

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

action	NONE/ALWAYS_BEGIN/BEGIN_OR_JOIN/ALWAYS_JOIN/JOIN_IF_POSSIBLE	yes		The type of action the transaction should take, one of the following: NONE - Never participate in a transaction. ALWAYS_BEGIN - Always start a new transaction when receiving a message. An exception will be thrown if a transaction already exists. BEGIN_OR_JOIN - If a transaction is already in progress when a message is received, join the transaction if possible. Otherwise, start a new transaction. ALWAYS_JOIN - Always expects a transaction to be in progress when a message is received. If there is no transaction, an exception is thrown. JOIN_IF_POSSIBLE - Join the current transaction if one is available. Otherwise, no transaction is created.
timeout	integer	no		Timeout for the transaction (ms).
factory-class	class name	no		A class that implements the TransactionFactory interface that will be instantiated and used to generate a transaction. This attribute and the 'factory-ref' attribute are mutually exclusive; one of the two is required.
factory-ref	string	no		A bean that implements the TransactionFactory interface that will be used to generate a transaction. This attribute and the 'factory-class' attribute are mutually exclusive; one of the two is required.

#### ***Child Elements of <custom-transaction...>***

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Xa transaction**

An XA transaction.

### Attributes of <xa-transaction...>

Name	Type	Required	Default	Description
action	NONE/ALWAYS_BEGIN/BEGIN_OR_JOIN/ALWAYS_JOIN/JOIN_IF_POSSIBLE	yes		The type of action the transaction should take, one of the following: NC - Never participate in a transaction. ALWAYS_BEGIN Always start a new transaction when receiving a message. An exception will be thrown if a transaction already exists. BEGIN_OR_JOIN If a transaction is already in progress when a message is received, join the transaction if possible. Otherwise start a new transaction. ALWAYS_JOIN Always expects transaction to be in progress when a message is received. If there is no transaction, an exception is thrown. JOIN_IF_POSSIBLE - Join the current transaction if one is available. Otherwise, no transaction is created.
timeout	integer	no		Timeout for the transaction (ms)
interactWithExternal	boolean	no		If this is set to "true", Mule interacts with transactions begun outside of Mule. If an external transaction is active, then BEGIN_OR_JOIN will join it, and ALWAYS_BEGIN will cause an exception to be thrown.

### Child Elements of <xa-transaction...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Websphere transaction manager

The WebSphere transaction manager.

#### **Attributes of <websphere-transaction-manager...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no	transactionManager	An optional name for the transaction manager. The default value is "transactionManager".

#### **Child Elements of <websphere-transaction-manager...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Jboss transaction manager**

The JBoss transaction manager.

#### **Attributes of <jboss-transaction-manager...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no	transactionManager	An optional name for the transaction manager. The default value is "transactionManager".

#### **Child Elements of <jboss-transaction-manager...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Weblogic transaction manager**

The WebLogic transaction manager.

#### **Attributes of <weblogic-transaction-manager...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### **Child Elements of <weblogic-transaction-manager...>**

Name	Cardinality	Description
environment	0..1	The JNDI environment.

cache: Unexpected program error: java.lang.NullPointerException

#### **Jrun transaction manager**

The JRun transaction manager.

#### **Attributes of <jrun-transaction-manager...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no	transactionManager	An optional name for the transaction manager. The default value is "transactionManager".

#### **Child Elements of <jrun-transaction-manager...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Resin transaction manager

The Resin transaction manager.

### Attributes of <resin-transaction-manager...>

Name	Type	Required	Default	Description
name	name (no spaces)	no	transactionManager	An optional name for the transaction manager. The default value is "transactionManager".

### Child Elements of <resin-transaction-manager...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## Jndi transaction manager

Retrieves a named transaction manager factory from JNDI.

cache: Unexpected program error: java.lang.NullPointerException

## Custom transaction manager

A user-implemented transaction manager.

### Attributes of <custom-transaction-manager...>

Name	Type	Required	Default	Description
class	class name	yes		The class to instantiate to create a transaction manager.

### Child Elements of <custom-transaction-manager...>

Name	Cardinality	Description
environment	0..1	The JNDI environment.
spring:property	0..*	Spring-style property element for custom configuration.

Your Rating: 

Results:  0 rates

## BPM Configuration Reference

### BPM Configuration Reference

[ [Process](#) ] [ [Jbpm](#) ] [ [Exception Strategy](#) ] [ [Service](#) ] [ [Entry Point Resolver](#) ]

This page provides details on the process components you configure for BPM. Some of this information is pulled directly from `mule-bpm.xsd` and is cached. If the information appears to be out of date, refresh the page. For more information on BPM, see [BPM Module Reference](#).

#### Schema Reference on BPM XSD

cache: Unexpected program error: java.lang.NullPointerException

### Process

A process backed by a BPMS such as jBPM.

### Attributes of <process...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

bpms-ref	string	no		An optional reference to the underlying BPMS. This is used to disambiguate in the case where more than one BPMS is available.
processName	string	yes		The logical name of the process. This is used to look up the running process instance from the BPMS.
processDefinition	string	yes		The resource containing the process definition, this will be used to deploy the process to the BPMS. The resource type depends on the BPMS being used.
processIdField	string	no		This field will be used to correlate Mule messages with processes. If not specified, it will default to MULE_BPM_PROCESS_ID.

#### Child Elements of <process...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException  
 cache: Unexpected program error: java.lang.NullPointerException

#### Jbpm

#### Attributes of <jbpm...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		An optional name for this BPMS. Refer to this from the "bpms-ref" field of your process in case you have more than one BPMS available.
configurationResource	string	no		The configuration file for jBPM, default is "jbpm.cfg.xml" if not specified.
processEngine-ref	string	no		A reference to the already-initialized jBPM ProcessEngine. This is useful if you use Spring to configure your jBPM instance. Note that the "configurationResource" attribute will be ignored in this case.

#### Child Elements of <jbpm...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException  
 cache: Unexpected program error: java.lang.NullPointerException  
 cache: Unexpected program error: java.lang.NullPointerException  
 cache: Unexpected program error: java.lang.NullPointerException

#### Exception Strategy

See [Exception Strategy Configuration Reference](#).

#### Service

See [Service Configuration Reference](#).

#### Entry Point Resolver

See [Entry Point Resolver Configuration Reference](#).

Your Rating:  Results:  0 rates

## Transports Reference

### Available Transports

[ [Mule Transports](#) ] [ [Transport Feature Matrix](#) ]

This topic relates to the most recent version of Mule ESB

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

Your Rating:  5 stars

Results:  2 rates

Following is a list of known transports (also called "providers") for Mule ESB. Some functionality is contained within modules instead of transports--see [Modules Reference](#). For more information on transports, see the following topics:

- [About Transports](#)
- [Configuring a Transport](#)
- [Creating Transports](#)

If you have created a transport for Mule and would like to share it with the Mule community, please [contact us](#).

The following list includes some prominent transports from MuleForge (denoted by ) . Transports that are available only in Mule Enterprise are denoted by .

Note that in Mule 3, CXF and Jersey are no longer classed as transports. They are now modules that use an underlying transport (for instance HTTP or HTTPS) to communicate between client and service.

## Mule Transports

Transport	Description
AJAX Transport	The Mule AJAX connector allows Mule events to be sent and received asynchronously to and from the web browser
 Abdera Transport	Allows you to easily integrate with Atom feeds and Atom Publishing Protocol servers via the <a href="#">Apache Abdera</a> project.
EJB Transport	Allows EJB invocations to be made using outbound endpoints.
Email Transport	This transport supplies various email connectivity options.
File Transport	This transport allows files to be read and written to directories on the local file system. The connector can be configured to filter the file it reads and the way files are written, such as whether binary output is used or the file is appended to.
FTP Transport	Allows files to be read / written to a remote FTP server.
HTTP Transport	This transport supplies HTTP transport of Mule messages between applications and other Mule servers.
HTTPS Transport	A secure version of the HTTP transport.
IMAP Transport	Connectivity to IMAP mail folders.
IMAPS Transport	A secure version of the IMAP transport.
 JCR Transport	A transport that reads from, writes to, and observes JCR 1.0 containers. This transport is available on MuleForge.
 JDBC Transport	A transport for JDBC connectivity. Some of its features are available only in Mule Enterprise.
Jetty Transport	Provides support for exposing services over HTTP by embedding a light-weight Jetty server. For inbound endpoints only.
Jetty SSL Transport	A secure version of the Jetty transport.
JMS Transport	A Mule transport for JMS connectivity. Mule itself is not a JMS server but can use the services of any JMS 1.1 or 1.02b compliant server such as ActiveMQ and OpenJms, and commercial vendors such as Weblogic, SonicMQ, and more.

 LDAP Transport	Allows you to send and receive Mule Messages to/from an LDAP directory.
 Legs4Mule Transport	Provides transformers and connectors for IBM mainframes.
Multicast Transport	Allows your components to receive and send events via IP multicast groups.
POP3 Transport	Connectivity to POP3 inboxes.
POP3S Transport	A secure version of the POP3 transport.
Quartz Transport	Provides scheduling facilities with cron / interval definitions and allows Mule events to be scheduled/rescheduled.
 Restlet Transport	Allows you to embed Restlet services inside of Mule, use the Restlet client API over Mule, and use URI templates to route messages inside of Mule.
RMI Transport	Enables events to be sent and received over RMI via JRMP.
Servlet Transport	Provides facilities for Mule components to listen for events received via a servlet request. There is also a servlet implementation that uses the Servlet transport to enable REST style services access. This transport is now bundled with the HTTP transport.
SMTP Transport	Connectivity to SMTP servers.
SMTPS Transport	A secure version of the SMTP transport.
SSL Transport	Provides secure socket-based communication using SSL or TLS.
STDIO Transport	This transport provides connectivity to streams such as <code>System.in</code> and <code>System.out</code> and is useful for testing.
TCP Transport	Enables events to be sent and received over TCP sockets.
TLS Transport	Provides secure socket-based communication using SSL or TLS.
UDP Transport	Enables events to be sent and received as datagram packets.
VM Transport	Enables event sending and receiving over VM, embedded memory, or persistent queues.
 WebSphere MQ Transport	A Mule transport for WebSphere MQ. This transport is available with Mule Enterprise version 1.6 and later.
WSDL Connectors	The CXF Module allows remote web services to be invoked using their WSDL contract.
XMPP Transport	Provides connectivity over the XMPP (Jabber) instant messaging protocol.

## Transport Feature Matrix

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	MEP
AJAX	JavaDoc SchemaDoc	✓	✓	✗	✗	✓	✗	one-way	one-way	or
Axis	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	request-response	or
EJB	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	request-response	or

SMTP	JavaDoc SchemaDoc	✗	✓	✓	✗	✗	✗	one-way	one-way	or
SMTPS	JavaDoc SchemaDoc	✗	✓	✓	✗	✗	✗	one-way	one-way	or
IMAP	JavaDoc SchemaDoc	✓	✗	✗	✗	✗	✗	one-way	one-way	or
IMAPS	JavaDoc SchemaDoc	✓	✗	✗	✗	✗	✗	one-way	one-way	or
POP3	JavaDoc SchemaDoc	✓	✗	✓	✗	✗	✗	one-way	one-way	or
POP3S	JavaDoc SchemaDoc	✓	✗	✓	✗	✗	✗	one-way	one-way	or
File	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way	one-way	or
FTP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✓	one-way	one-way	or
HTTP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
HTTPS	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
JDBC	JavaDoc SchemaDoc	✓	✓	✓	✓ (local, XA)	✗	✓	one-way, request-response	one-way	or
Jetty	JavaDoc SchemaDoc	✓	✗	✓	✗	✓	✗	one-way, request-response	request-response	or
Jetty SSL	JavaDoc SchemaDoc	✓	✗	✓	✗	✓	✗	one-way, request-response	request-response	or
JMS	JavaDoc SchemaDoc	✓	✓	✓	✓ (client ack, local, XA)	✗	✓	one-way, request-response	one-way	or
Multicast	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	request-response	or
Quartz	JavaDoc SchemaDoc	✓	✓	✗	✗	✗	✗	one-way	one-way	or
RMI	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	request-response	or
Servlet	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	request-response	request-response	or
SFTP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	one-way	or
SSL	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
TLS	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
STDIO	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way	one-way	or
TCP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
UDP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	or
VM	JavaDoc SchemaDoc	✓	✓	✓	✓ (XA)	✓	✗	one-way, request-response	one-way	or

XMPP	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	one-way	or
jdbc-ee	JavaDoc SchemaDoc	✗	✗	✗	✗	✗	✗			or
ftp-ee	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗			or

#### Legend

▶ Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in Maven

Was this article helpful?  

## AJAX Transport Reference

### AJAX Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Using the JavaScript Client](#) ] [ [Listening to Server Events](#) ] [ [Sending a Message](#) ] [ [Example Configurations](#) ] [ [Configuration Reference](#) ] [ [Maven](#) ] [ [Best Practices](#) ]

#### Introduction

AJAX (Asynchronous Java and XML) is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications. With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.

The Mule AJAX connector allows Mule events to be sent and received asynchronously to and from the web browser. The connector includes a JavaScript client that can be used to listen for events, send events and perform RPC calls. It can be deployed in Mule standalone or embedded in a servlet container such as Apache Tomcat or Tcat Server.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
AJAX	JavaDoc SchemaDoc	✓	✓	✗	✗	✓	✗	one-way	one-way	org.mule.transport:n

#### Legend

▶ Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

The AJAX transport has server and client parts. The server part is similar to every other Mule transport. The client part is a [JavaScript client](#) that enables users to publish and subscribe to Mule messages in the browser. The server connector can either run as an embedded AJAX (cometD) server or [bind to a servlet container](#). The following provides detail on configuring the server side of the AJAX support.

## Namespace and Syntax

XML namespace:

```
xmlns:ajax="http://www.mulesoft.org/schema/mule/ajax"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/ajax http://www.mulesoft.org/schema/mule/ajax/3.1/mule-ajax.xsd
```

Connector Syntax:

```
<ajax:connector name="ajaxServer" serverUrl="http://0.0.0.0:8090/mule/ajaxServer" resourceBase="${app.home}/docroot"/>
```

Endpoint Syntax

1. Inbound endpoint

```
<ajax:inbound-endpoint channel="/services/someAjaxService" />
```

2. Outbound endpoint

```
<ajax:outbound-endpoint channel="/mule/notifications" cacheMessages="true" />
```

## Considerations

You should use the Mule AJAX Transport for bi-directional communication with the browser. Under the covers it uses CometD with continuations enabling higher numbers of concurrent requests and optimised server resource usage.

## Features

- Easily perform asynchronous calls to Mule
- Broadcast events to browsers listening to AJAX channels
- Perform Sudo - RPC calls from the browser
- Use stand-alone or bound to a servlet container

## Usage

To the AJAX transport in your configuration, you need to add the ajax namespaces:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:ajax="http://www.mulesoft.org/schema/mule/ajax"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/ajax
          http://www.mulesoft.org/schema/mule/ajax/3.1/mule-ajax.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">
...

```

## Configuring the Server

The usual way of setting up the Ajax Server is to use the one embedded in Mule. This can be created by adding an `ajax:connector` element to your config:

```
<ajax:connector serverUrl="http://0.0.0.0:8080/ajax"/>
```

This starts an ajax server and is ready to start publishing and subscribing. Next you can create a Flow that listens to ajax messages on a channel:

```

<flow name="AjaxEcho">
  <ajax:inbound-endpoint channel="/services/echo" />
  <echo-component />
</flow>

```

Or to publish on an ajax channel, use an outbound endpoint:

```

<flow name="AjaxBridge">
  <jms:inbound-endpoint topic="football.scores" />

  <ajax:outbound-endpoint channel="/football/scores" />
</flow>

```

## Embedding in a Servlet Container

If you are running Mule inside a servlet container such as Apache Tomcat you will probably want to bind any ajax endpoints to the servlet container. To do this you need to add the `org.mule.transport.ajax.container.MuleAjaxServlet` to your `web.xml` and you need to use the `<ajax:servlet-xxx-endpoint>` elements.

Configure your `{[web.xml]}` using:

```

<servlet>
  <servlet-name>ajax</servlet-name>
  <servlet-class>org.mule.transport.ajax.container.MuleAjaxServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>ajax</servlet-name>
  <url-pattern>/ajax/*</url-pattern>
</servlet-mapping>

```

Then replace any `<ajax:inbound-endpoint>` and `<ajax:outbound-endpoint>` with `<ajax:servlet-inbound-endpoint>` and `<ajax:servlet-outbound-endpoint>` respectively. To use the footballs scores example again:

```

<flow name="AjaxBridge">
    <jms:inbound-endpoint topic="football.scores"/>

    <ajax:servlet-outbound-endpoint channel="/football/scores"/>
</flow>

```

Then you need to configure your connector and endpoints as described below.

## Using the JavaScript Client

Mule provides a powerful JavaScript client with full [Ajax](#) support that can be used to interact with Mule services directly in the browser. It also provides support for interacting directly with objects running inside the container using [Cometd](#), a message bus for Ajax web applications that allows multi-channel messaging between the server and client.

### Configuring the Server

To use the JavaScript client, you just need to have a service that has an AJAX inbound endpoint through which requests can be sent. The example below shows a simple echo service published on the /services/echo AJAX channel.

```

<flow name="AjaxEcho">
    <ajax:inbound-endpoint channel="/services/echo"/>
    <echo-component/>
</flow>

```

### Enabling the Client

To enable the client in an HTML page, you add a single script element to the page:

```

<head>
...
<script type="text/javascript" src="mule-resource/js/mule.js"></script>

```

Adding this script element will make a 'mule' client object available for your page.

### Making an RPC request

Let's say there is a button defined in the body that, when clicked, sends a request to the Echo service:

```

<input id="sendButton" class="button" type="submit" name="Go" value="Send" onclick="callEcho();"/>

```

The button calls the `callEcho` function, which handles the logic of the request:

```

function callEcho()
{
    var data = new Object();
    data.phrase = document.getElementById('phrase').value;
    mule.rpc("/services/echo", data, callEchoResponse);
}

```

This function uses the `rpc` method to request data from the service. The `rpc` method sets up a private response channel that Mule will publish when the response data is available. The first argument is the channel on which you're making the request (this matches the channel that our Echo Service is listening on), the second argument is the payload object, and the third argument is the callback function that processes the response, in this case a function called `callEchoResponse`:

```

function callEchoResponse(message)
{
    document.getElementById("response").innerHTML = "<b>Response:&nbsp;</b>" + message.data + "\n";
}

```

In cases where `rpc` is to be used just for a one-way request (no callback function will be passed as parameter as no response is expected) it is recommended to use the `disableReplyTo` flag in the AJAX connector:

```
<ajax:connector name="ajaxServer" ... disableReplyTo="true" />
```

## Handling Errors

To check if an error occurred, set the `error` parameter in the callback function to verify that the error is null before processing. If it is not null, an error occurred and the error should be logged or displayed to the user.

```

function callEchoResponse(message, error)
{
    if(error)
        handleError(error)
    else
        document.getElementById("response").innerHTML = "<b>Response:&nbsp;</b>" + message.data + "\n";
}

function handleError(error) {
    alert(error);
}

```

## Listening to Server Events

The Mule JavaScript client allows developers to subscribe to events from Mule services. These events just need to be published on an AJAX endpoint i.e. Here is a service that receives events on JMS and publishes them to an AJAX channel -

```

<flow name="AjaxBridge">
    <jms:inbound-endpoint topic="football.scores"/>

    <ajax:outbound-endpoint channel="/football/scores"/>
</flow>

```

Now you can register for interest in these football scores by adding a subscriber via the Mule JavaScript client -

```

<script type="text/javascript">
    mule.subscribe("/football/scores", scoresCallback);
</script>

```

The first argument of the `subscribe` method is the AJAX path that the service publishes to. The second argument is the name of the callback function that processes the message. In this example, it's the `scoresCallback` function, which is defined next:

```

function scoresCallback(message)
{
    console.debug("data:" + message.data);

    if (!message.data)
    {
        console.debug("bad message format " + message);
        return;
    }

    // logic goes here
    ...
}

```



### JSON Support

Mule 3.0 now has JSON Support including object/json bindings, this makes it really easy to marshal data to JSON markup before dispatching to the browser, where JSON is a native format.

## Sending a Message

Let's say you want to send a message out without getting a response. In this case, you call the `publish` function on the Mule client:

```

<script type="text/javascript">
    mule.publish("/services/foo", data);
</script>

```

## Example Configurations

Mule comes bundled with several examples that employ the Ajax Connector. We recommend you take a look at the "Notifications Example" and the "GPS Walker Example" (which is also explained in further detail in [this blog post](#)). In the following typical use cases we will focus on the key elements involved when using and configuring the connector.

### **Publish Example Server code**

First, we will set up an AJAX inbound endpoint in the Mule configuration to receive requests.

#### Configuring an AJAX Inbound Endpoint

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:ajax="http://www.mulesoft.org/schema/mule/ajax"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/ajax
          http://www.mulesoft.org/schema/mule/ajax/3.1/mule-ajax.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <ajax:connector name="ajaxServer" serverUrl="http://0.0.0.0:8090/services/updates"
                    resourceBase="${app.home}/docroot"/>

    <flow name="TestNoReply">
        <ajax:inbound-endpoint channel="/services/serverEndpoint" />
        <!-- From here on, the data from the browser will be available in Mule. -->
        ...
        <component .../>
    </flow>

</mule>

```

Please note the following changes:

- The Mule Ajax namespace and schema location have been added to the *mule* element.
- The Ajax Connector creates an embedded Ajax server for this application.
  - The ‘resourceBase’ attribute specifies a directory where HTML and other resources can be published. When the browser requests pages, they will be served from this location.
  - The \${app.home} is a new placeholder available in Mule that references the root directory of your application.
  - ‘0.0.0.0’ refers to the IP of the computer running the Mule instance.
- An Ajax inbound endpoint has been added to a sample flow. It will create a channel named /services/serverEndpoint and listen to incoming messages from the Mule JavaScript client.

### Publish Example Client code

The browser will send some information to Mule (using the JavaScript Mule client) when a button is pushed.

```
Publishing data

<head>
    <script type="text/javascript" src="mule-resource/js/mule.js"></script>
    <script type="text/javascript">

        function publishToMule() {
            // Create a new object and populate it with the request data
            var data = new Object();
            data.phrase = document.getElementById('phrase').value;
            data.user = document.getElementById('user').value;
            // Send the data to the mule endpoint and do not expect a response.
            // The "mule" element is provided by the Mule JavaScript client.
            mule.publish("/services/serverEndpoint", data);
        }
    </script>
</head>

<body>
    <div>
        Your phrase: <input id="phrase" type="text" />
        <select id="user">
            <option value="anonymous">Anonymous</option>
            <option value="administrator" selected="true">Administrator</option>
        </select>
        <input id="sendButton" class="button" type="submit" name="Go" value="Send" onclick="publishToMule();"/>
    </div>
</body>
```

Please note the following changes:

- Loading the *mule.js* script will make the Mule client automatically available via the ‘*mule*’ variable.
- The *rpcCallMule()* method will gather some data from the page and submit it to the ‘/services/noReplyEndpoint’ channel we configured beforehand.
- The *mule.publish()* method makes the actual call to Mule. It receives two parameters:
  - The channel name.
  - The data to publish.

### Subscribe Example Server code

This is a useful and friendly way to send information to several clients at once. All they have to do is subscribe themselves to a channel where the server will send whatever needs to be broadcasted.

Mule ESB provides an AJAX connector, an AJAX outbound endpoint and the required JavaScript client library to take care of this.

We will add an AJAX connector that will host the pages (HTML, CSS, etc.) using the JavaScript client and that will let them interact with Mule’s AJAX endpoints. It’s the same connector we used in the two previous examples.

We also need to publish some content via an AJAX outbound endpoint in a channel.

## Configuring an AJAX Outbound Endpoint Channel

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:ajax="http://www.mulesoft.org/schema/mule/ajax"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/ajax
          http://www.mulesoft.org/schema/mule/ajax/3.1/mule-ajax.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <ajax:connector name="ajaxServer" serverUrl="http://0.0.0.0:8090/services/updates"
                    resourceBase="${app.home}/docroot"/>

    <flow name="PublishUpdates">
        <!-- ... here we create the content to be published -->
        <ajax:outbound-endpoint channel="/mule/notifications" cacheMessages="true"/>
    </flow>

</mule>
```

Please note the following changes:

- The Mule Ajax namespace and schema location have been added to the *mule* element.
- The Ajax Connector creates an embedded Ajax server for this application.
  - The 'resourceBase' attribute specifies a directory where HTML and other resources can be published. When the browser requests pages, they will be served from this location.
  - The \${app.home} is a new placeholder available in Mule that references the root directory of your application.
  - '0.0.0.0' refers to the IP of the computer running the Mule instance.
- An Ajax outbound endpoint has been added to a sample flow.
  - It will submit the events it receives into a channel named */mule/notifications*.
  - Any page listening on that channel will receive a copy of the event.

## Subscribe Example Client code

### Listening to an AJAX Outbound Channel

```
<head>
    <script type="text/javascript" src="mule-resource/js/mule.js"></script>

    <script type="text/javascript">

        function init()
        {
            mule.subscribe( "/mule/notifications", notif);
        }

        function dispose()
        {
            mule.unsubscribe( "/mule/notifications", notif);
        }

        function notif(message)
        {
            console.debug( "data: " + message.data);

            //... code to handle the received data
        }

    </script>
</head>

<body onload="init()" onunload="dispose()">

</body>
```

Please note the following changes:

- Loading the *mule.js* script will make the Mule client automatically available via the '*mule*' variable.
- The *init()* method will associate all incoming events on the '*/mule/notifications*' with the *notif()* callback method.
- The *dispose()* method will dissociate all incoming events on the '*/mule/notifications*' from the *notif()* callback method.
- The *notif()* callback method will process the received messages.
- The *onload* and *onunload* attributes of the *body* HTML element should contain the calls to *init()* and *dispose()* respectively, to ensure the page is properly registered and de-registered to the '*/mule/notifications*' channel.

### RPC Example Server code

This configuration is very similar to the one in the previous example. As a matter of fact, the only significant changes are the channel name and an out-of-the-box echo component to bounce the request back to the caller.

#### Configuring an AJAX Inbound Endpoint that will send a response

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:ajax="http://www.mulesoft.org/schema/mule/ajax"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/ajax
          http://www.mulesoft.org/schema/mule/ajax/3.1/mule-ajax.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <ajax:connector name="ajaxServer" serverUrl="http://0.0.0.0:8090/services/updates"
                    resourceBase="${app.home}/docroot"/>

    <flow name="TestEcho">
        <ajax:inbound-endpoint channel="/services/echo" />
        <echo-component/>
    </flow>

</mule>
```

Please note the following changes:

- The Mule Ajax namespace and schema location have been added to the *mule* element.
- The Ajax Connector creates an embedded Ajax server for this application.
  - The 'resourceBase' attribute specifies a directory where HTML and other resources can be published. When the browser requests pages, they will be served from this location.
  - The \${app.home} is a new placeholder available in Mule that references the root directory of your application.
  - '0.0.0.0' refers to the IP of the computer running the Mule instance.
- An Ajax inbound endpoint has been added to a sample flow.
  - It will create a channel named /services/echo and listen to incoming RPC calls from the Mule JavaScript client.
  - When a request is received, it will be processed by the *<echo-component/>* and sent back via the Ajax channel to the client that submitted the request.

### RPC Example Client code

The browser will send some information to Mule (using the JavaScript Mule client) when a button is pushed, just as it did before. This time however, a callback method will display the response.

## Making an RPC Call - Expecting a response

```
<head>
    <script type="text/javascript" src="mule-resource/js/mule.js"></script>
    <script type="text/javascript">

        function rpcCallMuleEcho() {
            // Create a new object and populate it with the request data
            var data = new Object();
            data.phrase = document.getElementById('phrase').value;
            data.user = document.getElementById('user').value;
            // Send the data to the mule endpoint and set a callback to handle the response.
            // The "mule" element is provided by the Mule JavaScript client.
            mule.rpc("/services/echo", data, rpcEchoResponse);
        }

        // Display response message data.
        function rpcEchoResponse(message) {
            document.getElementById("response").innerHTML = "<b>Response:&nbsp;</b>" + message.data +
        "\n";
        }
    </script>
</head>

<body>
    <div>
        Your phrase: <input id="phrase" type="text" />
        <select id="user">
            <option value="anonymous">Anonymous</option>
            <option value="administrator" selected="true">Administrator</option>
        </select>
        <input id="sendButton" class="button" type="submit" name="Go" value="Send" onclick=
"rpcCallMuleEcho();"/>
    </div>
    <pre id="response"></pre>
</body>
```

Please note the following changes:

- Loading the `mule.js` script will make the Mule client automatically available via the '`mule`' variable.
- The `rpcCallMuleEcho()` method will gather some data from the page and submit it to the '`/services/echo`' channel we configured beforehand.
- The `mule.rpc()` method makes the actual call to Mule. This time, it receives **three** parameters:
  - The channel name.
  - The data to send.
  - The **callback method** to be invoked when the response is returned.
- The `rpcEchoResponse()` callback method takes a single parameter, which is the response message, and displays its data on the page.

## Configuration Reference

### Element Listing

cache: Unexpected program error: java.lang.NullPointerException

### Connector

Allows Mule to expose Mule Services over HTTP using a Jetty HTTP server and Cometd. A single Jetty server is created for each connector instance. One connector can serve many endpoints. Users should rarely need to have more than one AJAX servlet connector.

#### Attributes of `<connector...>`

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
serverUrl	string	yes		<p>When using AJAX embedded (not within a servlet container) a URL needs to be configured to create an AJAX server hosted in Mule. The URL should be in the form of <a href="http://(host):(port)/(path)">http://(host):(port)/(path)</a></p> <p>) note that https can also be used, but you will need to set the TLS information on the connector.</p>
resourceBase	string	no		Specifies a local path where files will be served from. The local path gets mapped directly to the path on the 'serverUrl'.
disableReplyTo	boolean	no		By default, an asynchronous reply to the inbound endpoint is sent back. This can cause unwanted side effects in some cases, use this attribute to disable.
logLevel	integer	no		0=none, 1=info, 2=debug
timeout	integer	no		The server side poll timeout in milliseconds (default 250000). This is how long the server will hold a reconnect request before responding.
interval	integer	no		The client side poll timeout in milliseconds (default 0). How long a client will wait between reconnects
maxInterval	integer	no		The max client side poll timeout in milliseconds (default 30000). A client will be removed if a connection is not received in this time.
jsonCommented	boolean	no		If "true" (default) then the server will accept JSON wrapped in a comment and will generate JSON wrapped in a comment. This is a defence against Ajax Hijacking.
multiFrameInterval	integer	no		The client side poll timeout if multiple connections are detected from the same browser (default 1500).
refsThreshold	integer	no		The number of message refs at which a single message response will be cached instead of being generated for every client delivered to. Done to optimize a single message being sent to multiple clients.

#### Child Elements of <connector...>

Name	Cardinality	Description
client	0..1	
key-store	0..1	
server	0..1	
protocol-handler	0..1	

cache: Unexpected program error: java.lang.NullPointerException

#### Inbound endpoint

Allows a Mule service to receive AJAX events over HTTP using a Jetty server. This is different from the equivalent `servlet-inbound-endpoint` because it uses an embedded servlet container rather than relying on a pre-existing servlet container instance. This endpoint type should not be used if running Mule embedded in a servlet container.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.

ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
channel	string	yes		the ajax channel to bind the service endpoint to. This channel path is independent context path that your application is deployed to in the servlet container.

**Child Elements of <inbound-endpoint...>**

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

#### Outbound endpoint

Allows a Mule service to send AJAX events over HTTP using Bayeux. JavaScript clients can register interest in these events using the Mule JavaScript client.

##### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
channel	string	yes		the ajax channel to bind the service endpoint to. This channel path is independent context path that your application is deployed to in the servlet container.
cacheMessages	boolean	no		If set to true the dispatcher will cache messages if there are no clients subscribed to this channel.
messageCacheSize	int	no		If cache messages is set to true, this value determines the size of the memory cache. The cache will automatically expire older items to make room for newer items.

##### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.

abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

## Maven

The AJAX Transport can be included with the following dependency:

```
<dependency>
    <groupId>org.mule.transports</groupId>
    <artifactId>mule-transport-ajax</artifactId>
</dependency>
```

## Best Practices

- Use AJAX Outbound Endpoints mainly for broadcasting information to several clients simultaneously. E.G.: Broadcasting live news updates to several browsers in real time without reloading the page.
- It's recommended to subscribe/unsubscribe callback methods associated with outbound channels on `<body> onload/onunload`. See example above. Pay special attention to unsubscribing callback methods.
- When sending information back and forth between clients and servers using AJAX you should consider using JSON. Mule provides a JSON module to handle transformations gracefully.

Your Rating: 

Results:  3 rates

## EJB Transport Reference

### EJB Transport Reference

The EJB transport allows EJB session beans to be invoked as part of an event flow. Components can be given an EJB outbound endpoint, which will invoke the remote object and optionally return a result.

The Javadoc for this transport can be found [here](#).

cache: Unexpected program error: java.lang.NullPointerException

### Connector

The Mule EJB Connector provides connectivity for EJB beans.

## Attributes of <connector...>

Name	Type	Required	Default	Description
pollingFrequency	long	no		Period (ms) between polling connections.
securityManager-ref	name (no spaces)	no		Bean reference to the security manager that should be used.
securityPolicy	string	no		The security policy (file name) used to enable connections.
serverClassName	name (no spaces)	no		The target class name.
serverCodebase	string	no		The target method.

## Child Elements of <connector...>

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<ejb:connector name="ejb" jndiContext-ref="jndiContext" securityPolicy="rmi.policy" />
```

## Using the EJB Transport

To use the EJB transport, you must define the EJB namespace and schema location at the top of the Mule configuration file. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:ejb="http://www.mulesoft.org/schema/mule/ejb"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/ejb
          http://www.mulesoft.org/schema/mule/ejb/3.0/mule-ejb.xsd">
    ...

```

EJB endpoints are configured the same way as RMI endpoints. Note that only outbound endpoints can use the EJB transport. For a given endpoint, you must provide the following information:

- registry host
- registry port
- remote home name
- remote method name

These values will be used to establish the dispatcher connection. For example:

```
<ejb:endpoint host="localhost" port="1099" object="SomeService" method="remoteMethod"/>
```

Alternatively, you could use URI-based configuration:

```
<outbound-endpoint address="ejb://localhost:1099/SomeService?method=remoteMethod"/>
```

If the method can take one or more input arguments, you configure their types as a comma-separated list using the `methodArgumentTypes` attribute. Multiple arguments are passed in as an array of objects as the payload of the Mule message.

For an example of using the EJB transport to integrate with BEA WebLogic, see the cookbook entry [Integration with BEA WebLogic](#).

## Transformers

No specific transformers are required for EJB.

Your Rating:  Results:  1 rates

## Email Transport Reference

### Email Transport Reference

#### Introduction

The email transport comprises of several transports for email connectivity. The implementation supports CC/BCC/ReplyTo addresses, attachments, custom Header properties and customizable authentication. See each email transport page for more details.

- SMTP
- IMAP
- POP3

#### Configuration Examples

Here is a configuration snippet from the bookstore example. This flow is triggered when you send a string to vm://emailNotification and sends an order verification email to a specified address:

```
<flow name="EmailNotificationService">
    <vm:inbound-endpoint path="emailNotification" exchange-pattern="one-way" />
    <smtplib:outbound-endpoint user="${user}" password="${password}" host="${host}" from="${from}"
        subject="Your order has been placed!">
        <custom-transformer class="org.mule.example.bookstore.transformers.OrderToEmailTransformer" />
        <email:string-to-email-transformer />
    </smtplib:outbound-endpoint>
</flow>
```

Here is a configuration snippet from the errorhandler example. This endpoint takes a ErrorMessageToExceptionBean java object and converts it to an email to a specified address:

```
<smtplib:outbound-endpoint user="${smtp.username}" password="${smtp.password}"
    host="${smtp.host}" port="${smtp.port}"
    to="${email.toAddress}" from="${email.fromAddress}"
    subject="${email.subject}">
    <transformer ref="ErrorMessageToExceptionBean" />
    <transformer ref="ExceptionBeanToXML" />
    <email:string-to-email-transformer/>
</smtplib:outbound-endpoint>
```

You can find full working configuration examples in each transport reference page.

The full example code for the bookstore and errorhandler examples are included in the full Mule ESB distribution.

Your Rating:  Results:  2 rates

## Email Transport Filters

#### Filters

Filters can be set on an endpoint to filter out any unwanted messages. The Email transport provides a couple of filters that can either be used directly or extended to implement custom filtering rules.

Filter	Description

org.mule.providers.email.filters.AbstractMailFilter	A base filter implementation that must be extended by any other mail filter.
org.mule.providers.email.filters.MailSubjectRegExFilter	Applies a regular expression to a Mail Message subject.

This is how you define the MailSubjectRegExFilter in your Mule configuration:

```
<message-property-filter pattern="to=barney@mule.org" />
```

The 'pattern' attribute is a regular expression pattern. This is defined as java.util.regex.Pattern.

Your Rating:  Results:  1 rates

## Email Transport Limitations

### Limitations

The following known limitations affect email transports:

- Retry policies do not work with email transports
- Timeouts are not supported in email transports
- Can't send same object to different email users
- MailSubjectRegExFilter cannot handle mails with attachments

Your Rating:  Results:  2 rates

## Email Transport Transformers

**cache: Unexpected program error: java.lang.NullPointerException**

### Transformers

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description
email-to-string-transformer	Converts an email message to string format.
string-to-email-transformer	Converts a string message to email format.
object-to-mime-transformer	Converts an object to MIME format.
mime-to-bytes-transformer	Converts a MIME message to a byte array.
bytes-to-mime-transformer	Converts a byte array message to MIME format.

Here is how you define transformers in your Mule configuration file:

```

<email:bytes-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:email-to-string-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:mime-to-bytes-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:object-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" useInboundAttachments="true" useOutboundAttachments="true"/>
{Note}Need to explain attributes somewhere; can we pull them in from xsd?{Note}
<email:string-to-email-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />

```

Each transformer supports all the common transformer attributes and properties:

**cache:** Unexpected program error: java.lang.NullPointerException

### Transformer

A reference to a transformer defined elsewhere.

#### Attributes of <transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
ref	string	yes		The name of the transformer to use.

#### Child Elements of <transformer...>

Name	Cardinality	Description
------	-------------	-------------

The object-to-mime-transformer has the following attributes:

Attribute	Description	Default Value
useInboundAttachments	Whether to transform inbound attachment in the input message into MIME parts.	true
useOutboundAttachments	Whether to transform outbound attachment in the input message into MIME parts.	true

To use these transformers, make sure you include the 'email' namespace in your mule configuration.

Your Rating:  Results:  1 rates

## SMTP Transport Reference

## SMTP Transport Reference

### Introduction

The SMTP transport can be used for sending messages over SMTP using the `javax.mail` API. The implementation supports CC/BCC/ReplyTo addresses, attachments and custom Header properties. It also provides support for `javax.mail.Message` transformation. The SMTPTS connector enables SMTP over SSL using the `javax.mail` APIs. It supports all the elements and attributes of the SMTP transport, plus some required properties for setting up the client key store and the trust store for the SSL connection.

TLS/SSL connections are made on behalf of an entity, which can be anonymous or identified by a certificate. The *key store* provides the certificates and associated private keys necessary for identifying the entity making the connection. Additionally, connections are made to trusted systems. The public certificates of trusted systems are stored in a *trust store*, which is used to verify that the connection made to a remote system matches the expected identity.

### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
SMTP	JavaDoc SchemaDoc	✗	✓	✓	✗	✗	✗	one-way	one-way	org.mule.transport:n
SMTPTS	JavaDoc SchemaDoc	✗	✓	✓	✗	✗	✗	one-way	one-way	org.mule.transport:n

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name and artifact name for this transport in Maven

### Namespace and Syntax

XML namespace:

```
xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
xmlns:smtpts="http://www.mulesoft.org/schema/mule/smtpts"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/schema/mule/smtp/3.1/mule-smtp.xsd
http://www.mulesoft.org/schema/mule/smtpts http://www.mulesoft.org/schema/mule/smtpts/3.1/mule-smtpts.xsd
```

Connector syntax:

```

<smtp:connector name="smtpConnector" bccAddresses="abc@example.com" ccAddresses="bcd@example.com"
contentType="foo/bar"
fromAddress="cde@example.com" replyToAddresses="def@example.com"
subject="subject">
  <smtp:header key="foo" value="bar" />
  <smtp:header key="baz" value="boz" />
</smtp:connector>

<smtps:connector name="smptsConnector" bccAddresses="abc@example.com" ccAddresses="bcd@example.com"
contentType="foo/bar"
fromAddress="cde@example.com" replyToAddresses="def@example.com"
subject="subject">
  <smpts:header key="foo" value="bar" />
  <smpts:header key="baz" value="boz" />
  <smpts:tls-client path="clientKeystore" storePassword="mulepassword" />
  <smpts:tls-trust-store path="greenmail-truststore" storePassword="password" />
</smpts:connector>

<smtp:gmail-connector name="smtpGmailConnector" bccAddresses="abc@example.com" ccAddresses=
"bcd@example.com" contentType="foo/bar"
fromAddress="cde@example.com" replyToAddresses="def@example.com"
subject="subject">
  <smtp:header key="foo" value="bar" />
  <smtp:header key="baz" value="boz" />
</smtp:gmail-connector>

```

#### Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```

<smtp:outbound-endpoint host="localhost" port="65437" from="steve@mycompany.com"
to="bob@example.com" subject="Please verify your account details"/>
<smpts:outbound-endpoint host="localhost" port="65437" from="steve@mycompany.com"
to="bob@example.com" subject="Please verify your account details"/>

```

2. Non-prefixed URI:

```

smtp://muletestbox:123456@smtp.mail.yahoo.co.uk?address=dave@mycompany.com
smpts://muletestbox:123456@smtp.mail.yahoo.co.uk?address=dave@mycompany.com

```

See the sections below for more information.

### Features

- Simple to configure email access on outbound endpoints
- Easy to configure TLS security

### Usage

If you want to include the SMTP email transport in your configuration, these are the namespaces you need to define:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/smtp http://www.mulesoft.org/schema/mule/smtp/3.1/mule-smtp.xsd">
  ...

```

Secure version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:smpts="http://www.mulesoft.org/schema/mule/smpts"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/smpts
    http://www.mulesoft.org/schema/mule/smpts/3.1/mule-smpts.xsd">

```

Then you need to configure your connector and endpoints as described below.

#### Configuration Example

Say your CFO wants an email notification of all processed orders. The following configuration will pick up any files in the 'processed' directory, convert them to a string and send it as the email body to the CFO.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:smpts="http://www.mulesoft.org/schema/mule/smpts"
  xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
  xmlns:file="http://www.mulesoft.org/schema/mule/file"
  xmlns:email="http://www.mulesoft.org/schema/mule/email"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
    http://www.mulesoft.org/schema/mule/smpts http://www.mulesoft.org/schema/mule/smpts/3.1/mule-smpts.xsd
    http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
    http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

  <smpt:connector name="smtpConnector" />

  <flow name="processed-orders">
    <file:inbound-endpoint path="/tmp/processed">
      <file:file-to-string-transformer/>
    </file:inbound-endpoint>
    <smpt:outbound-endpoint host="smptsServer" port="25" from="bob" subject="processed order" to="cfo@example.com">
      <email:string-to-email-transformer/>
    </smpt:outbound-endpoint>
  </flow>
</mule>

```

This configuration defines an inbound file endpoint which looks in the '/tmp/processed' directory and converts any files found to a string. An outbound smtp server is defined on . A string-to-email-transformer will convert the string to email format before the email is sent.

Secure version:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:email="http://www.mulesoft.org/schema/mule/email"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/smtps http://www.mulesoft.org/schema/mule/smtps/3.1/mule-smtps.xsd
          http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <smtps:connector name="smptsConnector">
        <smpts:tls-client path="clientKeystore" storePassword="mulepassword" />
        <smpts:tls-trust-store path="greenmail-truststore" storePassword="password" />
    </smpts:connector>

    <flow name="processed-orders">
        <file:inbound-endpoint path="/tmp/processed">
            <file:file-to-string-transformer/>
        </file:inbound-endpoint>
        <smpts:outbound-endpoint host="smtpsServer" port="25" from="bob" subject="processed order" to="cfo@example.com">
            <email:string-to-email-transformer/>
        </smpts:outbound-endpoint>
    </flow>
</mule>
```

The smpts connector has a TLS client and server keystore information as defined on . An inbound file endpoint looks in the '/tmp/processed' directory and converts any files found to a string. An outbound smtp server is defined on . A string-to-email-transformer will convert the string to email format before the email is sent.

## Configuration Reference

### Connectors

The SMTP connector supports all the common connector attributes and properties and the following optional elements and attributes:

Attribute	Description	Default	Required
bccAddresses	Comma separated list of addresses for blind copies.		False
ccAddresses	Comma separated list of addresses for copies.		False
contentType	Mime type for the outgoing message.		False
fromAddress	The from address for the outgoing message.		False
replyToAddresses	The reply-to address for the outgoing message.		False
subject	The default subject for the outgoing message if none is set in the message.		False

Element	Description
header	Additional header name and value, added to the message.

For the secure version, the following elements are also required:

Element	Description
tls-client	Configures the client key store with the following attributes: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the keystore that contains public certificates and private keys for identification</li> <li>storePassword: The password used to protect the keystore</li> <li>class: The type of keystore used (a Java class name)</li> </ul>
tls-trust-store	Configures the trust store. The attributes are: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the trust store that contains public certificates of trusted servers</li> <li>storePassword: The password used to protect the trust store</li> </ul>

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:spring="http://www.springframework.org/schema/beans"
       xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
           http://www.mulesoft.org/schema/mule/smtp
           http://www.mulesoft.org/schema/mule/smtp/3.0/mule-smtp.xsd">
...
<smtp:connector name="smtpConnector" bccAddresses="abc@example.com" ccAddresses="bcd@example.com"
contentType="foo/bar"
fromAddress="cde@example.com" replyToAddresses="def@example.com"
subject="subject">
    <smtp:header key="foo" value="bar" />
    <smtp:header key="baz" value="boz" />
</smtp:connector>

```

Secure version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:smtps="http://www.mulesoft.org/schema/mule/smtps"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/smtps
          http://www.mulesoft.org/schema/mule/smtps/3.0/mule-smtps.xsd">

<smtps:connector name="smtpsConnector">
    <smtps:tls-client path="clientKeystore" storePassword="mulepassword" />
    <smtps:tls-trust-store path="greenmail-truststore" storePassword="password" />
</smtps:connector>
<model name="test">
    <service name="relay">
        <inbound>
            <vm:inbound-endpoint path="send" />
        </inbound>
        <outbound>
            <pass-through-router>
                <smtps:outbound-endpoint host="localhost" port="65439" to="bob@example.com" />
            </pass-through-router>
        </outbound>
    ...

```

The gmail-connector connector supports all of the above.

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:smtp="http://www.mulesoft.org/schema/mule/smtp"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/smtp
          http://www.mulesoft.org/schema/mule/smtp/3.0/mule-smtp.xsd">
    ...
    <smtp:gmail-connector name="smtpGmailConnector" bccAddresses="abc@example.com" ccAddresses="bcd@example.com" contentType="foo/bar" fromAddress="cde@example.com" replyToAddresses="def@example.com" subject="subject">
        <smtp:header key="foo" value="bar" />
        <smtp:header key="baz" value="boz" />
    </smtp:gmail-connector>

```

## Endpoints

SMTP endpoints describe details about the SMTP server and the recipients of messages sent from the SMTP endpoint. You configure the endpoints just as you would with any other transport, with the following additional attributes:

Attribute	Description
user	The user name of the mailbox owner
password	The password of the user
host	The IP address of the SMTP server, such as www.mulesoft.com, localhost, or 127.0.0.1

port	The port number of the SMTP server
to	The destination for the email
from	The address of the sender of the email
subject	The email subject
cc	A comma-separated list of email addresses to copy on this email
bcc	A comma-separated list of email addresses to blind-copy on this email
replyTo	The address used by default if someone replies to the email

For example:

```
<outbound>
  <pass-through-router>
    <smtp:outbound-endpoint host="localhost" port="65437" from="steve@mycompany.com"
                           to="bob@example.com" subject="Please verify your account details"/>
  </pass-through-router>
</outbound>
```

Secure version:

```
<outbound>
  <pass-through-router>
    <smtpls:outbound-endpoint host="localhost" port="65437" from="steve@mycompany.com"
                           to="bob@example.com" subject="Please verify your account details"/>
  </pass-through-router>
</outbound>
```

You can also define the endpoints using a URI syntax:

```
<outbound-endpoint address="smtp://muletestbox:123456@smtp.mail.yahoo.co.uk?address=dave@mycompany.com"/>
<outbound-endpoint address="smtpls://muletestbox:123456@smtp.mail.yahoo.co.uk?address=dave@mycompany.com"/>
```

This will send mail using `smtp.mail.yahoo.co.uk` (using the default SMTP port) to the address `dave@mycompany.com`. The SMTP request is authenticated using the username `muletestbox` and the password `123456`.

**cache: Unexpected program error: java.lang.NullPointerException**

## Transformers

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description
email-to-string-transformer	Converts an email message to string format.
string-to-email-transformer	Converts a string message to email format.
object-to-mime-transformer	Converts an object to MIME format.
mime-to-bytes-transformer	Converts a MIME message to a byte array.

bytes-to-mime-transformer	Converts a byte array message to MIME format.
---------------------------	---

Here is how you define transformers in your Mule configuration file:

```
<email:bytes-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:email-to-string-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:mime-to-bytes-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:object-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" useInboundAttachments="true" useOutboundAttachments="true"/>
{Note}Need to explain attributes somewhere; can we pull them in from xsd?{Note}
<email:string-to-email-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
```

Each transformer supports all the common transformer attributes and properties:

cache: Unexpected program error: java.lang.NullPointerException

### Transformer

A reference to a transformer defined elsewhere.

#### Attributes of <transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[ ]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
ref	string	yes		The name of the transformer to use.

#### Child Elements of <transformer...>

Name	Cardinality	Description
------	-------------	-------------

The object-to-mime-transformer has the following attributes:

Attribute	Description	Default Value
useInboundAttachments	Whether to transform inbound attachment in the input message into MIME parts.	true
useOutboundAttachments	Whether to transform outbound attachment in the input message into MIME parts.	true

To use these transformers, make sure you include the 'email' namespace in your mule configuration.

Your Rating:  Results:  1 rates

## Filters

Filters can be set on an endpoint to filter out any unwanted messages. The Email transport provides a couple of filters that can either be used directly or extended to implement custom filtering rules.

Filter	Description
org.mule.providers.email.filters.AbstractMailFilter	A base filter implementation that must be extended by any other mail filter.
org.mule.providers.email.filters.MailSubjectRegExFilter	Applies a regular expression to a Mail Message subject.

This is how you define the MailSubjectRegExFilter in your Mule configuration:

```
<message-property-filter pattern="to=barney@mule.org" />
```

The 'pattern' attribute is a regular expression pattern. This is defined as java.util.regex.Pattern.

Your Rating:  Results:  1 rates

## Exchange patterns / features of the transport

(see [transport matrix](#))

## Schema Reference

You can view the full schema for the SMTP email transport [here](#). The secure version is [here](#).

## Java API Reference

The Javadoc for this transport can be found [here](#).

## Maven module

The email transports are implemented by the mule-transport-email module. You can find the source for the email transport under transports/email.

If you are using Maven to build your application, use the following dependency snippet to include the email transport in your project:

```
<dependency>
    <groupId>org.mule.transports</groupId>
    <artifactId>mule-transport-email</artifactId>
</dependency>
```

## Mule-Maven Dependencies

If you are building Mule ESB from source or including Mule artifacts in your Maven project, it may be necessary to add the 'mule-deps' repository to your Maven configuration. This repository contains third-party binaries which may not be in any other public Maven repository.

To add the 'mule-deps' repository to your Maven project, add the following to your pom.xml:

```
<repositories>
    <repository>
        <id>mule-deps</id>
        <name>Mule Dependencies</name>
        <url>http://dist.codehaus.org/mule/dependencies/maven2</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

Your Rating: Results:  4 rates

## Limitations

The following known limitations affect email transports:

- Retry policies do not work with email transports
- Timeouts are not supported in email transports
- Can't send same object to different email users
- MailSubjectRegExFilter cannot handle mails with attachments

Your Rating: Results:  2 rates

So far, all configuration has been static, in that you define all the information in the configuration of the endpoint. However, you can set the **connector properties** to control the settings of the outgoing message. These properties will override the endpoint properties. If you always want to set the email address dynamically, you can leave out the `to` attribute (or the `address` parameter if you're using URIs) on the SMTP endpoint.



### Escape Your Credentials

If you use a URI-style endpoint and you include the user name and password, escape any characters that are illegal for URIs, such as the @ character. For example, if the user name is `user@myco.com`, you should enter it as `user%40myco.com`.

Your Rating: Results:  0 rates

## File Transport Reference

### File Transport Reference

[ Introduction ] [ Transport Info ] [ Namespace and Syntax ] [ Considerations ] [ Features ] [ Usage ] [ Example Configurations ] [ Configuration Options ] [ Configuration Reference ] [ Associated Elements ] [ Schema ] [ Javadoc API Reference ] [ Maven ]

#### Introduction

The File transport allows files on the local file system to be read and written to. The connector can be configured to filter the file it reads and the way files are written, such as whether the output will be placed in a new file or if it should be appended.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
File	JavaDoc SchemaDoc							one-way	one-way	org.mule.transport:n

#### Legend

Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in Maven

### Namespace and Syntax

**XML namespace:**

```
xmlns:file="http://www.mulesoft.org/schema/mule/file"
```

**XML Schema location:**

```
http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
```

**Connector syntax:**

```
<!-- Typical Connector for Inbound Endpoint: Read files -->
<file:connector name="input" fileAge="500" autoDelete="true" pollingFrequency="100" moveToDirectory=
"/backup" moveToPattern="#[header:originalFilename].backup" />

<!-- Typical Connector for Outbound Endpoint: Write files -->
<file:connector name="output" outputAppend="true" outputPattern=
"#function:datestamp]-#[header:originalFilename]" />
```

**Endpoint syntax:**

File endpoints can be expressed using standard File URI syntax:

```
file://<path>[MULE:<params>]
```

For example, to connect to a directory called /temp/files -

**Unix**

```
file:///temp/files
```

Note the extra slash to denote a path from the root (absolute path).

**Windows**

```
file:///C:/temp/files
```

The Unix style will still work in Windows if you map to a single drive (the one Mule was started from).

To specify a relative path use:

```
file://./temp
```

or

```
file://temp
```

Note only two slashes are used for the protocol, so it's a relative path.

or

```
file://?address=../temp
```

To connect to a windows network drive:

```
file:///192.168.0.1/temp/
```

#### Inbound endpoint:

```
<file:inbound-endpoint connector-ref="input" path="/tmp/input" />
```

#### Outbound endpoint:

```
<file:outbound-endpoint connector-ref="output" path="/tmp/output" />
```

### Considerations

As it can be seen, Mule ESB provides lots of functionality ready to use that can just be modified by changing a XML file. Everyone knows how to handle files in their programming language, but when advanced features are required, coding gets complex. Mule ESB easily allows you to rename and archive files and handles the uncomfortable task of validating when input files are completely generated.

- This transport should be used to both read and write files in the filesystem. Use the inbound endpoint to read files every certain period of time, filtering input files by different name patterns and deleting, moving or leaving the file as it is once processed. The outbound endpoint allows you to generate new files (the file name can be defined in runtime) or to append content to an existing file.
- Take into account that the account running mule (in standalone mode, the user that launched the Mule ESB server, if not, the user which runs the Application Server) should have read and/or write permissions on the directories configured for this transport.
- Be careful not to permanently delete or overwrite input/output files. Be careful when using, for example, *autoDelete* and *moveToDirectory* attributes.
- Check the examples below and find out how to copy files from one directory to another, process input files while saving a backup of the input file and creating a new file with a specific name.
- Though most configuration parameters can be defined globally at in the connector, they can be overridden in the endpoint configuration.
- If streaming is enabled a *ReceiverFileInputStream* will be used as the payload for each file that is processed. This input stream's *close()* method takes care of moving the file or deleting it. Streams are closed by transformers reading the input stream. If you process the stream in your own component implementation make sure to properly close the stream after reading.

### Features

- Read files at a regular polling interval
- Write files

### Usage

To use the file transport in your Mule configuration, [import the file namespace](#) and use the `<file:connector>`, `<file:inbound-endpoint>` and/or `<file:outbound-endpoint>` elements. Refer to the [example configurations](#) below.

You will also be able to use the following expressions inside attributes (check [here](#) for more information):

- #[function:dateStamp]
- #[function:datesetamp:dd-MM-yy]
- #[function:systime]
- #[function:uuid]
- #[header:originalFilename]
- #[function:count]
- #[header:message property name]

### Example Configurations

#### Mule Flow

### Copying files in Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">

    <file:connector name="input" autoDelete="false" pollingFrequency="1000" />

    <file:connector name="output" outputAppend="false" />

    <flow name="copyFile">
        <file:inbound-endpoint connector-ref="input" path="/tmp/input"/>
        <file:outbound-endpoint connector-ref="output" path="/tmp/output"/>
    </flow>
</mule>
```

### Mule Service

### Copying files in Service

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">

    <file:connector name="input" autoDelete="false" pollingFrequency="1000" />

    <file:connector name="output" outputAppend="false" />

    <model>
        <service name="copyFile">
            <inbound>
                <file:inbound-endpoint connector-ref="input" path="/tmp/input"/>
            </inbound>
            <outbound>
                <pass-through-router>
                    <file:outbound-endpoint connector-ref="output" path="/tmp/output"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

This simple example copies files from `/tmp/input` to `/tmp/output` every 1 second (1000 ms). As input files are not deleted they are processed every time. Changing `autoDelete` to `true` will just move files.

## Mule Flow

### Moving files in Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">

    <file:connector name="input" autoDelete="true" fileAge="500" pollingFrequency="5000" />

    <file:connector name="output" outputAppend="false" />

    <flow name="moveFile">
        <file:inbound-endpoint connector-ref="input" path="/tmp/input"
            moveToDirectory="/tmp/backup"
            moveToPattern="#{header:originalFilename}.backup"/>
        <file:outbound-endpoint connector-ref="output" path="/tmp/output"
            outputPattern="#{function:datestamp}-#{header:originalFilename}"/>
    </flow>
</mule>
```

## Mule Service

## Moving files in Service

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">

    <file:connector name="input" autoDelete="true" fileAge="500" pollingFrequency="5000" />

    <file:connector name="output" outputAppend="false" />

    <model>
        <service name="moveFile">
            <inbound>
                <file:inbound-endpoint connector-ref="input" path="/tmp/input"
                                         moveToDirectory="/tmp/backup"
                                         moveToPattern="#{header:originalFilename}.backup"/>
            </inbound>
            <outbound>
                <pass-through-router>
                    <file:outbound-endpoint connector-ref="output" path="/tmp/output"
                                              outputPattern=
"#[function:datestamp]-#[header:originalFilename]"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

This example moves files from `/tmp/input` to `/tmp/output` every 5 second (5000 ms) , saving a backup file of the original file (with the extension `.backup`) in `/tmp/backup` . The new file is renamed with the current date and time as prefix . Note that `fileAge` will prevent moving files that are still being generated as the file has to be untouched for at least half second .

### Different connector configurations

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <file:connector name="sendConnector" outputAppend="true" outputPattern="[TARGET_FILE]" />

    <file:connector name="receiveConnector" fileAge="500" autoDelete="true" pollingFrequency="100" />

    <file:connector name="inboundFileConnector" pollingFrequency="10000"
        streaming="false" autoDelete="false">
        <service-overrides messageFactory="org.mule.transport.file.FileMuleMessageFactory"
            inboundTransformer="org.mule.transformer.NoActionTransformer" />
        <file:expression-filename-parser />
    </file:connector>

    <flow name="RefreshFileManager">
        <file:inbound-endpoint connector-ref="inboundFileConnector"
            path="C:/temp/filewatcher/inbox" moveToDirectory="C:/temp/filewatcher/history"
            moveToPattern="#{function:timestamp}-#[header:originalFilename]" />
        ...
    </flow>
    ...
</mule>

```

This last example shows different connector configurations. The third example overrides parts of the transport implementation and does not delete the file after processing it . The inbound endpoint moves it to a directory for archiving after it is processed .

## Configuration Options

### File Transport **inbound endpoint** attributes

Name	Description	Default
autoDelete	set this attribute to false if you don't want Mule to delete the file once processed	true
fileAge	setting this value (minimum age in milliseconds for a file to be processed) is useful when consuming large files, as Mule will wait before reading this file until the file last modification timestamp indicates that the file is older than this value	true
moveToDirectory	use this parameter if you want Mule to save a backup copy of the file it reads	
moveToPattern	use this parameter together with moveToPattern if you want to rename the copy of the backed up file	
pollingFrequency	the frequency in milliseconds that the read directory should be checked	0
recursive	use this parameter if Mule should recurse when a directory is read	false
streaming	if you want the payload to be a byte array instead of a FileInputStream set this parameter to false	true
workDirectory	if you require moving input files before they are processed by Mule, then assign a working directory (in the same file system) with this parameter	
workFileNamePattern	use this parameter together with workDirectory if you need to rename input files prior to processing them	

### File Transport **outbound endpoint** attributes

Name	Description	Default
------	-------------	---------

outputAppend	if the file to be written already exists, set this parameter to true to append new contents instead of overwriting the file	false
outputPattern	the pattern to use when writing a file to disk	

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Connector

The File connector configures the default behavior for File endpoints that reference the connector. If there is only one File connector configured, all file endpoints will use that connector.

#### Attributes of <connector...>

Name	Type	Required	Default	Description
writeToDirectory	string	no		The directory path where the file should be written on dispatch. This path is usually set as the endpoint of the dispatch event, however this allows you to explicitly force a single directory for the connector.
readFromDirectory	string	no		The directory path where the file should be read from. This path is usually set as the inbound endpoint, however this allows you to explicitly force a single directory for the connector.
autoDelete	boolean	no	true	If set to true (the default), it will cause the file to be deleted once it is read. If streaming is turned on, this occurs when the InputStream for the file is closed. Otherwise the file will be read into memory and deleted immediately. To access the java.io.File object set this property to false and specify a NoActionTransformer transformer for the connector. Mule will not delete the file, so it is up to the component to delete it when it is done. If the moveToDirectory is set, the file is first moved, then the File object of the moved file is passed to the component. It is recommended that a moveToDirectory is specified when turning autoDelete off.
outputAppend	boolean	no	false	Whether the output should be appended to the existing file. Default is false.
serialiseObjects	boolean	no		Determines whether objects should be serialized to the file. If false (the default), the raw bytes or text is written.
streaming	boolean	no	true	Whether a FileInputStream should be sent as the message payload (if true) or a byte array. (if false). The default is true.
workDirectory	string	no		(As of Mule 2.1.4) The directory path where the file should be moved to prior to processing. The work directory must reside on the same file system as the read directory.
workFileNamePattern	string	no		(As of Mule 2.1.4) The pattern to use when moving a file to a new location determined by the workDirectory property. You can use the patterns supported by the filename parser configured for this connector.
recursive	boolean	no	false	Whether to recurse or not when a directory is read
pollingFrequency	long	no		The frequency in milliseconds that the read directory should be checked (default is 0). Note that the read directory is specified by the endpoint of the listening component.
fileAge	long	no		Minium age (ms) for a file to be processed. This can be useful when consuming large files. It tells Mule to wait for a period of time before consuming the file, allowing the file to be completely written before the file is processed.
moveToPattern	string	no		The pattern to use when moving a read file to a new location determined by the moveToDirectory property. This can use the patterns supported by the filename parser configured for this connector.
moveToDirectory	string	no		The directory path where the file should be written after it has been read. If this is not set, the file is deleted after it has been read.
comparator	class name	no		Sorts incoming files using the specified comparator, such as comparator="org.mule.transport.file.comparator.OlderFirstComparator". The class must implement the java.util.Comparator interface.
reverseOrder	boolean	no		Whether the comparator order should be reversed. Default is false.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename parser configured for this connector.

#### Child Elements of <connector...>

Name	Cardinality	Description
abstract-filenameParser	0..1	The abstract-filenameParser element is a placeholder for filename parser elements. The filename parser is set on the connector used when writing files to a directory. The parser will convert the outputPattern attribute to a string using the parser and the current message. The default implementation used is expression-filename-parser, but you can also specify a custom-filename-parser.

#### Associated Elements

cache: Unexpected program error: java.lang.NullPointerException

#### Endpoint

##### Attributes of <endpoint...>

Name	Type	Required	Default	Description
path	string	no		A file directory location.
pollingFrequency	long	no		The frequency in milliseconds that the read directory should be checked (default is 0). Note that the read directory is specified by the endpoint of the listening component.
fileAge	long	no		Minimum age (ms) for a file to be processed. This can be useful when consuming large files. It tells Mule to wait for a period of time before consuming the file, allowing the file to be completely written before the file is processed.
moveToPattern	string	no		The pattern to use when moving a read file to a new location determined by the moveToDirectory property. This can use the patterns supported by the filename parser configured for this connector.
moveToDirectory	string	no		The directory path where the file should be written after it has been read. If this is not set, the file is deleted after it has been read.
comparator	class name	no		Sorts incoming files using the specified comparator, such as comparator="org.mule.transport.file.comparator.OlderFirstComparator". The class must implement the java.util.Comparator interface.
reverseOrder	boolean	no		Whether the comparator order should be reversed. Default is false.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename parser configured for this connector.

#### Child Elements of <endpoint...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Inbound endpoint

##### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
path	string	no		A file directory location.
pollingFrequency	long	no		The frequency in milliseconds that the read directory should be checked (default is 0). Note that the read directory is specified by the endpoint of the listening component.
fileAge	long	no		Minimum age (ms) for a file to be processed. This can be useful when consuming large files. It tells Mule to wait for a period of time before consuming the file, allowing the file to be completely written before the file is processed.
moveToPattern	string	no		The pattern to use when moving a read file to a new location determined by the moveToDirectory property. This can use the patterns supported by the filename parser configured for this connector.

moveToDirectory	string	no		The directory path where the file should be written after it has been read. If this is not set, the file is deleted after it has been read.
comparator	class name	no		Sorts incoming files using the specified comparator, such as comparator="org.mule.transport.file.comparator.OlderFirstComparator". The class must implement the java.util.Comparator interface.
reverseOrder	boolean	no		Whether the comparator order should be reversed. Default is false.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Outbound endpoint**

#### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
path	string	no		A file directory location.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename parser configured for this connector.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **File to byte array transformer**

The file-to-byte-array-transformer element configures a transformer that reads the contents of a java.io.File into a byte array (byte[]).

#### Child Elements of <file-to-byte-array-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **File to string transformer**

The file-to-string-transformer element configures a transformer that reads the contents of a java.io.File into a java.lang.String.

#### Child Elements of <file-to-string-transformer...>

Name	Cardinality	Description
------	-------------	-------------

Note that this transformer does not close file streams. This prevents files from being deleted or moved if the flow is asynchronous. If you have streaming enabled for an asynchronous endpoint, use the ObjectToString transformer instead.

cache: Unexpected program error: java.lang.NullPointerException

#### **Filename wildcard filter**

The filename-wildcard-filter element configures a filter that can be used to restrict the files being processed by applying wildcard expressions to the filename. For example, you can read only .xml and .txt files by entering the following: <file:filename-wildcard-filter pattern=".txt,.xml"/>

#### Child Elements of <filename-wildcard-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

## **Filename regex filter**

The filename-regex-filter element configures a filter that can be used to restrict the files being processed by applying Java regular expressions to the filename, such as pattern="myCustomerFile(.\*)".

### **Child Elements of <filename-regex-filter...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

cache: Unexpected program error: java.lang.NullPointerException

## **Expression filename parser**

The expression-filename-parser element configures the ExpressionFilenameParser, which can use any expression language supported by Mule to construct a file name for the current message. Expressions can be xpath, xquery, ognl, mvel, header, function, and more.

### **Attributes of <expression-filename-parser...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### **Child Elements of <expression-filename-parser...>**

Name	Cardinality	Description
------	-------------	-------------

For example, an XPath expression can be defined to pull a message ID out of an XML message and use that as the file name as follows:

```
# [xpath:/message/header/@id]
```

Following is an example of using the parser:

```
<file:connector name="FileConnector" >
  <file:expression-filename-parser/>
</file:connector>
...
<file:outbound-endpoint path="file://temp"
outputPattern="#[header:originalFilename]--#[function:datestamp].txt"/>
```

This parser supersedes <legacy-filename-parser> from previous releases of Mule. The following demonstrates how to achieve the same results when using <expression-filename-parser> over <legacy-filename-parser>.

- #[DATE] : #[function:dateStamp]
- #[DATE:dd-MM-yy] : #[function:datestamp:dd-MM-yy]
- #[SYSTIME] : #[function:systime]
- #[UUID] : #[function:uuid]
- #[ORIGINALNAME] : #[header:originalFilename]
- #[COUNT] : #[function:count] - note that this is a global counter. If you want a local counter per file connector then you should use the legacy-filename-parser.
- #[message property name] : #[header:message property name]

cache: Unexpected program error: java.lang.NullPointerException

## **Custom filename parser**

The custom-filename-parser element allows the user to specify a custom filename parser. The implementation must implement org.mule.transport.file.FilenameParser.

### **Attributes of <custom-filename-parser...>**

Name	Type	Required	Default	Description
class	string	yes		The implementation class name that implements org.mule.transport.file.FilenameParser.

#### Child Elements of <custom-filename-parser...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### **Abstract filenameParser**

The abstract-filenameParser element is a placeholder for filename parser elements. The filename parser is set on the connector used when writing files to a directory. The parser will convert the outputPattern attribute to a string using the parser and the current message. The default implementation used is expression-filename-parser, but you can also specify a custom-filename-parser.

#### Attributes of <abstract-filenameParser...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

#### Child Elements of <abstract-filenameParser...>

Name	Cardinality	Description
------	-------------	-------------

#### Schema

The schema for the file transport appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/file"

Targeting Schemas (1):

[mule-file.xsd](#)

Targeting Components:

[11 global elements, 7 complexTypes, 3 attribute groups](#)

Schema Summary	
mule-file.xsd	<p>The File transport allows files to be read and written to and from directories on the local file system.</p> <p>Target Namespace:</p> <p><a href="http://www.mulesoft.org/schema/mule/file">http://www.mulesoft.org/schema/mule/file</a></p> <p>Defined Components:</p> <p>11 global elements, 7 complexTypes, 3 attribute groups</p> <p>Default Namespace-Qualified Form:</p> <p>Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:</p> <p><a href="http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd">http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (3):</p> <p><a href="#">mule-schemadoc.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p> <p>Imported by Schema:</p> <p><a href="#">mule-ftp.xsd</a></p>

#### All Element Summary

abstract-filenameParser	<p>The abstract-filenameParser element is a placeholder for filename parser elements.</p> <p>Type: <a href="#">filenameParserType</a></p> <p>Content: empty</p> <p>Abstract: (may not be used directly in instance XML documents)</p> <p>Subst.Gr:may be substituted with 2 elements</p> <p>Defined: globally in <a href="#">mule-file.xsd</a>; see <a href="#">XML source</a></p> <p>Used: at 4 locations</p>
-------------------------	--

connector	The File connector configures the default behavior for File endpoints that reference the connector. Type: <a href="#">fileConnectorType</a> Content: complex, 22 attributes, attr. wildcard, 6 elements Subst.Gr:may substitute for element <code>mule:abstract-connector</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
custom-filename-parser	The custom-filename-parser element allows the user to specify a custom filename parser. Type: <a href="#">customFilenameParserType</a> Content: empty, 1 attribute Subst.Gr:may substitute for element <code>abstract-filenameParser</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
endpoint	Type: <a href="#">globalEndpointType</a> Content: complex, 18 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element <code>mule:abstract-global-endpoint</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
expression-filename-parser	The expression-filename-parser element configures the ExpressionFilenameParser, which can use any expression language supported by Mule to construct a file name for the current message. Type: <a href="#">expressionFilenameParserType</a> Content: empty Subst.Gr:may substitute for element <code>abstract-filenameParser</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
file-to-byte-array-transformer	The file-to-byte-array-transformer element configures a transformer that reads the contents of a <code>java.io.File</code> into a byte array ( <code>byte[]</code> ). Type: <a href="#">mule:abstractTransformerType</a> Content: empty, 5 attributes, attr. wildcard Subst.Gr:may substitute for elements: <code>mule:abstract-transformer</code> , <code>mule:abstract-message-processor</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
file-to-string-transformer	The file-to-string-transformer element configures a transformer that reads the contents of a <code>java.io.File</code> into a <code>java.lang.String</code> . Type: <a href="#">mule:abstractTransformerType</a> Content: empty, 5 attributes, attr. wildcard Subst.Gr:may substitute for elements: <code>mule:abstract-transformer</code> , <code>mule:abstract-message-processor</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
filename-regex-filter	The filename-regex-filter element configures a filter that can be used to restrict the files being processed by applying Java regular expressions to the filename, such as <code>pattern="myCustomerFile(.*)"</code> . Type: <a href="#">mule:wildcardFilterType</a> Content: empty, 3 attributes, attr. wildcard Subst.Gr:may substitute for elements: <code>mule:abstract-filter</code> , <code>mule:abstract-message-processor</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
filename-wildcard-filter	The filename-wildcard-filter element configures a filter that can be used to restrict the files being processed by applying wildcard expressions to the filename. Type: <a href="#">mule:wildcardFilterType</a> Content: empty, 3 attributes, attr. wildcard Subst.Gr:may substitute for elements: <code>mule:abstract-filter</code> , <code>mule:abstract-message-processor</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
inbound-endpoint	Type: <a href="#">inboundEndpointType</a> Content: complex, 17 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element <code>mule:abstract-inbound-endpoint</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
outbound-endpoint	Type: <a href="#">outboundEndpointType</a> Content: complex, 12 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element <code>mule:abstract-outbound-endpoint</code> Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Used: never
Complex Type Summary	
customFilenameParserType	Content: empty, 1 attribute Defined: globally in <a href="#">mule-file.xsd</a> ; see XML source Includes:definition of 1 attribute

	Used: at 1 location
expressionFilenameParserType	Content:empty Defined:globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
fileConnectorType	Content: complex, 22 attributes, attr. wildcard, 6 elements Defined: globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 9 attributes, 1 element Used: at 1 location
filenameParserType	The filenameParser is used when writing files to a directory. Content:empty Defined:globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Used: at 3 locations
globalEndpointType	Content:complex, 18 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
inboundEndpointType	Content:complex, 17 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
outboundEndpointType	Content:complex, 12 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location

#### Attribute Group Summary

addressAttributes	Content: 1 attribute Defined: globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 3 locations
inboundAttributes	Content: 6 attributes Defined: globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 6 attributes Used: at 3 locations
outboundAttributes	Content: 1 attribute Defined: globally in <a href="#">mule-file.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 3 locations

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this transport can be found here: [File](#).

## Maven

The File Transport can be included with the following dependency:

```
<dependency>
<groupId>org.mule.transports</groupId>
<artifactId>mule-transport-file</artifactId>
</dependency>
```

## Extending this Module or Transport

### Best Practices

If reading input files which are generated directly in the input path, configure the `fileAge` attribute in the connector or endpoint. In this way, Mule will process these files once they are completely written to disk.

### Notes

## FTP Transport Reference

### FTP Transport Reference

[ Introduction ] [ Transport Info ] [ Namespace and Syntax ] [ Considerations ] [ Features ] [ Usage ] [ Example Configurations ] [ Configuration Options ] [ Configuration Reference ] [ FTP Transport ] [ Schema ] [ Javadoc API Reference ] [ Maven ] [ Extending this Module or Transport ] [ Best Practices ] [ Notes ]

#### Introduction

The FTP transport allows integration of the File Transfer Protocol into Mule. Mule can poll a remote FTP server directory, retrieve files and process them as Mule messages. Messages can also be uploaded as files to a directory on a remote FTP server.

Mule also supports the SFTP protocol for secure file transfer. As of Mule 3, the [SFTP Transport](#) is included in the Mule distribution.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
FTP	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>							one-way	one-way	org.mule.transport:n

#### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

#### Namespace and Syntax

Namespace (Community edition)

`http://www.mulesoft.org/schema/mule/ftp`

XML schema location (Community Edition)

`http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd`

Namespace (Enterprise edition) 

`http://www.mulesoft.org/schema/mule/ee/ftp`



```
http://www.mulesoft.org/schema/mule/ee/ftp/3.1/mule-ftp-ee.xsd
```

#### Syntax:

straight URI example `ftp://theUser:secret@theHost:port/path`

XML version `<ftp:endpoint host="theHost" port="22" path="/path" user="theUser" password="secret"/>`



#### Escape Your Credentials

If you use a URI-style endpoint and you include the user name and password, escape any characters that are illegal for URIs, such as the @ character. For example, if the user name is `user@myco.com`, you should enter it as `user%40myco.com`.

Connector and endpoint syntax `<ftp:connector name="ftpConnector" passive="true" binary="true" streaming="true"/>`

## Considerations

### Features

- Poll a directory on a remote FTP server for new files
- Retrieve files an FTP server
- Transfer binary or text files
- Filter files at the endpoint based on filename wildcards
- Filter files at the endpoint based on Mule expressions
- Upload and store files on an FTP server
- Rename output files based on Mule expressions
- Streaming for transferring large files
- Support for [reconnection strategies](#)

The Mule Enterprise Edition includes several additional features that allow to filter files to be processed by file age and moving and renaming files on the source FTP server after processing.

### Usage

Each endpoint carries all the information for the FTP connection, i.e. host, port, path, username and password at least. Additional properties (like binary or passive) can be specified on the connector and overridden at the endpoint level.

The FTP transport periodically polls the FTP server. Upon each poll request, a new connection to the FTP server is opened, the specified user is logged in and all files are listed under the specified path. This means that if the FTP server goes down no special provisions need to be made - the current poll attempt will fail but polling will not be stopped.

If [reconnection strategies](#) are configured, the FTP connection can be re-established automatically by Mule based on the policy you have configured.

The FTP transport does not support transactions as the File Transfer Protocol itself is not transactional. Instead you should design compensating transactions into your architecture using [exception strategies](#) in Mule.

## Example Configurations

### Mule Flow

### Downloading files from FTP using a Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ftp="http://www.mulesoft.org/schema/mule/ftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/ftp
          http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd">

    <flow name="ftp2file">
        <ftp:inbound-endpoint host="localhost" port="21" path="/" user="theUser" password="secret"/>
        <file:outbound-endpoint path="/some/directory" outputPattern="#{header:originalFilename}"/>
    </flow>
</mule>
```

### Mule Service

### Downloading files from FTP using a Service

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ftp="http://www.mulesoft.org/schema/mule/ftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/ftp
          http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd">

    <model name="example">
        <service name="ftp2file">
            <inbound>
                <ftp:inbound-endpoint host="localhost" port="21" path="/" user="theUser" password="secret"/>
            </inbound>
            <outbound>
                <pass-through-router>
                    <file:outbound-endpoint path="/some/directory" outputPattern="#{header:originalFilename}"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

This example shows a simple flow that picks up all available files on the FTP server (in its root directory) and stores them into a directory on the local filesystem.

### Mule Flow

### Filtering filenames using a Flow

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ftp="http://www.mulesoft.org/schema/mule/ftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/ftp
          http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd">

    <flow name="fileFilter">
        <ftp:inbound-endpoint host="localhost" port="21" path="/" user="theUser" password="secret"/>
            <file:filename-wildcard-filter pattern=".txt,.xml"/>
        </ftp:endpoint>
        <file:outbound-endpoint path="/some/directory" outputPattern="#{header:originalFilename}"/>
    </flow>
</mule>
```

### Mule Service

### Filtering filenames using a Service

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ftp="http://www.mulesoft.org/schema/mule/ftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/ftp
          http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd">

    <model name="example">
        <service>
            <inbound>
                <ftp:inbound-endpoint host="localhost" port="21" path="/" user="theUser" password="secret"/>
                    <file:filename-wildcard-filter pattern=".txt,.xml"/>
                </ftp:endpoint>
            </inbound>
            <outbound>
                <pass-through-router>
                    <file:outbound-endpoint path="/some/directory" outputPattern="#{header:originalFilename}"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

This example shows how to pick only certain files on the FTP server. You do this by configuring filename filters to control which files the endpoint receives. The filters are expressed in a comma-separated list. Note that in order to use a filter from the file transport's schema it must be included.

## Processing a file from FTP

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:ftp="http://www.mulesoft.org/schema/mule/ftp"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/ftp
          http://www.mulesoft.org/schema/mule/ftp/3.1/mule-ftp.xsd">

    <simple-service name="ftpProcessor"
        address="ftp://theUser:secret@host:21/"
        component-class="com.mycompany.mule.MyProcessingComponent"/>
</mule>
```

This example uses a `simple-service` to route files retrieved from the FTP server to `MyProcessingComponent` for further processing.

## Configuration Options

### Streaming

If streaming is not enabled on the FTP connector, Mule will attempt to read a file it picks up from the FTP server into a `byte[]` to be used as the payload of the `MuleMessage`. This behaviour can cause trouble if large files need to be processed.

In this case, enable streaming on the connector:

```
<ftp:connector name="ftpConnector" streaming="true">
```

Instead of reading the file's content into memory Mule will now send an `InputStream` as the payload of the `MuleMessage`. The name of the file that this input stream represents is stored as the `originalFilename` property on the message. If streaming is used on inbound endpoints it is the responsibility of the user to close the input stream. If streaming is used on outbound endpoints Mule closes the stream automatically.

## Configuration Reference

### Element Listing

cache: Unexpected program error: java.lang.NullPointerException

## FTP Transport

The FTP transport provides connectivity to FTP servers to allow files to be read and written as messages in Mule.

### Connector

The FTP connector is used to configure the default behavior for FTP endpoints that reference the connector. If there is only one FTP connector configured, all FTP endpoints will use that connector.

#### Attributes of `<connector...>`

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the <code>MuleContext</code> , and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.

validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
streaming	boolean	no		Whether an InputStream should be sent as the message payload (if true) or a byte array (if false). Default is false.
connectionFactoryClass	class name	no		A class that extends FtpConnectionFactory. The FtpConnectionFactory is responsible for creating a connection to the server using the credentials provided by the endpoint. The default implementation supplied with Mule uses the Commons Net project from Apache.
pollingFrequency	long	no		How frequently in milliseconds to check the read directory. Note that the read directory is specified by the endpoint of the listening component.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename-parser configured for this connector
binary	boolean	no		Select/disable binary file transfer type. Default is true.
passive	boolean	no		Select/disable passive protocol (more likely to work through firewalls). Default is true.

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
file:abstract-filenameParser	0..1	The filenameParser is used when writing files to an FTP server. The parser will convert the outputPattern attribute to a string using the parser and the current message. To add a parser to your configuration, you import the "file" namespace into your XML configuration. For more information about filenameParsers, see the <a href="#">File Transport Reference</a> .

#### Inbound endpoint

##### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.

responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
path	string	no		A file location on the remote server.
user	string	no		If FTP is authenticated, this is the username used for authentication.
password	string	no		The password for the user being authenticated.
host	string	no		An IP address (such as www.mulesoft.com, localhost, or 192.168.0.1).
port	port number	no		The port number to connect on.
binary	boolean	no		Select/disable binary file transfer type. Default is true.
passive	boolean	no		Select/disable passive protocol (more likely to work through firewalls). Default is true.
pollingFrequency	long	no		How frequently in milliseconds to check the read directory. Note that the read directory is specified by the endpoint of the listening component.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.

properties	0..1	A map of Mule properties.
------------	------	---------------------------

## Outbound endpoint

Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
path	string	no		A file location on the remote server.
user	string	no		If FTP is authenticated, this is the username used for authentication.
password	string	no		The password for the user being authenticated.
host	string	no		An IP address (such as www.mulesoft.com, localhost, or 192.168.0.1).
port	port number	no		The port number to connect on.
binary	boolean	no		Select/disable binary file transfer type. Default is true.
passive	boolean	no		Select/disable passive protocol (more likely to work through firewalls). Default is true.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename-parser configured for this connector

Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.

abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

## Endpoint

### Attributes of <endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mime-type	string	no		The mime type, e.g. text/plain or application/json

path	string	no		A file location on the remote server.
user	string	no		If FTP is authenticated, this is the username used for authentication.
password	string	no		The password for the user being authenticated.
host	string	no		An IP address (such as www.mulesoft.com, localhost, or 192.168.0.1).
port	port number	no		The port number to connect on.
binary	boolean	no		Select/disable binary file transfer type. Default is true.
passive	boolean	no		Select/disable passive protocol (more likely to work through firewalls). Default is true.
pollingFrequency	long	no		How frequently in milliseconds to check the read directory. Note that the read directory is specified by the endpoint of the listening component.
outputPattern	string	no		The pattern to use when writing a file to disk. This can use the patterns supported by the filename-parser configured for this connector

#### Child Elements of <endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

#### Mule Enterprise Connector Attributes

The following additional attributes are available on the FTP connector in Mule Enterprise only:

moveToDirectory	The directory path where the file should be written after it has been read. If this property is not set, the file is deleted.
moveToPattern	The pattern to use when moving a read file to a new location as specified by the moveToDirectory property. This property can use the patterns supported by the filenameParser configured for this connector.
fileAge	Do not process the file unless it's older than the specified age in milliseconds.

## Schema

Complete schema reference documentation.

## Javadoc API Reference

Javadoc for this transport can be found [here](#).

## Maven

The FTP transport can be included with the following dependency:

Community edition

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-ftp</artifactId>
  <version>3.1.0</version>
</dependency>
```

Enterprise edition 

```
<dependency>
  <groupId>com.mulesoft.muleesb.transports</groupId>
  <artifactId>mule-transport-ftp-ee</artifactId>
  <version>3.1.0</version>
</dependency>
```

## Extending this Module or Transport

### *Custom FtpConnectionFactory*

The `FtpConnectionFactory` establishes Mule's connection to the FTP server. The default connection factory should be sufficient in 99% of the cases. If you need to change the way Mule connects to your FTP server use the `connectionFactoryClass` attribute on the connector:

```
<ftp:connector name="ftpConnector" connectionFactoryClass="com.mycompany.mule.MyFtpConnectionFactory"
/>
```

Use the fully qualified class name of your `FtpConnectionFactory` subclass. Note that this **must** be a subclass of `FtpConnectionFactory` as the `FtpConnector` attempts to cast the factory to that class.

### *Filename parser*

The `filenameParser` is used when writing files to the FTP server. The parser will convert the output pattern configured on an endpoint to the name of the file that will be written using the parser and the current message.

The filename parser used in the FTP transport should be sufficient in 99% of the cases. It is an instance of `ExpressionFilenameParser` which allows to use `arbitrary expressions` to compose the filename that is used when storing files on the FTP server.

It is possible to configure a custom filename parser as a child element of the connector declaration:

```
<ftp:connector name="ftpConnector" passive="true" binary="true" streaming="true">
  <file:custom-filename-parser class="com.mycompany.mule.MyFilenameParser"/>
</ftp:connector>
```

Note that the class you configure here must implement the `FilenameParser` interface.

## Best Practices

Put your login credentials in a properties file, not hard-coded in the configuration. This also allows you to use different settings between development, test and production systems.

## Notes

Your Rating: 

Results:  2 rates

## HTTPS Transport Reference

### HTTPS Transport Reference

[ [HTTPS Connector](#) ] [ [Setting up a HTTPS Server](#) ] [ [Configuration Reference](#) ] [ [Polling Connector](#) ] [ [HTTPS Endpoints](#) ]

The Secure HTTP transport provides support for exposing services over HTTP and making HTTP client requests from Mule services to external services as part of service event flows. Mule supports secure inbound, secure outbound, and secure polling HTTP endpoints. These endpoints support all common features of the HTTP spec, such as ETag processing, cookies, and keepalive. Both HTTP 1.0 and 1.1 are supported.

### HTTPS Connector

This connector provides Secure HTTP connectivity on top of what is already provided with the Mule [HTTP Transport](#). Secure connections are made on behalf of an entity, which can be anonymous or identified by a certificate. The *key store* provides the certificates and associated private keys necessary for identifying the entity making the connection. Additionally, connections are made to trusted systems. The public certificates of trusted systems are stored in a *trust store*, which is used to verify that the connection made to a remote system matches the expected identity.

### Setting up a HTTPS Server

In order to setup a HTTPS server with Mule a few first steps need to be performed. First a keystore must be created, this can be done using the keytool provided by Java. You can find this in the bin directory of the Java installation. Once located you can then execute the following command to create a keystore:

```
keytool -genkey -alias mule -keyalg RSA -keystore keystore.jks
```

This will create a file in the local directory called keystore.jks. Ideally this should be created in the MULE\_HOME/conf directory if to be used across multiple applications or can be put into the <MY MULE APP>/src/main/resources directory if being used within a single application.

Once the keystore is in place the following needs to be added to your mule configuration file:

```
<https:connector name="httpsConnector">
  <https:tls-key-store path="keystore.jks" keyPassword="" storePassword="
```

If the keystore was in the <MY MULE APP>/src/main/resources directory then you can just specify the name in the path. Otherwise if the keystore was located in the MULE\_HOME/conf directory then you will have to specify "\${mule.home}/conf/keystore.jks" as the path.

### Configuration Reference

Property	Description
tls-client	Configures the client key store with the following attributes: <ul style="list-style-type: none"><li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the keystore that contains public certificates and private keys for identification</li><li>storePassword: The password used to protect the keystore</li><li>class: The type of keystore used (a Java class name)</li></ul>

<code>tls-key-store</code>	Configures the direct key store with the following attributes: <ul style="list-style-type: none"> <li>• path: The location (which will be resolved relative to the current classpath and file system, if possible) of the keystore that contains public certificates and private keys for identification</li> <li>• class: The type of keystore used (a Java class name)</li> <li>• keyPassword: The password used to protect the private key</li> <li>• storePassword: The password used to protect the keystore</li> <li>• algorithm: The algorithm used by the keystore</li> </ul>
<code>tls-server</code>	Configures the trust store. The attributes are: <ul style="list-style-type: none"> <li>• path: The location (which will be resolved relative to the current classpath and file system, if possible) of the trust store that contains public certificates of trusted servers</li> <li>• storePassword: The password used to protect the trust store</li> <li>• class: The type of trust store used (a Java class name)</li> <li>• algorithm: The algorithm used by the trust store</li> <li>• factory-ref: Reference to the trust manager factory</li> <li>• explicitOnly: Whether this is an explicit trust store</li> <li>• requireClientAuthentication: Whether client authentication is required</li> </ul>
<code>tls-protocol-handler</code>	Configures the global Java protocol handler. It has one attribute, <code>property</code> , which specifies the <code>java.protocol.handler.pkgs</code> system property.

For example:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:beans="http://www.springframework.org/schema/beans"
      xmlns:https="http://www.mulesoft.org/schema/mule/https"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/https
          http://www.mulesoft.org/schema/mule/https/3.0/mule-https.xsd">

    <https:connector name="httpConnector">
        <https:client path="clientKeystore" storePassword="mulepassword"/>
        <https:tls-key-store path="serverKeystore" keyPassword="mulepassword" storePassword="mulepassword"/>
        <https:tls-server path="trustStore" storePassword="mulepassword"/>
    </https:connector>

    <https:endpoint name="clientEndpoint" host="localhost" port="60202"
                    connector-ref="httpConnector" />
</mule>
```

## Polling Connector

The polling connector allows Mule to poll an external HTTP server and generate events from the result. This is useful for pull-only web services. This connector provides a secure version of the `PollingHttpConnector`. It includes all the properties of the HTTPS connector plus the following optional attributes:

Attribute	Description
<code>pollingFrequency</code>	The time in milliseconds to wait between each request to the remote http server.
<code>checkEtag</code>	Whether the ETag header from the remote server is processed if the header is present.
<code>discardEmptyContent</code>	Whether Mule should discard any messages from the remote server that have a zero content length. For many services, a zero length would mean there was no data to return. If the remote HTTP server does return content to say that the request is empty, users can configure a content filter on the endpoint to filter these messages out.

For example, after defining the HTTP namespace in the header, you could configure the polling connector like this:

```
<http:polling-connector name="PollingHttpConnector" pollingFrequency="2000" />
```

## HTTPS Endpoints

An inbound HTTPS endpoint exposes a service securely over HTTPS, essentially making it an HTTP server. If polling of a remote HTTP service is required, this endpoint should be configured with a polling HTTPS connector.

An outbound HTTPS endpoint allows Mule to send requests securely using SSL to external servers or Mule inbound HTTP endpoints using HTTP over SSL protocol.

A global HTTPS endpoint can be referenced by services. Services can augment the configuration defined in the global endpoint with local configuration elements.

For more information on configuring HTTP endpoints, see [HTTP Transport Reference](#).

Your Rating:  Results:  0 rates

## HTTP Transport Reference

### HTTP Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Features](#) ] [ [Basic Usage](#) ] [ [Security](#) ] [ [Cookies](#) ] [ [Polling HTTP Services](#) ] [ [Handling HTTP Content-Type and Encoding](#) ] [ [Including Custom Header Properties](#) ] [ [Building the Target URL from the Request](#) ] [ [Handling Redirects](#) ] [ [Getting a Hash Map of POST body params](#) ] [ [Processing GET Query Parameters](#) ] [ [Serving Static Content \(since Mule 3.2\)](#) ] [ [Examples](#) ] [ [Configuration Reference](#) ] [ [Connector](#) ] [ [Polling Connector](#) ] [ [Rest Service Component](#) ] [ [Inbound endpoint](#) ] [ [Outbound endpoint](#) ] [ [Endpoint](#) ] [ [Request wildcard filter](#) ]

### Introduction

The HTTP transport provides support for exposing services over HTTP and making HTTP client requests from Mule services to external services as part of service event flows. Mule supports inbound, outbound, and polling HTTP endpoints. These endpoints support all common features of the HTTP spec, such as ETag processing, cookies, and keepalive. Both HTTP 1.0 and 1.1 are supported.

HTTP/S endpoints are synchronous by default, so you do not have to set `exchange-pattern="request-response"`. If you set `exchange-pattern="one-way"`, the messages are sent asynchronously. Note that if you're doing an asynchronous POST, streaming is disabled.

### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	M
HTTP	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>							one-way, request-response	<a href="#">request-response</a>	or
HTTPS	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>							one-way, request-response	<a href="#">request-response</a>	or

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

## Namespace and Syntax

```
http://www.mulesoft.org/schema/mule/http
```

XML schema location

```
http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
```

## Syntax

### URI example

```
http://theUser:secret@theHost:port/path?query
```

### XML version

```
<http:endpoint host="theHost" port="8080" path="/path" user="theUser"  
password="secret"/>
```



#### Escape Your Credentials

If you use a URI-style endpoint and you include the user name and password, escape any characters that are illegal for URIs, such as the @ character. For example, if the user name is user@myco.com, you should enter it as user%40myco.com.

## Features

- Server as an HTTP server or client
- Create HTTP services such as SOAP, REST or XML-RPC
- Make HTTP requests to external services
- Support for polling an HTTP endpoints including ETag support
- Transfer binary or text files, including forms and attachment
- Security including SSL, certificates and Authentication
- Build flows to different paths on the same port
- Support for reading and writing cookies
- Streaming for transferring large files
- Custom HTTP header support
- Redirect Handling
- Handling of Content-Type and Encoding
- Serve up static content such as HTML, JavaScript, Images and CSS (since Mule 3.2)

## Basic Usage

To create a HTTP server you just need to create a flow with an inbound HTTP endpoint:

```
<flow name="testComponent">  
    <http:inbound-endpoint name="clientEndpoint" address="http://localhost:8080"  
        path="foo" />  
    <echo-component>  
</flow>
```

This will accept incoming HTTP requests on `http://localhost:8080/foo` and echo the response back to the client.

To make a client invocation of an HTTP endpoint you need to configure an outbound endpoint on your flow or you can use the `MuleClient` to invoke an HTTP endpoint directly in your code.

```

<flow name="OutboundDelete">
    <vm:inbound-endpoint path="doDelete"
        exchange-pattern="one-way" />

    <http:outbound-endpoint host="localhost" port="8080" path="foo"
        method="DELETE" exchange-pattern="one-way" />

</flow>

```

Or from within your code:

```

MuleClient client = muleContext.getClient();
MuleMessage result = client.send("http://localhost:8080/foo", "");

```

Finally, you can reference an endpoint by name from your Mule configuration in the Mule client. From the previous example we can create a global HTTP endpoint that can be referenced from the flow or from code:

```

<http:endpoint name="deleteEndpoint" host="localhost" port="8080" path="foo"
    method="DELETE" exchange-pattern="one-way" />
<flow name="OutboundDelete">
    <vm:inbound-endpoint path="doDelete" exchange-pattern="one-way" />

    <http:outbound-endpoint ref="deleteEndpoint" />
</flow>

```

```

MuleClient client = muleContext.getClient();
MuleMessage result = client.send("deleteEndpoint", "");

```

Global endpoints allow you to remove actual addresses from your code and flows so that you can move Mule applications between environments.

## Security

You can use the [HTTPS Transport Reference](#) to create secure connections over HTTP. If you want to secure requests to your HTTP endpoint, the HTTP connector supports HTTP Basic/Digest authentication methods (as well as the Mule generic header authentication). To configure HTTP Basic, you configure a [Security Endpoint Filter](#) on an HTTP endpoint.

```

<inbound-endpoint address="http://localhost:4567">
    <spring-sec:http-security-filter realm="mule-realm" />
</inbound-endpoint>

```

You must configure the security manager on the Mule instance against which this security filter will authenticate. For information about security configuration options and examples, see [Configuring Security](#).

If you want to make an HTTP request that requires authentication, you can set the credentials on the endpoint:

```

<inbound-endpoint address="user:password@mycompany.com/secure">
    <spring-sec:http-security-filter realm="mule-realm" />
</inbound-endpoint>

```

For general information about endpoint configuration, see [Configuring Endpoints](#).

## Sending Credentials

If you want to make an HTTP request that requires authentication, you can set the credentials on the endpoint:

```
http://user:password@mycompany.com/secure
```

## Cookies

If you want to send cookies along on your outgoing request, simply configure them on the endpoint:

```
<http:outbound-endpoint address="http://localhost:8080" method="POST">
  <properties>
    <spring:entry key="Content-Type" value="text/xml" />
    <spring:entry key="cookies">
      <spring:map>
        <spring:entry key="customCookie" value="yes" />
      </spring:map>
    </spring:entry>
  </properties>
</http:outbound-endpoint>
```

## Polling HTTP Services

The HTTP transport supports polling an HTTP URL, which is useful for grabbing periodic data from a page that changes or to invoke a REST service, such as polling an [Amazon Queue](#).

To configure the HTTP Polling receiver, you include an HTTP polling-connector configuration in your Mule configuration:

```
<http:polling-connector name="PollingHttpConnector" pollingFrequency="30000"
  reuseAddress="true" />
```

To use the connector in your endpoints, use:

```
<http:inbound-endpoint user="marie" password="marie" host="localhost" port="61205"
  connector-ref="PollingHttpConnector" />
```

## Handling HTTP Content-Type and Encoding

### *Sending*

The following behavior applies when sending POST request bodies as a client and when returning a response body:

For a String, char[], Reader, or similar:

- If the endpoint has encoding set explicitly, use that
- Otherwise, take it from the message's property Content-Type
- If none of these is set, use the Mule Context's configuration default.
- For Content-Type, send the message's property Content-Type but with the actual encoding set.

For binary content, encoding is not relevant. Content-Type is set as follows:

- If the Content-Type property is set on the message, send that.
- Send "application/octet-stream" as Content-Type if none is set on the message.

### *Receiving*

When receiving HTTP responses, the payload of the MuleMessage will always be the InputStream of the HTTP response.

## Including Custom Header Properties

When making a new HTTP client request, Mule filters out any existing HTTP request headers because they are often from a previous request. For

example, if you have an HTTP endpoint that proxies another HTTP endpoint, you wouldn't want to copy the `Content-Type` header property from the first HTTP request to the second request.

If you do want to include HTTP headers, you can specify them as properties on the outbound endpoint as follows:

```
<outbound>
  <chaining-router>
    <outbound-endpoint address="http://localhost:9002/events"
      connector-ref="HttpConnector" content-type="image/png">
      <property key="Accept" value="*.*" />
    </outbound-endpoint>
  </chaining-router>
</outbound>
```

## Building the Target URL from the Request

The HTTP request URL is available in the Mule header. You can access this using the header expression evaluator `#[header:http.request]`. For example, if you want to redirect the request to a different server based on a filter, you can build the target URL as shown below:

```
<filtering-router>
  <outbound-endpoint address="http://localhost:8080#[header:http.request]" />
</filtering-router>
```

## Handling Redirects

To redirect an HTTP client, you must set two properties on the endpoint. First, set the `http.status` property to '307', which instructs the client that the resource has been temporarily redirected. Alternatively, you can set the property to '301' for a permanent redirect. Second, set the `Location` property, which specifies the location where you want to redirect your client.



See the HTTP protocol specification for detailed information about status codes at <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

Following is an example of a service that is listening on the local address `http://localhost:8080/mine` and will send a response with the redirection code, instructing the client to go to `http://mule.mulesoft.org/`.

```
<service name="redirecter">
  <inbound>
    <inbound-endpoint address="http://localhost:8080/mine">
      <property key="http.status" value="307"/>
      <property key="Location" value="http://mule.mulesoft.org/" />
    </inbound-endpoint>
  <inbound>
  </service>
```

Note that you must set the `exchange-pattern` attribute to `request-response`. Otherwise, a response will be immediately returned while the request is placed on an internal queue.

## Getting a Hash Map of POST body params

You can use the custom transformer `HttpRequestBodyToParamMap` on your inbound endpoint to return the message properties as a hash map of name-value pairs. This transformer handles GET and POST with `application/x-www-form-urlencoded` content type.

For example:

```
<http:inbound-endpoint ...>
  <http:body-to-parameter-map-transformer />
</http:inbound-endpoint>
```

## Processing GET Query Parameters

GET parameters posted to an HTTP inbound endpoint are automatically available in the payload on the Mule Message in their raw form and the query parameters are also passed and stored as inbound-scoped headers of the Mule Message.

For example, the following flow creates a simple HTTP server:

```
<flow name="flows1Flow1">
    <http:inbound-endpoint host="localhost" port="8081" encoding="UTF-8"/>
    <logger message="#{groovy: return message.toString(); }" level="INFO"/>
</flow>
```

Doing a request from a browser using the URL:

```
http://localhost:8081/echo?reverb=4&flange=2
```

Will result in a message payload of /echo?reverb=4&flange=2 and two additional inbound headers on the message reverb=4 and flange=2.

These headers can then be accessed using expressions i.e. #[header:INBOUND:reverb] which can be used by filters and routers or injected into your code.

## Serving Static Content (since Mule 3.2)

The HTTP connector can be used as a web server to deliver static content such as images, HTML, JavaScript, CSS files etc. To enable this, configure a flow with an HTTP static-resource-handler:

```
<flow name="main-http">
    <http:inbound-endpoint address="http://localhost:8080/static"/>
    <http:static-resource-handler resourceBase="${app.home}/docroot"
        defaultFile="index.html"/>
</flow>
```

The important attribute here is the `resourceBase` since it defines where on the local system that files will be served from. Typically, this should be set to `${app.home}/docroot`, but it can point to any fully qualified location.

The default file allows you to specify the default resource to load if none is specified. If not set the default is `index.html`.



When developing your Mule application, the `docroot` directory should be located at `<project.home>/src/main/app/docroot`.

## Content-Type Handling

The static-resource-handler uses the same mime type mapping system as the JDK, if you need to add your own mime type to file extension mappings, you need to add a the following file to your application `<project.home>/src/main/resources/META-INF/mime.types`. With content similar to:

image/png	png
text/plain	txt cgi java

This maps the mime type to one or more file extensions.

## Examples

The following provides some common usage examples that will help you get an understanding of how you can use HTTP and Mule.

### Mule Flow

## Polling HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns:vm=
"http://www.mulesoft.org/schema/mule/vm"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/test/3.2/mule-test.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.2/mule.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.2/mule-vm.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.2/mule-http.xsd">

    <!-- We are using two different types of HTTP connector so we must declare them
        both in the config -->
    <http:polling-connector name="PollingHttpConnector"
        pollingFrequency="30000" reuseAddress="true" />

    <http:connector name="HttpConnector" />

    <flow name="polling">
        <http:inbound-endpoint host="localhost" port="8080"
            connector-ref="PollingHttpConnector" exchange-pattern="one-way">
            <property key="Accept" value="application/xml" />
        </http:inbound-endpoint>

        <outbound-endpoint address="vm://toclient" exchange-pattern="one-way" />
    </flow>

    <flow name="polled">
        <inbound-endpoint address="http://localhost:8080"
            connector-ref="HttpConnector" />

        <test:component>
            <test:return-data>foo</test:return-data>
        </test:component>
    </flow>
</mule>
```

## Mule Service

## Polling HTTP

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/test/3.2/mule-test.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.2/mule.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.2/mule-vm.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.2/mule-http.xsd">

    <http:polling-connector name="PollingHttpConnector" pollingFrequency="30000" reuseAddress="true" />

    <http:connector name="HttpConnector" />

    <vm:connector name="vmQueue" />

    <model name="http polling test model">
        <service name="polling">
            <inbound>
                <http:inbound-endpoint host="localhost" port="8080"
                                      connector-ref="PollingHttpConnector" exchange-pattern="one-way">
                    <property key="Accept" value="application/xml"/>
                </http:inbound-endpoint>
            </inbound>
            <test:component/>
            <outbound>
                <pass-through-router>
                    <outbound-endpoint address="vm://toclient" exchange-pattern="one-way"/>
                </pass-through-router>
            </outbound>
        </service>
        <service name="polled">
            <inbound>
                <inbound-endpoint address="http://localhost:8080"
                                  connector-ref="HttpConnector"/>
            </inbound>
            <test:component>
                <test:return-data>foo</test:return-data>
            </test:component>
        </service>
    </model>
</mule>
```

## Mule Flow

### WebServer - Static Content

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.2/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.2/mule-http.xsd">

    <flow name="httpWebServer">
        <http:inbound-endpoint address="http://localhost:8080/static"/>

        <http:static-resource-handler resourceBase="${app.home}/docroot"
                                       defaultFile="index.html"/>
    </flow>
</mule>
```

### Mule Flow

#### Setting Cookies on a Request

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http" xmlns:vm=
"http://www.mulesoft.org/schema/mule/vm"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.2/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.2/mule-http.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.2/mule-vm.xsd">

    <http:connector name="httpConnector" enableCookies="true" />

    <flow name="testService">
        <vm:inbound-endpoint path="vm-in" exchange-pattern="one-way" />

        <http:outbound-endpoint address="http://localhost:${port1}"
                               method="POST" exchange-pattern="one-way" content-type="text/xml">
            <properties>
                <spring:entry key="cookies">
                    <spring:map>
                        <spring:entry key="customCookie" value="yes"/>
                        <spring:entry key="expressionCookie" value="#[header:INBOUND:COOKIE_HEADER]"/>
                    </spring:map>
                </spring:entry>
            </properties>
        </http:outbound-endpoint>
    </flow>
</mule>
```

### Mule Service

## Setting Cookies on a Request

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.2/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.2/mule-http.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.2/mule-vm.xsd">

    <http:connector name="httpConnector" enableCookies="true" />

    <model name="main">
        <service name="testService">
            <inbound>
                <vm:inbound-endpoint path="vm-in" exchange-pattern="one-way" />
            </inbound>
            <outbound>
                <pass-through-router>
                    <http:outbound-endpoint address="http://localhost:${port1}" method="POST"
                        exchange-pattern="one-way" >
                        <properties>
                            <spring:entry key="Content-Type" value="text/xml" />
                            <spring:entry key="cookies">
                                <spring:map>
                                    <spring:entry key="customCookie" value="yes" />
                                    <spring:entry key="expressionCookie"
value="#[header:INBOUND:COOKIE_HEADER]" />
                                </spring:map>
                            </spring:entry>
                        </properties>
                    </http:outbound-endpoint>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Connector

Allows Mule to communicate over HTTP. All parts of the HTTP spec are covered by Mule, so you can expect ETags to be honored as well as keep alive semantics and cookies.

#### Attributes of <connector...>

Name	Type	Required	Default	Description
cookieSpec	netscape/rfc2109	no		The cookie specification to be used by this connector when cookies are enabled.
proxyHostname	string	no		The proxy host name or address.
proxyPassword	string	no		The password to use for proxy access.
proxyPort	port number	no		The proxy port number.
proxyUsername	string	no		The username to use for proxy access.

enableCookies	boolean	no		Whether to support cookies.
---------------	---------	----	--	-----------------------------

#### Child Elements of <connector...>

Name	Cardinality	Description
------	-------------	-------------

This connector also accepts all the attributes from the TCP connector.

For example:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule-http.xsd">

    <http:connector name="HttpConnector" enableCookies="true" keepAlive="true" />
    ...
</mule>
```

cache: Unexpected program error: java.lang.NullPointerException

## Polling Connector

Allows Mule to poll an external HTTP server and generate events from the result. This is useful for pull-only web services.

#### Attributes of <polling-connector...>

Name	Type	Required	Default	Description
cookieSpec	netscape/rfc2109	no		The cookie specification to be used by this connector when cookies are enabled.
proxyHostname	string	no		The proxy host name or address.
proxyPassword	string	no		The password to use for proxy access.
proxyPort	port number	no		The proxy port number.
proxyUsername	string	no		The username to use for proxy access.
enableCookies	boolean	no		Whether to support cookies.
pollingFrequency	long	no		The time in milliseconds to wait between each request to the remote HTTP server.
checkEtag	boolean	no		Whether the ETag header from the remote server is processed if the header is present.
discardEmptyContent	boolean	no		Whether Mule should discard any messages from the remote server that have a zero content length. For many services a zero length would mean there was no data to return. If the remote HTTP server does return content to say that that the request is empty, users can configure a content filter on the endpoint to filter these messages out.

This connector also accepts all the attributes from the TCP connector.

For more information, see [Polling HTTP Services](#) below.

cache: Unexpected program error: java.lang.NullPointerException

## Rest Service Component

Built-in RestServiceWrapper can be used to proxy REST style services as local Mule components.

#### **Attributes of <rest-service-component...>**

Name	Type	Required	Default	Description
httpMethod	DELETE/GET/POST	no	GET	The HTTP method to use when making the service request.
serviceUrl		no		The service URL to use when making the request. This should not contain any parameters, since these should be configured on the component. The service URL can contain Mule expressions, so the URL can be dynamic for each message request.

#### **Child Elements of <rest-service-component...>**

Name	Cardinality	Description
error-filter	0..1	An error filter can be used to detect whether the response from the remote service resulted in an error.
payloadParameterName	0..*	If the payload of the message is to be attached as a URL parameter, this should be set to the parameter name. If the message payload is an array of objects that multiple parameters can be set to, use each element in the array.
requiredParameter	0..*	These are parameters that must be available on the current message for the request to be successful. The Key maps to the parameter name, the value can be any one of the valid expressions supported by Mule.
optionalParameter	0..*	These are parameters that if they are on the current message will be added to the request, otherwise they will be ignored. The Key maps to the parameter name, the value can be any one of the valid expressions supported by Mule.

cache: Unexpected program error: java.lang.NullPointerException

#### **Inbound endpoint**

An inbound HTTP endpoint exposes a service over HTTP, essentially making it an HTTP server. If polling of a remote HTTP service is required, this endpoint should be configured with a polling HTTP connector.

#### **Attributes of <inbound-endpoint...>**

Name	Type	Required	Default	Description
user	string	no		The user name (if any) that will be used to authenticate against.
password	string	no		The password for the user.
host	string	no		The host to connect to. For inbound endpoints, this should be an address of a local network interface.
port	port number	no		The port number to use when a connection is made.
path	string	no		The path for the HTTP URL.
contentType	string	no		The HTTP ContentType to use.
method	httpMethodTypes	no		The HTTP method to use.
keep-alive	boolean	no		Controls if the socket connection is kept alive. If set to true, a keep-alive header with the connection timeout specified in the connector will be returned. If set to false, a "Connection: close" header will be returned.

#### **Child Elements of <inbound-endpoint...>**

Name	Cardinality	Description

For example:

```
<http:inbound-endpoint host="localhost" port="63081" path="services/Echo" keep-alive="true"/>
```

The HTTP inbound endpoint attributes override those specified for the default inbound endpoint attributes.

cache: Unexpected program error: java.lang.NullPointerException

## Outbound endpoint

The HTTP outbound endpoint allows Mule to send requests to external servers or Mule inbound HTTP endpoints using the HTTP protocol.

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
followRedirects	boolean	no		If a request if made using GET that responds with a redirectLocation header, setting this to true will make the request on the redirect URL. This only works when using GET since you cannot automatically follow redirects when performing a POST (a restriction according to RFC 2616).
user	string	no		The user name (if any) that will be used to authenticate against.
password	string	no		The password for the user.
host	string	no		The host to connect to. For inbound endpoints, this should be an address of a local network interface.
port	port number	no		The port number to use when a connection is made.
path	string	no		The path for the HTTP URL.
contentType	string	no		The HTTP ContentType to use.
method	httpMethodTypes	no		The HTTP method to use.
keep-alive	boolean	no		Controls if the socket connection is kept alive. If set to true, a keep-alive header with the connection timeout specified in the connector will be returned. If set to false, a "Connection: close" header will be returned.

### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<http:outbound-endpoint host="localhost" port="8080" method="POST" />
```

The HTTP outbound endpoint attributes override those specified for the default outbound endpoint attributes.

cache: Unexpected program error: java.lang.NullPointerException

## Endpoint

Configures a 'global' HTTP endpoint that can be referenced by services. Services can augment the configuration defined in the global endpoint with local configuration elements.

### Attributes of <endpoint...>

Name	Type	Required	Default	Description
followRedirects	boolean	no		If a request if made using GET that responds with a redirectLocation header, setting this to true will make the request on the redirect URL. This only works when using GET since you cannot automatically follow redirects when performing a POST (a restriction according to RFC 2616).
user	string	no		The user name (if any) that will be used to authenticate against.
password	string	no		The password for the user.

host	string	no		The host to connect to. For inbound endpoints, this should be an address of a local network interface.
port	port number	no		The port number to use when a connection is made.
path	string	no		The path for the HTTP URL.
contentType	string	no		The HTTP ContentType to use.
method	httpMethodTypes	no		The HTTP method to use.
keep-alive	boolean	no		Controls if the socket connection is kept alive. If set to true, a keep-alive header with the connection timeout specified in the connector will be returned. If set to false, a "Connection: close" header will be returned.

#### **Child Elements of <endpoint...>**

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<http:endpoint name="serverEndpoint1" host="localhost" port="60199" path="test1" />
```

The HTTP endpoint attributes override those specified for the [default](#) global endpoint attributes.

cache: Unexpected program error: java.lang.NullPointerException

#### **Transformers**

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description
http-response-to-object-transformer	A transformer that converts an HTTP response to a Mule Message. The payload may be a String, stream, or byte array.
http-response-to-string-transformer	Converts an HTTP response payload into a string. The headers of the response will be preserved on the message.
object-to-http-request-transformer	This transformer will create a valid HTTP request using the current message and any HTTP headers set on the current message.
message-to-http-response-transformer	This transformer will create a valid HTTP response using the current message and any HTTP headers set on the current message.
body-to-parameter-map-transformer	This transformer parses the body of a HTTP request into a Map.

cache: Unexpected program error: java.lang.NullPointerException

#### **Request wildcard filter**

(As of 2.2.2) The `request-wildcard-filter` element can be used to restrict the request by applying wildcard expressions to the URL.

#### **Child Elements of <request-wildcard-filter...>**

Name	Cardinality	Description
------	-------------	-------------

Your Rating: 

Results:  0 rates

# IMAP Transport Reference

## IMAP Transport Reference

### Introduction

The IMAP transport can be used for receiving messages from IMAP inboxes using the `javax.mail` API. The IMAPS Transport uses secure connections over SSL/TLS.

Using the IMAP transport provides a simple way to interact with an IMAP server without having to write your own IMAP client. It allows you to log in to an IMAP server at a specified frequency, pull messages from the server, optionally filter them based on the email subject, and transform them into standard java objects which you can use in your application.

TLS/SSL connections are made on behalf of an entity, which can be anonymous or identified by a certificate. The key store provides the certificates and associated private keys necessary for identifying the entity making the connection. Additionally, connections are made to trusted systems. The public certificates of trusted systems are stored in a trust store, which is used to verify that the connection made to a remote system matches the expected identity.

### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
IMAP	JavaDoc SchemaDoc	✓	✗	✗	✗	✗	✗	one-way	one-way	org.mule.transport:n
IMAPS	JavaDoc SchemaDoc	✓	✗	✗	✗	✗	✗	one-way	one-way	org.mule.transport:n

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

### Namespace and Syntax

XML namespace:

```
xmlns:imap "http://www.mulesoft.org/schema/mule/imap"
xmlns:imaps "http://www.mulesoft.org/schema/mule/imaps"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/imap http://www.mulesoft.org/schema/mule/imap/3.1/mule-imap.xsd
http://www.mulesoft.org/schema/mule/imaps http://www.mulesoft.org/schema/mule/imaps/3.1/mule-imaps.xsd
```

Connector syntax:

```

<imap:connector name="imapConnector" backupEnabled="true" backupFolder="backup" checkFrequency="90000"

    deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED"/>
<imaps:connector name="imapsConnector" backupEnabled="true" backupFolder="backup" checkFrequency=
"90000"
    deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED"/>
<imaps:tls-client path="clientKeystore" storePassword="mulepassword" />
<imaps:tls-trust-store path="greenmail-truststore" storePassword="password" />
</imaps:connector>

```

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```

<imap:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>
<imaps:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>

```

2. Non-prefixed URI:

```

<inbound-endpoint address="imap://bob:password@localhost:65433"/>
<inbound-endpoint address="imaps://bob:password@localhost:65433"/>

```

See the sections below for more information.

## Features

- Simple to configure email access on inbound endpoints: including authentication information and check frequency
- Automate the handling of email attachments
- Automatically back up messages to a specified folder
- Automatically delete read messages
- Easy to configure tls security

## Usage

If you want to include the IMAP email transport in your configuration, these are the namespaces you need to define:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:spring="http://www.springframework.org/schema/beans"
xmlns:imap="http://www.mulesoft.org/schema/mule/imap"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/imap http://www.mulesoft.org/schema/mule/imap/3.1/mule-imap.xsd">
...

```

For the secure version, you would use the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:imaps="http://www.mulesoft.org/schema/mule/imaps"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/imaps
          http://www.mulesoft.org/schema/mule/imaps/3.1/mule-imaps.xsd">
    ...

```

Then you need to configure your connector and endpoints as described below.

### **Configuration Example**

Say you had a business and wanted to take orders through email attachments. After you receive the email, you want to save the order attachments so they could be picked up by your order fulfillment process. The following Mule configuration checks an email box for emails, and saves the attachments to the local disk, where it can be picked up from a separate fulfillment process:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:imap="http://www.mulesoft.org/schema/mule/imap"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:email="http://www.mulesoft.org/schema/mule/email"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule http://www.mulesoft.org/schema/mule/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/imap http://www.mulesoft.org/schema/mule/imap/3.1/mule-imap.xsd
          http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <imap:connector name="imapConnector" />

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <model name="test">
        <service name="incoming-orders">
            <inbound>
                <imap:inbound-endpoint user="bob" password="password" host="emailHost"
                                      port="143" transformer-refs="returnAttachments" disableTransportTransformer="true"
                "/>
                </inbound>
                <outbound>
                    <list-message-splitter-router>
                        <file:outbound-endpoint path=".//received" outputPattern=
                            "##[function:datetimestamp].dat">
                            <expression-transformer>
                                <return-argument expression="payload.inputStream" evaluator="groovy" />
                            </expression-transformer>
                        </file:outbound-endpoint>
                    </list-message-splitter-router>
                </outbound>
            </service>
        </model>
    </mule>

```

The built-in transformer is declared and gets the list of email attachments. This transformer is then applied to the pop3 inbound endpoint defined. Then we define a list list-message-splitter-router which will iterate through all of the email attachments. Next we define a file outbound endpoint which will write the attachment to the './received' directory with a datestamp as the file name . A simple groovy expression gets the inputStream of the attachment to write the file.

Here is a flow-based version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:imap="http://www.mulesoft.org/schema/mule/imap"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:email="http://www.mulesoft.org/schema/mule/email"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/imap http://www.mulesoft.org/schema/mule/imap/3.1/mule-imap.xsd
          http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <imap:connector name="imapConnector" />

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <flow name="incoming-orders">
        <imap:inbound-endpoint user="bob" password="password" host="emailHost"
                               port="143" transformer-refs="returnAttachments" disableTransportTransformer="true
        "/>
        <collection-splitter/>
        <file:outbound-endpoint path=".//received" outputPattern="#{function:timestamp}.dat">
            <expression-transformer>
                <return-argument expression="payload.inputStream" evaluator="groovy" />
            </expression-transformer>
        </file:outbound-endpoint>
    </flow>
</mule>

```

Here is the secure version of the same configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:imaps="http://www.mulesoft.org/schema/mule/imaps"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:file="http://www.mulesoft.org/schema/mule/file"
    xmlns:email="http://www.mulesoft.org/schema/mule/email"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
        http://www.mulesoft.org/schema/mule/imaps http://www.mulesoft.org/schema/mule/imaps/3.1/mule-imaps.xsd
        http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <imaps:connector name="imapsConnector">
        <imaps:tls-client path="clientKeystore" storePassword="mulepassword" />
        <imaps:tls-trust-store path="greenmail-truststore" storePassword="password" />
    </imaps:connector>

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <model name="test">
        <service name="incoming-orders">
            <inbound>
                <imaps:inbound-endpoint user="bob" password="password" host="mailServer"
                    port="110" transformer-refs="returnAttachments" disableTransportTransformer="true"/>
            </inbound>
            <outbound>
                <list-message-splitter-router>
                    <file:outbound-endpoint path=".//received" outputPattern="#{function:datetimestamp}.dat">
                        <expression-transformer>
                            <return-argument expression="payload.inputStream" evaluator="groovy" />
                        </expression-transformer>
                    </file:outbound-endpoint>
                </list-message-splitter-router>
            </outbound>
        </service>
    </model>
</mule>

```

The IMAPS connector has tls client and server keystore information . The built-in transformer is declared and gets the list of email attachments. This transformer is then applied to the inbound endpoint . Then we define a list list-message-splitter-router which will iterate through all of the email attachments. Next we define a file outbound endpoint which will write the attachment to the './received' directory with a datestamp as the file name . A simple groovy expression gets the inputStream of the attachment to write the file.

## Configuration Reference

### Connectors

The IMAP connector supports all the common connector attributes and properties and the following additional attributes:

Attribute	Description	Default	Required
backupEnabled	Whether to save copies to the backup folder	False	No

backupFolder	The folder where messages are moved after they have been read.		No
checkFrequency	Period (ms) between poll connections to the server.	60000	Yes
mailboxFolder	TThe remote folder to check for email.	INBOX	No
deleteReadMessages	Whether to delete messages from the server when they have been downloaded. If set to false, the messages are set to defaultProcessMessageAction attribute value.	true	No
moveToFolder	The remote folder to move mail to once it has been read. It is recommended that 'deleteReadMessages' is set to false when this is used. This is very useful when working with public email services such as GMail where marking messages for deletion doesn't work. Instead set the @moveToFolder=GMail/Trash.		No
defaultProcessMessageAction	The action performed if the deleteReadMessages attribute is set to false. Valid values are: ANSWERED, DELETED, DRAFT, FLAGGED, RECENT, SEEN, USER, and NONE	SEEN	No

For the secure version, the following elements are also required:

Element	Description
tls-client	Configures the client key store with the following attributes: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the keystore that contains public certificates and private keys for identification</li> <li>storePassword: The password used to protect the keystore</li> <li>class: The type of keystore used (a Java class name)</li> </ul>
tls-trust-store	Configures the trust store. The attributes are: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the trust store that contains public certificates of trusted servers</li> <li>storePassword: The password used to protect the trust store</li> </ul>

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:spring="http://www.springframework.org/schema/beans"
       xmlns:imap="http://www.mulesoft.org/schema/mule/imap"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
           http://www.mulesoft.org/schema/mule/imap
           http://www.mulesoft.org/schema/mule/imap/3.1/mule-imap.xsd">

    <imap:connector name="imapConnector" backupEnabled="true" backupFolder="backup" checkFrequency=
    "90000"
                    deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED" />
    ...

```

Secure version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:imaps="http://www.mulesoft.org/schema/mule/imaps"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/imaps
          http://www.mulesoft.org/schema/mule/imaps/3.1/mule-imaps.xsd">

    <imaps:connector name="imapsConnector" backupEnabled="true" backupFolder="backup" checkFrequency=
    "90000"
        deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED"/>
    <imaps:tls-client path="clientKeystore" storePassword="mulepassword" />
    <imaps:tls-trust-store path="greenmail-truststore" storePassword="password" />
</imaps:connector>
...

```

## Endpoints

IMAP and IMAPS endpoints include details about connecting to an IMAP mailbox. You [configure the endpoints](#) just as you would with any other transport, with the following additional attributes:

- user: the user name of the mailbox owner
- password: the password of the user
- host: the name or IP address of the IMAP server, such as www.mulesoft.com, localhost, or 127.0.0.1
- port: the port number of the IMAP server.

For example:

```
<imap:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>
```

Secure version:

```
<imaps:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>
```

You can also define the endpoints using a URI syntax:

```
<inbound-endpoint address="imap://bob:password@localhost:65433"/>
<inbound-endpoint address="imaps://bob:password@localhost:65433"/>
```

This will log into the bob mailbox on localhost on port 65433 using password password. You can also specify the endpoint settings using a URI, but the above syntax is easier to read.

**cache: Unexpected program error: java.lang.NullPointerException**

## Transformers

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description
email-to-string-transformer	Converts an email message to string format.

string-to-email-transformer	Converts a string message to email format.
-----------------------------	--

object-to-mime-transformer	Converts an object to MIME format.
----------------------------	------------------------------------

mime-to-bytes-transformer	Converts a MIME message to a byte array.
---------------------------	--

bytes-to-mime-transformer	Converts a byte array message to MIME format.
---------------------------	---

Here is how you define transformers in your Mule configuration file:

```
<email:bytes-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:email-to-string-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:mime-to-bytes-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
<email:object-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" useInboundAttachments="true" useOutboundAttachments="true"/>
{Note}Need to explain attributes somewhere; can we pull them in from xsd?{Note}
<email:string-to-email-transformer encoding="" ignoreBadInput="" mimeType="" name="" returnClass="" xsi:type="" />
```

Each transformer supports all the common transformer attributes and properties:

**cache: Unexpected program error: java.lang.NullPointerException**

### Transformer

A reference to a transformer defined elsewhere.

#### Attributes of <transformer...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
ref	string	yes		The name of the transformer to use.

#### Child Elements of <transformer...>

Name	Cardinality	Description
------	-------------	-------------

The object-to-mime-transformer has the following attributes:

Attribute	Description	Default Value
useInboundAttachments	Whether to transform inbound attachment in the input message into MIME parts.	true
useOutboundAttachments	Whether to transform outbound attachment in the input message into MIME parts.	true

To use these transformers, make sure you include the 'email' namespace in your mule configuration.

Your Rating: 

Results:  1 rates

### Filters

Filters can be set on an endpoint to filter out any unwanted messages. The Email transport provides a couple of filters that can either be used directly or extended to implement custom filtering rules.

Filter	Description
org.mule.providers.email.filters.AbstractMailFilter	A base filter implementation that must be extended by any other mail filter.
org.mule.providers.email.filters.MailSubjectRegExFilter	Applies a regular expression to a Mail Message subject.

This is how you define the MailSubjectRegExFilter in your Mule configuration:

```
<message-property-filter pattern="to=barney@mule.org" />
```

The 'pattern' attribute is a regular expression pattern. This is defined as java.util.regex.Pattern.

Your Rating: 

Results:  1 rates

### Maven module

The email transports are implemented by the mule-transport-email module. You can find the source for the email transport under transports/email.

If you are using maven to build your application, use the following dependency snippet to include the email transport in your project:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-email</artifactId>
</dependency>
```

### Mule-Maven Dependencies

If you are building Mule ESB from source or including Mule artifacts in your Maven project, it may be necessary to add the 'mule-deps' repository to your Maven configuration. This repository contains third-party binaries which may not be in any other public Maven repository.

To add the 'mule-deps' repository to your Maven project, add the following to your pom.xml:

```
<repositories>
  <repository>
    <id>mule-deps</id>
    <name>Mule Dependencies</name>
    <url>http://dist.codehaus.org/mule/dependencies/maven2</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

Your Rating: Results:  4 rates

## Limitations

The following known limitations affect email transports:

- Retry policies do not work with email transports
- Timeouts are not supported in email transports
- Can't send same object to different email users
- MailSubjectRegExFilter cannot handle mails with attachments

Your Rating: Results:  2 rates

### Escape Your Credentials

If you use a URI-style endpoint and you include the user name and password, escape any characters that are illegal for URIs, such as the @ character. For example, if the user name is user@myco.com, you should enter it as user%40myco.com.

Your Rating: Results:  0 rates

## JDBC Transport Reference

### JDBC Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Example Configurations](#) ] [ [Configuration Reference](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ] [ [Mule-Maven Dependencies](#) ] [ [Best Practices](#) ] [ [\[Mule 3.2\] Data Source Configuration](#) ]

### Introduction

The JDBC Transport allows you to send and receive messages with a database using the JDBC protocol. Common usage includes retrieving, inserting, updating, and deleting database records, as well as invoking stored procedures (e.g., to create new tables dynamically).

Some features are available only with the Mule Enterprise version of the JDBC transport, which is available with version 1.6 and later of Mule Enterprise. These enterprise-only features are noted below.

### Transport Info

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEP's	Default MEP	Maven Art
JDBC	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>				(local,XA)			one-way, request-response	one-way	<a href="#">org.mule.transport.jdbc</a>

### Legend

Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in Maven

### Namespace and Syntax

XML namespace (CE version):

```
xmlns:jdbc="http://www.mulesoft.org/schema/mule/jdbc"
```

XML namespace (EE version):

```
xmlns:jdbc="http://www.mulesoft.org/schema/mule/ee/jdbc"
```

XML Schema location (CE version):

```
http://www.mulesoft.org/schema/mule/jdbc http://www.mulesoft.org/schema/mule/jdbc/3.1/mule-jdbc.xsd
```

XML Schema location (EE version):

```
http://www.mulesoft.org/schema/mule/ee/jdbc  
http://www.mulesoft.org/schema/mule/ee/jdbc/3.1/mule-jdbc-ee.xsd">
```

**For CE or EE, if you create the Spring bean right in the Mule configuration file, you must include the following namespaces:**

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"  
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
      xmlns:spring="http://www.springframework.org/schema/beans"  
      xmlns:jee="http://www.springframework.org/schema/jee"  
      xmlns:util="http://www.springframework.org/schema/util"  
      xmlns:jdbc="http://www.mulesoft.org/schema/mule/jdbc"  
      xsi:schemaLocation="  
        http://www.springframework.org/schema/beans  
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd  
        http://www.springframework.org/schema/jee  
        http://www.springframework.org/schema/jee/spring-jee-2.5.xsd  
        http://www.springframework.org/schema/util  
        http://www.springframework.org/schema/util/spring-util-2.5.xsd  
        http://www.mulesoft.org/schema/mule/core  
        http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd  
        http://www.mulesoft.org/schema/mule/jdbc  
        http://www.mulesoft.org/schema/mule/jdbc/3.1/mule-jdbc.xsd">
```

Connector syntax (CE version):

```
<spring:bean id="jdbcDataSource" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">  
    <spring:property name="driverName" value="org.apache.derby.jdbc.EmbeddedDriver"/>  
    <spring:property name="url" value="jdbc:derby:muleEmbeddedDB;create=true"/>  
</spring:bean>  
  
<jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource" pollingFrequency="10000"  
                 queryRunner-ref="queryRunner" queryTimeout="10" resultSetHandler-ref="resultSetHandler"  
                 transactionPerMessage="true"/>
```

Connector syntax (EE version):

```

<spring:bean id="jdbcDataSource" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">
    <spring:property name="driverName" value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <spring:property name="url" value="jdbc:derby:muleEmbeddedDB;create=true"/>
</spring:bean>

<jdbc:connector name="jdbcEeConnector" pollingFrequency="1000" dataSource-ref="jdbcDataSource"
    queryRunner-ref="queryRunner" queryTimeout="10" resultSetHandler-ref="resultSetHandler"
    transactionPerMessage="true">
    <jdbc:ackSqlCommandExecutorFactory ref="ackSqlCommandExecutorFactory"/>
    <jdbc:sqlCommandRetryPolicyFactory ref="sqlCommandRetryPolicyFactory"/>
    <jdbc:query key="myQuery" value="select * from table"/>
    <jdbc:sqlCommandExecutorFactory ref="sqlCommandExecutorFactory"></jdbc:sqlCommandExecutorFactory>
    <jdbc:sqlStatementStrategyFactory ref="sqlStatementStrategyFactory"/>
</jdbc:connector>

```

Endpoint syntax:

Inbound endpoints:

You can define your endpoints 2 different ways:

1. Prefixed endpoint (valid for both CE and EE jdbc endpoints):

```

<jdbc:inbound-endpoint queryKey="selectQuery" name="jdbcInbound" pollingFrequency="10000"
    queryTimeout="10"
    connector-ref="jdbcConnector" exchange-pattern="one-way">
    <jdbc:transaction action="ALWAYS_BEGIN" />
</jdbc:inbound-endpoint>

```

2. Non-prefixed URI:

```
<inbound-endpoint address="jdbc://getTest?type=1"/>
```

Outbound endpoints:

1. Prefixed endpoint (valid for both CE and EE jdbc endpoints):

```

<jdbc:outbound-endpoint queryKey="selectCount" exchange-pattern="one-way" connector-ref="jdbcConnector"
    queryTimeout="10" >
    <jdbc:transaction action="ALWAYS_BEGIN"/>
</jdbc:outbound-endpoint>

```

2. Non-prefixed URI:

```
<outbound-endpoint address="jdbc://writeTest?type=2"/>
```

## Considerations

Using the JDBC transport is a good idea if you don't already have a database abstraction layer defined for your application. It saves you trouble of writing your own database client code and will be more portable if you decide to change databases in the future. If your application uses a database abstraction layer, then it is usually preferable to use that instead of the JDBC transport.

## Features

The Mule Enterprise JDBC Transport provides key functionality, performance improvements, transformers, and examples not available in the Mule community release. The following table summarizes the feature differences.

Feature	Summary	Mule Community	Mule Enterprise
Inbound SELECT Queries	Retrieve records using the SQL SELECT statement configured on inbound endpoints.	x	x
Large Dataset Retrieval	Enables retrieval arbitrarily large datasets by consuming records in smaller batches.		x
Acknowledgment Statements	Supports ACK SQL statements that update the source or other table after a record is read.	x	x
Basic Insert/Update/Delete Statements	Individual SQL INSERT, UPDATE, and DELETE queries specified on outbound endpoints. One statement is executed at a time.	x	x
Batch Insert/Update/Delete Statements	Support for JDBC batch INSERT, UPDATE, and DELETE statements, so that many statements can be executed together.		x
Advanced JDBC-related Transformers	XML and CSV transformers for easily converting to and from datasets in these common formats.		x
Outbound SELECT Queries	Retrieve records using SQL SELECT statement configured on outbound endpoints. Supports synchronous queries with dynamic runtime parameters.	x	x
Outbound Stored Procedure Support - Basic	Ability to invoke stored procedures on outbound endpoints. Supports IN parameters but not OUT parameters.	x	x
Outbound Stored Procedure Support - Advanced	Same as Basic but includes both IN and OUT parameter support. OUT parameters can be simple data types or cursors		x
Unnamed Queries	Queries that can be invoked programmatically from within service components or other Java code. This is the most flexible option, but also requires writing code.	x	x
Flexible Data Source Configuration	Support for configuration of data sources through JNDI, XAPool, or Spring.	x	x
Transactions	Support for transactions via underlying Transaction Manager.	x	x

Within this features section, each subsection is marked with either the  icon, for features available in the Community Edition, or the  icon, for features available only in the Enterprise Edition.

## Inbound SELECT Queries

Inbound SELECT queries are queries that are executed periodically (according to the `pollingFrequency` set on the connector).

Here is an example:

```
<spring:bean id="jdbcDataSource" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">
    <spring:property name="driverName" value="oracle.jdbc.driver.OracleDriver"/>
    <spring:property name="url" value="jdbc:oracle:thin:user/pass@host:1521:db"/>
</spring:bean>
...
<jdbc:connector name="jdbcConnector" pollingFrequency="10000" dataSource-ref="jdbcDataSource">
    <jdbc:query key="selectLoadedMules"
        value="SELECT ID, MULE_NAME, RANCH, COLOR, WEIGHT, AGE from mule_source"/>
</jdbc:connector>
...
<flow name="AllMules">
    <jdbc:inbound-endpoint queryKey="selectLoadedMules" exchange-pattern="request-response"/>
...
</flow>
...
```

In this example, the `selectLoadedMules` would be invoked every 10 seconds (`pollingFrequency=10000 ms`) . Each record from the result set is converted into a Map (consisting of column/value pairs).

Inbound SELECT queries are limited because (1) generally, they cannot be called synchronously (unnamed queries are an exception), and (2)

they do not support runtime parameters.

## Large Dataset Retrieval

### Overview

Large dataset retrieval is a strategy for retrieving large datasets by fetching records in smaller, more manageable batches. Mule Enterprise provides the key components and transformers needed to implement a wide range of these strategies.

### When To Use It

- When the dataset to be retrieved is large enough to overwhelm memory and connection resources.
- When preserving the order of messages is important.
- When resumable processing is desired (that is, retrieval of the dataset can pick up where it left off, even after service interruption).
- When load balancing the data retrieval among clustered Mule nodes.

### How It Works

Large dataset retrieval does not use conventional inbound SELECT queries to retrieve data. Instead, it uses a Batch Manager component to compute ID ranges for the next batch of records to be retrieved. An outbound SELECT query uses this range to actually fetch the records. The Batch Manager also controls batch processing flow to make sure that it does not process the next batch until the previous batch has finished processing.

Here is an example:

```
...
<spring:bean id="idStore" class="com.mulesoft.mule.transport.jdbc.util.IdStore">
    <spring:property name="fileName" value="/tmp/large-dataset.txt"/>
</spring:bean>
<spring:bean id="seqBatchManager" class="com.mulesoft.mule.transport.jdbc.components.BatchManager">
    <spring:property name="idStore" ref="idStore"/>
    <spring:property name="batchSize" value="10"/>
    <spring:property name="startingPointForNextBatch" value="0" />
</spring:bean>
<spring:bean id="noArgsWrapper"
            class="com.mulesoft.mule.transport.jdbc.components.NoArgsWrapper">
    <spring:property name="batchManager" ref="seqBatchManager"/>
</spring:bean>
<model name="LargeDataSet">
    <service name="BatchService">
        <inbound>
            <inbound-endpoint address="vm://next.batch" exchange-pattern="one-way"/>
        </inbound>
        <component>
            <spring-object bean="noArgsWrapper" />
        </component>
        <outbound>
            ...
        </outbound>
    </service>
</model>
```

First you set up the file which will hold the starting point id for the next batch of records . Next you define your BatchManager and set the idStore, batchSize and starting point . Then you define a 'noArgsWrapper' spring bean and set a reference to the batch manager . is where you define the component which will be called after the inbound endpoint is triggered. Your outbound endpoints can use

```
# [map-payload:lowerId]
```

and

```
# [map-payload:upperId]
```

to reference a batch of database rows.

### Important Limitations

Large dataset retrieval requires that:

1. The source data contains a unique, sequential numeric ID. Records should also be fetched in ascending order with respect to this ID.
2. There are no large gaps in these IDs (no larger than the configured batch size).

### In Combination with Batch Inserts

Combining large dataset retrieval with batch inserts can support simple but powerful ETL use cases.

### Acknowledgment (ACK) Statements

ACK statements are optional SQL statements that are paired with inbound SELECT queries. When an inbound SELECT query is invoked by Mule, the ACK statement is invoked **for each record** returned by the query. Typically, the ACK statement is an UPDATE, INSERT, or DELETE.

An ACK statement would be configured as follows:

```
...
<jdbc:connector name="jdbcConnector" pollingFrequency="10000" dataSource-ref="jdbcDataSource">
    <jdbc:query key="selectLoadedMules"
        value="SELECT ID, PROCESSED from mule_source WHERE PROCESSED is null order by ID"/>
    <jdbc:query key="selectLoadedMules.ack"
        value="update mule_source set PROCESSED='Y' where ID = #[map-payload:ID] "/>
</jdbc:connector>
...
```

Notice the required convention of appending an ".ack" extension to the query name. This convention lets Mule know which inbound SELECT query to pair with the ACK statement.

Also, note that the ACK statement supports parameters. These parameters are bound to any of the column values from the inbound SELECT query (such as #[map-payload:ID] in the case above).

ACK statements are useful when you want an inbound SELECT query to retrieve records from a source table no more than once. Be careful, however, when using ACK statements with larger result sets. As mentioned earlier, an ACK statement gets issued for each record retrieved, and this can be very resource-intensive for even a modest number of records per second (> 100).

### Basic Insert/Update/Delete Statements

SQL INSERT, UPDATE, and DELETE statements are specified on outbound endpoints. These statements are typically configured with parameters, which are bound with values passed along to the outbound endpoint from an upstream component.

**Basic** statements execute just one statement at a time, as opposed to **batch** statements, which execute multiple statements at a time. Basic statements are appropriate for low-volume record processing (<20 records per second), while batch statements are appropriate for high-volume record processing (thousands of records per second).

For example, when a message with a java.util.Map payload is sent to a basic insert/update/delete endpoint, the parameters in the statement are bound with corresponding entries in the Map. In the configuration below, if the message contains a Map payload with {ID=1,TYPE=1,DATA='hello',ACK=0}, the following insert will be issued: "INSERT INTO TEST (ID,TYPE,DATA,ACK) values (1,1,'hello',0)".

```
<jdbc:connector name="jdbcConnector" pollingFrequency="10000" dataSource-ref="jdbcDataSource">
    <jdbc:query key="outboundInsertStatement"
        value="INSERT INTO TEST (ID, TYPE, DATA, ACK) VALUES (#[map-payload:ID],
            #[map-payload:TYPE], #[map-payload:DATA], #[map-payload:ACK])"/>
</jdbc:connector>
...
<flow name="ExampleFlow">
    <inbound-endpoint address="vm://doInsert"/>
    <jdbc:outbound-endpoint queryKey="outboundInsertStatement" />
</flow>
...
```

### Batch Insert/Update/Delete Statements

As mentioned above, **batch** statements represent a significant performance improvement over their **basic** counterparts. Records can be inserted at a rate of thousands per second with this feature.

Usage of batch INSERT, UPDATE, and DELETE statements is the same as for basic statements, except the payload sent to the VM endpoint should be a List of Maps, instead of just a single Map.

Batch Callable Statements are also supported. Usage is identical to Batch Insert/Update/Delete.

## Advanced JDBC-related Transformers

Common integration use cases involve moving CSV and XML data from files to databases and back. This section describes the transformers that perform these actions. These transformers are available in Mule Enterprise only.

### XML-JDBC Transformer

The XML Transformer converts between XML and JDBC-format Maps. The JDBC-format Maps can be used by JDBC outbound endpoints (for select, insert, update, or delete operations).

Transformer Details:

Name	Class	Input	Output
XML -> Maps	com.mulesoft.mule.transport.jdbc.transformers.XMLToMapsTransformer	java.lang.String (XML)	java.util.List (List of Maps. Each Map corresponds to a "record" in the XML.)
Maps -> XML	com.mulesoft.mule.transport.jdbc.transformers.MapsToXMLTransformer	java.util.List (List of Maps. Each Map will be converted into a "record" in the XML)	java.lang.String (XML)

Also, the XML message payload (passed in or out as a String) must adhere to a particular schema format:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="table">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="record"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="record">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded" ref="field"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="field">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:NMTOKEN">
                    <xs:attribute name="name" use="required" type="xs:NCName"/>
                    <xs:attribute name="type" use="required" type="xs:NCName"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Here is an example of a valid XML instance:

```

<table>
    <record>
        <field name="id" type="java.math.BigDecimal">0</field>
        <field name="name" type="java.lang.String">hello</field>
    </record>
</table>

```

The transformer converts each "record" element to a Map of column/value pairs using "fields". The collection of Maps is returned in a List.

The following will return any processed rows in xml format when you go to 'http://localhost:8080/first20' in your browser:

```

<jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource">
    <jdbc:query key="selectLoadedMules"
        value="SELECT ID, PROCESSED from mule_source WHERE PROCESSED is null order by ID"
    />
    <jdbc:query key="selectLoadedMules.ack"
        value="update mule_source set PROCESSED='Y' where ID = #[map-payload:ID]"/>
</jdbc:connector>

<jdbc:maps-to-xml-transformer name="XMLResponseTransformer" />

<message-properties-transformer name="XMLContentTransformer">
    <add-message-property key="Content-Type" value="text/xml" />
</message-properties-transformer>

<flow name="ReportModel">
    <inbound-endpoint address="http://localhost:8080/first20" responseTransformer-refs="XMLResponseTransformer XMLContentTransformer" exchange-pattern="request-response" />
    <jdbc:outbound-endpoint queryKey="selectLoadedMules" exchange-pattern="request-response" />
</flow>

```

## CSV-JDBC Transformer

The CSV Transformer converts between CSV data and JDBC-format Maps. The JDBC-format Maps can be used by JDBC outbound endpoints (for select, insert, update, or delete operations).

Transformer Details:

Name	Class	Input	Output
CSV -> Maps	com.mulesoft.mule.transport.jdbc.transformers.CSVToMapsTransformer	java.lang.String (CSV data)	java.util.List (List of Maps. Each Map corresponds to a "record" in the CSV)
Maps -> CSV	com.mulesoft.mule.transport.jdbc.transformers.MapsToCSVTransformer	java.util.List (List of Maps. Each Map will be converted into a "record" in the CSV)	java.lang.String (CSV data)

The following table summarizes the properties that can be set on this transformer:

Property	Description
delimiter	The delimiter character used in the CSV file. Defaults to comma.
qualifier	The qualifier character used in the CSV file. Used to signify if text contains the delimiter character. Defaults to double quote.
ignoreFirstRecord	Instructs transformer to ignore the first record. Use this if your first row is a list of column names. Defaults to false.
mappingFile	Location of Mapping file. Required. Can either be physical file location or classpath resource name. The DTD format of the Mapping File can be found at: <a href="http://flatpack.sourceforge.net/flatpack.dtd">http://flatpack.sourceforge.net/flatpack.dtd</a> . For examples of this format, see <a href="http://flatpack.sourceforge.net/documentation/index.html">http://flatpack.sourceforge.net/documentation/index.html</a> .

This configuration loads a csv file in the 'mule\_source' table of a database

```

<jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource">
    <jdbc:query key="commitLoadedMules"
        value="insert into mule_source
            (ID, MULE_NAME, RANCH, COLOR, WEIGHT, AGE)
            values
                (#[map-payload:ID:int;in], #[map-payload:MULE_NAME], #[map-payload:RANCH],
                #[map-payload:COLOR], #[map-payload:WEIGHT:int;in], #[map-payload:AGE:int;in])"/>
    </jdbc:connector>

    <file:connector name="fileConnector" autoDelete="false" pollingFrequency="100000000"/>
    <file:endpoint path="/tmp/data" name="get" connector-ref="fileConnector"/>
    <custom-transformer name="ObjectToString" class="org.mule.transformer.simple.ObjectToString"/>
    <jdbc:csv-to-maps-transformer name="CSV2Maps" delimiter="," mappingFile="/tmp/mules-csv-format.xml"
    ignoreFirstRecord="true"/>

    <flow name="CSVLoader">
        <file:inbound-endpoint ref="get" transformer-refs="ObjectToString CSV2Maps">
            <file:filename-wildcard-filter pattern="*.csv"/>
        </file:inbound-endpoint>
        <echo-component/>
        <jdbc:outbound-endpoint queryKey="commitLoadedMules"/>
    </flow>

```

## Outbound SELECT Queries

An inbound SELECT query is invoked on an inbound endpoint according to a specified polling frequency. A major improvement to the inbound SELECT query is the outbound SELECT query, which can be invoked on an outbound endpoint. As a result, the outbound SELECT query can do many things that the inbound SELECT query cannot, such as:

1. Support synchronous invocation of queries. For example, you can implement the classic use case of a web page that serves content from a database using an HTTP inbound endpoint and an outbound SELECT query endpoint.
2. Allows parameters so that values can be bound to the query at runtime. This requires that the message contain a Map payload containing key names that match the parameter names. For example, the following configuration could be used to retrieve an outbound SELECT query:

```

<jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource">
    <jdbc:query key="selectMules"
        value="select * from mule_source where ID between 0 and
        #[header:inbound:max]"/>
    </jdbc:connector>
    <jdbc:maps-to-xml-transformer name="XMLResponseTransformer"/>
    <message-properties-transformer name="XMLContentTransformer">
        <add-message-property key="Content-Type" value="text/xml"/>
    </message-properties-transformer>
    <flow name="ExampleModel">
        <inbound-endpoint address="http://localhost:8080/getMules" exchange-pattern=
        "request-response" responseTransformer-refs="XMLResponseTransformer XMLContentTransformer"/>
        <jdbc:outbound-endpoint queryKey="selectMules" exchange-pattern="request-response"/>
    </flow>

```

In this scenario, if hit the 'http://localhost:8080/getMules?max=3' url, then the following query is executed:

```
SELECT * FROM mule_source WHERE ID between 0 and 3
```

The database rows are transformed into xml which you will see in your browser.

## Outbound Stored Procedure Support - Basic

Stored procedures are supported on outbound endpoints in Mule. Like any other query, stored procedure queries can be listed in the queries map. Following is an example of how stored procedure queries could be defined:

```
<jdbc:connector name="jdbcConnector" pollingFrequency="10000" dataSource-ref="jdbcDataSource">
    <jdbc:query key="storedProc" value="CALL addField()"/>
</jdbc:connector>
```

To denote that we are going to execute a stored procedure and not a simple SQL query, we must start off the query by the text **CALL** followed by the name of the stored procedure.

Parameters to stored procedures can be forwarded by either passing static parameters in the configuration or using the same syntax as for SQL queries (see "Passing in Parameters" below). For example:

```
<jdbc:query key="storedProc1" value="CALL addFieldWithParams(24)"/>
<jdbc:query key="storedProc2" value="CALL addFieldWithParams([#map:payload:value])"/>

<flow name="ExampleModel1">
    <inbound-endpoint address="http://localhost:8080/get" exchange-pattern="request-response"/>
    <jdbc:outbound-endpoint queryKey="storedProc1" exchange-pattern="request-response"/>
</flow>

<flow name="ExampleModel1">
    <inbound-endpoint address="http://localhost:8080/get2" exchange-pattern="request-response"/>
    <jdbc:outbound-endpoint address="jdbc://storedProc2?value=25"/>
</flow>
```

If you do not want to poll the database, you can write a stored procedure that uses HTTP to start a Mule service. The stored procedure can be called from an Oracle trigger. If you take this approach, make sure the exchange pattern is 'one-way'. Otherwise, the trigger/transaction won't commit until the HTTP post returns.

Note that stored procedures are only supported on outbound endpoints. If you want to set up a service that calls a stored procedure at a regular interval, you can define a [Quartz](#) inbound endpoint and then define the stored procedure call in the outbound endpoint. For information on using Quartz to trigger services, see the following [blog post](#).

### Passing in Parameters

To pass in parameter values and get returned values to/from stored procedures or stored functions in Oracle, you declare the parameter name, direction, and type in the JDBC query key/value pairs on JDBC connectors using the following syntax:

```
Call #[<return parameter name>;<int | float | double | string | resultSet>;<out>] :=  
<Oracle package name>.<stored procedure/function name>($PARAM1, $PARAM2, ...)
```

where \$PARAMn is specified using the following syntax:

```
# [<parameter name>;<int | float | double | string | resultSet>;<in | out | inout>]
```

For example:

```
<jdbc:query key="SingleCursor" value="call MULEPACK.TEST_CURSOR(#[mules;resultSet;out])"/>
```

This SQL statement calls a stored procedure TEST\_CURSOR in the package of MULEPACK, specifying an out parameter whose name is "mules" of type java.sql.ResultSet.

Here is another example:

```
<jdbc:query key="itcCheckMsgProcessedOrNot"
value="call #[mules:int;out] := ITCPACK.CHECK_IF_MSG_IS_HANDLED_FNC(487568,#[mules1:string;out],
#[mules2:string;out],#[mules3:int;out],#[mules4:string;out])"/>
```

This SQL statement calls a stored function `CHECK_IF_MSG_IS_HANDLED_FNC` in the package of `ITCPACK`, assigning a return value of integer to the parameter whose name is "mules" while specifying other parameters, e.g., parameter "mules2" is a out string parameter.

Stored procedures/functions can only be called on JDBC outbound endpoints. Once the values are returned from the database, they are put in a `java.util.HashMap` with key/value pairs. The keys are the parameter names, e.g., "mules2", while the values are the Java data values (Integer, String, etc.). This hash map is the payload of MuleMessage either returned to the caller or sent to the next endpoint depending on the Mule configuration.

## Outbound Stored Procedure Support - Advanced

Mule Enterprise provides advanced stored procedure support for outbound endpoints beyond what is available in the Mule community release. This section describes the advanced support.

### OUT Parameters

In Mule Enterprise, you can execute your stored procedures with `out` and `inout` scalar parameters. The syntax for such parameters is:

```
<jdbc:query key="storedProcl" value="CALL myProc(#[a], #[b:int;inout], #[c:string;out])"/>
```

You must specify the type of each output parameter (OUT, INOUT) and its data type (int, string, etc.). The result of such stored procedures is a map containing (out parameter name, value) entries.

### Oracle Cursor Support

For Oracle databases only, an OUT parameter can return a cursor. The following example shows how this works.

If you want to handle the cursor as a `java.sql.ResultSet`, see the "cursorOutputAsResultSet" service below, which uses the "MapLookup" transformer to return the `ResultSet`.

If you want to handle the cursor by fetching the `java.sql.ResultSet` to a collection of Map objects, see the "cursorOutputAsMaps" service below, which uses both the "MapLookup" and "ResultSet2Maps" transformers to achieve this result.

```
<jdbc:connector name="jdbcConnector" pollingFrequency="1000" cursorTypeConstant="-10"
    dataSource-ref="jdbcDataSource">
    <jdbc:query key="SingleCursor" value="call TEST_CURSOR(#[mules:resultSet;out])"/>
</jdbc:connector>

<custom-transformer class="org.mule.transformer.simple.MapLookup" name="MapLookup">
    <spring:property name="key" value="mules"/>
</custom-transformer>

<jdbc:resultSet-to-maps-transformer name="ResultSet2Maps" />

<flow name="SPModel">
    <vm:inbound-endpoint path="returns.maps" responseTransformer-refs="ResultSet2Maps MapLookup"/>
    <jdbc:outbound-endpoint queryKey="SingleCursor"/>
</flow>
<flow name="cursorOutputAsResultSet">
    <vm:inbound-endpoint path="returns.resultset" responseTransformer-refs="MapLookup"/>
    <jdbc:outbound-endpoint queryKey="SingleCursor"/>
</flow>
```

In the above example, note that it is also possible to call a function that returns a cursor ref. For example, if `TEST_CURSOR2()` returns a cursor ref, the following statement could be used to get that cursor as a `ResultSet`:

```
<jdbc:query key="SingleCursor" value="call #[mules:resultSet;out] := TEST_CURSOR2()"/>
```



**Important note on transactions:** When calling stored procedures or functions that return cursors (`ResultSet`), it is recommended that you process the `ResultSet` within a transaction.

## Unnamed Queries

SQL statements can also be executed without configuring queries in the Mule configuration file. For a given endpoint, the query to execute can be specified as the address of the URI.

```
MuleMessage msg = eventContext.receiveEvent("jdbc://SELECT * FROM TEST", 0);
```

## Flexible Data Source Configuration

You can use any JDBC data source library with the JDBC Connector. The "myDataSource" reference below refers to a DataSource bean created in Spring:

```
<jdbc:connector name="jdbcConnector" pollingFrequency="10000" dataSource-ref="myDataSource">
    ...
</jdbc:connector>
```

You can also create a JDBC connection pool so that you don't create a new connection to the database for each message. You can easily create a pooled data source in Spring using `xapool`. The following example shows how to create the Spring bean right in the Mule configuration file.

```
<spring:bean id="pooledDS" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">
    <spring:property name="driverName" value="oracle.jdbc.driver.OracleDriver"/>
    <spring:property name="url" value="jdbc:oracle:thin:user/pass@host:1521:db"/>
</spring:bean>
```

If you need more control over the configuration of the pool, you can use the standard JDBC classes. For example, you could create the following bean in the Spring configuration file (you could also create them in the Mule configuration file by prefixing everything with the Spring namespace):

```
<bean id="c3p0DataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
    <property name="driverClass">
        <value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="jdbcUrl">
        <value>jdbc:oracle:thin:@MyUrl:MySID</value>
    </property>

    <property name="user">
        <value>USER</value>
    </property>
    <property name="password">
        <value>PWD</value>
    </property>

    <property name="properties">
        <props>
            <prop key="c3p0.acquire_increment">5</prop>
            <prop key="c3p0.idle_test_period">100</prop>
            <prop key="c3p0.max_size">100</prop>
            <prop key="c3p0.max_statements">1</prop>
            <prop key="c3p0.min_size">10</prop>
            <prop key="user">USER</prop>
            <prop key="password">PWD</prop>
        </props>
    </property>
</bean>
```

You could then reference the `c3p0DataSource` bean in your Mule configuration:

```

<connector name="C3p0Connector" className="org.mule.providers.jdbc.JdbcConnector">
  <properties>
    <container-property name="dataSource" reference="c3p0DataSource"/>
    <map name="queries">
      <property name="test1" value="select * from Table1"/>
      <property name="test2" value="call testd(1)"/>
    </map>
  </properties>
</connector>

```

Or you could call it from your application as follows:

```

JdbcConnector jdbcConnector = (JdbcConnector)
MuleServer.getMuleContext().getRegistry().lookupConnector("C3p0Connector");
ComboPooledDataSource datasource = (ComboPooledDataSource)jdbcConnector.getDataSource();
Connection connection = (Connection)datasource.getConnection();

String query = "select * from Table1"; //any query
Statement stat = connection.createStatement();
ResultSet rs = stat.executeQuery(query);

```

To retrieve the data source from a JNDI repository, you would configure the connector as follows:

```

<spring:beans>
  <jee:jndi-lookup id="myDataSource" jndi-name="yourJndiName" environment-ref="yourJndiEnv" />
  <util:map id="jndiEnv">
    <spring:entry key="java.naming.factory.initial" value="yourJndiFactory" />
  </util:map>
</spring:beans>

```

## Transactions

Transactions are supported on JDBC endpoints. See [Transaction Management](#) for details.

## Usage

Copy your JDBC client jar to the <MULE\_HOME>/lib/user directory of your installation.

If you want to include the JDBC transport in your configuration, these are the namespaces you need to define:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:jdbc="http://www.mulesoft.org/schema/mule/jdbc"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/jdbc http://www.mulesoft.org/schema/mule/jdbc/3.1/mule-jdbc.xsd">
  ...

```

For the enterprise version of the JDBC transport:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:jdbc="http://www.mulesoft.org/schema/mule/ee/jdbc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/ee/jdbc
        http://www.mulesoft.org/schema/mule/ee/jdbc/3.1/mule-jdbc-ee.xsd">
    ...

```

Then you need to define a connector:

```

<spring:bean id="jdbcDataSource" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">
    <spring:property name="driverName" value="org.apache.derby.jdbc.EmbeddedDriver"/>
    <spring:property name="url" value="jdbc:derby:muleEmbeddedDB;create=true"/>
</spring:bean>

<jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource" pollingFrequency="10000"
    queryRunner-ref="queryRunner" queryTimeout="10" resultSetHandler-ref="resultSetHandler"
    transactionPerMessage="true"/>

```

Finally, you define an inbound or outbound endpoint.

- Use an inbound endpoint if you want changes to your database to trigger a Mule flow or service
- Use an outbound endpoint to make changes to the database data or to return database data to an inbound endpoint, such as using an http endpoint to display database data.

Endpoints look like this:

Inbound endpoints:

```

<jdbc:inbound-endpoint queryKey="selectQuery" name="jdbcInbound" pollingFrequency="10000"
    queryTimeout="10">
    <connector-ref="jdbcConnector" exchange-pattern="one-way">
        <jdbc:transaction action="ALWAYS_BEGIN" />
    </connector-ref>
</jdbc:inbound-endpoint>

```

Outbound endpoints:

```

<jdbc:outbound-endpoint queryKey="selectCount" exchange-pattern="one-way" connector-ref="jdbcConnector"
    queryTimeout="10" >
    <jdbc:transaction action="ALWAYS_BEGIN"/>
</jdbc:outbound-endpoint>

```



If you are using Mule Enterprise edition, then you must use the EE version of the JDBC transport. Therefore, if you are migrating from CE to EE, you will need to update the namespace and schemaLocation declarations to the EE versions as described above.

## Exchange patterns

one-way and request-response exchange patterns are supported. If an exchange pattern is not defined, 'one-way' is the default.

## Polling transport

The inbound endpoint for JDBC transport uses polling to look for new data. The default is to check every second, but it can be changed via the 'pollingFrequency' attribute on the connector.

### **Features supported by this module: Transactions, reconnect, expressions, etc.**

Most standard transport features are supported for the jdbc transport: transactions, retry, expressions, etc. Streaming is not supported for the JDBC transport.

## **Example Configurations**

The following example demonstrates how you would write rows in a database to their own files.

### **Writing database rows to their own files**

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jdbc="http://www.mulesoft.org/schema/mule/jdbc"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/jdbc
          http://www.mulesoft.org/schema/mule/jdbc/3.1/mule-jdbc.xsd">

    <!-- This placeholder bean will let you import the properties from the db.properties file. -->
    <spring:bean id="property-placeholder" class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:db.properties"/>
    </spring:bean>

    <!-- This data source is used to connect to the database using the values loaded from the properties file -->
    <spring:bean id="jdbcDataSource"
        class="org.enhydra.jdbc.standard.StandardDataSource"
        destroy-method="shutdown">
        <spring:property name="driverName" value="${database.driver}"/>
        <spring:property name="url" value="${database.connection}"/>
    </spring:bean>

    <jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource" pollingFrequency="5000"
transactionPerMessage="false">
        <jdbc:query key="read" value="SELECT id, type, data FROM test3 WHERE type=1"/>
        <jdbc:query key="read.ack" value="UPDATE test3 SET type=2 WHERE id=#{map-payload:id}"/>
    </jdbc:connector>

    <file:connector name="output" outputAppend="true" outputPattern="#{function:datetestamp}.txt" />

    <flow name="allDbRows">
        <jdbc:inbound-endpoint queryKey="read" connector-ref="jdbcConnector"/>
        <object-to-string-transformer />
        <file:outbound-endpoint connector-ref="output" path="/tmp/rows" />
    </flow>
</mule>
```

The database authentication information is stored in a properties file named 'db.properties' . For a MySQL database, the file would look similar to this:

```
database.driver=com.mysql.jdbc.Driver
database.connection=jdbc:mysql://localhost/test?user=<user>&password=<password>
```

The values in the property file are used in and to configure the data source bean. The jdbc connector references the data source and defines a couple of queries ( and ) which the inbound endpoint will use. The 'read' query checks the database for rows which have a 'type' column set to 1. The 'read.ack' query is automatically run for every new record found and sets the 'type' column to 2 so it will not be picked up again by the indound endpoint. A file connector is defined at to write each row found to a file with a date stamp name. Next, the flow is defined which calls the

jdbc 'read' query on the inbound endpoint . New database rows are then processed by the object-to-string transformer and finally written to the '/tmp/rows' directory .

This example shows how to display database rows in a browser:

### Display database rows in a browser

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jdbc="http://www.mulesoft.org/schema/mule/ee/jdbc"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/ee/jdbc
          http://www.mulesoft.org/schema/mule/ee/jdbc
          http://www.mulesoft.org/schema/mule/ee/jdbc-ee.xsd">

    <!-- This placeholder bean will let you import the properties from the db.properties file. -->
    <spring:bean id="property-placeholder" class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:db.properties"/>
    </spring:bean>

    <!-- This data source is used to connect to the database using the values loaded from the
properties file -->
    <spring:bean id="jdbcDataSource"
        class="org.enhydra.jdbc.standard.StandardDataSource"
        destroy-method="shutdown">
        <spring:property name="driverName" value="${database.driver}"/>
        <spring:property name="url" value="${database.connection}"/>
    </spring:bean>
    <jdbc:connector name="jdbcConnector" dataSource-ref="jdbcDataSource">
        <jdbc:query key="selectRows"
            value="select * from mule_source where ID between 0 and #[header:inbound:max]"/>
    </jdbc:connector>
    <jdbc:maps-to-xml-transformer name="XMLResponseTransformer"/>
    <message-properties-transformer name="XMLContentTransformer">
        <add-message-property key="Content-Type" value="text/xml"/>
    </message-properties-transformer>
    <flow name="ExampleModel">
        <inbound-endpoint address="http://localhost:8080/rows" exchange-pattern="request-response"
responseTransformer-refs="XMLResponseTransformer XMLContentTransformer"/>
        <jdbc:outbound-endpoint queryKey="selectRows" exchange-pattern="request-response"/>
    </flow>
</mule>
```

This example requires Mule Enterprise Edition to run. defines a select database query using the 'max' parameter which will be passed in the requesting url. We define some transformers at and to turn the database row into xml and set the correct Content-type for the browser to display it correctly. declares the http inbound endpoint with a url of 'http://localhost:8080/rows'. Since we are using an inbound parameter in the select query, we also need to include the 'max' parameter on the requesting url, i.e. <http://localhost:8080/rows?max=5>. is where the jdbc outbound endpoint will call the 'selectRows' query after the http endpoint is triggered.

## Configuration Reference

### **Community edition:**

cache: Unexpected program error: java.lang.NullPointerException

### **Connector**

#### **Attributes of <connector...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database. This is only applied to queries configured on inbound endpoints.
dataSource-ref	string	yes		Reference to the JDBC DataSource object. This object is typically created using Spring. When using XA transactions, an XADataSource object must be provided.
queryRunner-ref	string	no		Reference to the QueryRunner object, which is the object that actually runs the Query. This object is typically created using Spring. Default is org.apache.commons.dbutils.QueryRunner.
resultSetHandler-ref	string	no		Reference to the ResultSetHandler object, which is the object that determines which java.sql.ResultSet gets handled. This object is typically created using Spring. Default is org.apache.commons.dbutils.handlers.MapListHandler, which steps through the ResultSet and stores records as Map objects on a List.
transactionPerMessage	boolean	no		Whether each database record should be received in a separate transaction. If false, there will be a single transaction for the entire result set. Default is true.
queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-sqlStatementStrategyFactory	0..1	The factory that determines the execution strategy based on the SQL provided.
abstract-query	0..*	Defines a set of queries. Each query has a key and a value (SQL statement). Queries are later referenced by key.

cache: Unexpected program error: java.lang.NullPointerException

#### Inbound endpoint

Receives or fetches data from a database. You can reference SQL select statements or call stored procedures on inbound endpoints. Statements on the inbound endpoint get invoked periodically according to the pollingInterval. Statements that contain an insert, update, or delete are not allowed.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
queryKey	string	no		The key of the query to use.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-xa-transaction	0..1	A placeholder for XA transaction elements. XA transactions allow a series of operations to be grouped together spanning different transports, such as JMS and JDBC.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.

abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
abstract-query	0..*	

cache: Unexpected program error: java.lang.NullPointerException

### **Outbound endpoint**

You can reference any SQL statement or call a stored procedure on outbound endpoints. Statements on the outbound endpoint get invoked synchronously. SQL select statements or stored procedures may return output that is handled by the ResultSetHandler and then attached to the message as the payload.

#### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
exchange-pattern	one-way/request-response	no		
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mime-type	string	no		The mime type, e.g. text/plain or application/json

queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
queryKey	string	no		The key of the query to use.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-xa-transaction	0..1	A placeholder for XA transaction elements. XA transactions allow a series of operations to be grouped together spanning different transports, such as JMS and JDBC.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
abstract-query	0..*	

#### Enterprise edition:

cache: Unexpected program error: java.lang.NullPointerException

#### Connector

#### Attributes of <connector...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.

dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
createMultipleTransactedReceivers	boolean	no		Whether to create multiple concurrent receivers for this connector. This property is used by transports that support transactions, specifically receivers that extend the TransactedPollingMessageReceiver, and provides better throughput.
numberOfConcurrentTransactedReceivers	integer	no		If createMultipleTransactedReceivers is set to true, the number of concurrent receivers that will be launched.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database. This is only applied to queries configured on inbound endpoints.
dataSource-ref	string	yes		Reference to the JDBC DataSource object. This object is typically created using Spring. When using XA transactions, an XADataSource object must be provided.
queryRunner-ref	string	no		Reference to the QueryRunner object, which is the object that actually runs the Query. This object is typically created using Spring. Default is org.apache.commons.dbutils.QueryRunner.
resultSetHandler-ref	string	no		Reference to the ResultSetHandler object, which is the object that determines which java.sql.ResultSet gets handled. This object is typically created using Spring. Default is org.apache.commons.dbutils.handlers.MapListHandler, which steps through the ResultSet and stores records as Map objects on a List.
transactionPerMessage	boolean	no		Whether each database record should be received in a separate transaction. If false, there will be a single transaction for the entire result set. Default is true.
queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.

spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-sqlStatementStrategyFactory	0..1	The factory that determines the execution strategy based on the SQL provided.
abstract-query	0..*	Defines a set of queries. Each query has a key and a value (SQL statement). Queries are later referenced by key.
sqlCommandExecutorFactory	0..1	The factory that creates the command executor for the read SQL statement.
ackSqlCommandExecutorFactory	0..1	The factory that creates the command executor for the acknowledge SQL statement.
sqlCommandRetryPolicyFactory	0..1	The factory that creates the retry policies which decide if a SQL statements must be reexecuted in case of errors.

cache: Unexpected program error: java.lang.NullPointerException

### ***Inbound endpoint***

Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### ***Outbound endpoint***

Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

NOTE: The XSLT has been modified so that you can set the top level wiki header. In this example, the topstylelevel is set to 3, enabling you to generate element docs from here without rupturing the logic of your other header styles.

### ***Transformers***

The following transformers can be found in the enterprise version of the jdbc transport:

cache: Unexpected program error: java.lang.NullPointerException

#### ***Maps to xml transformer***

Converts a List of Map objects to XML. The Map List is the same as what you get from using the default ResultSetHandler. The XML schema format is provided in the documentation.

Child Elements of <maps-to-xml-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### ***Xml to maps transformer***

Converts XML to a List of Map objects. The Map List is the same as what you get from using the default ResultSetHandler. The XML schema format is provided in the documentation.

Child Elements of <xml-to-maps-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Maps to csv transformer**

Converts a List of Map objects to a CSV file. The Map List is the same as what you get from using the default ResultSetHandler.

#### **Attributes of <maps-to-csv-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.
ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
contentType	string	no		The mime type, e.g. text/plain or application/json
delimiter	string	no		Delimiter used in CSV file. Default is comma.
mappingFile	string	no		Name of the "mapping file" used to describe the CSV file. See <a href="http://flatpack.sourceforge.net">http://flatpack.sourceforge.net</a> for details.
ignoreFirstRecord	boolean	no		Whether to ignore the first record. If the first record is a header, you should ignore it.
qualifier	string	no		The character used to escape text that contains the delimiter.

#### **Child Elements of <maps-to-csv-transformer...>**

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### **Csv to maps transformer**

Converts a CSV file to a List of Map objects. The Map List is the same as what you get from using the default ResultSetHandler.

#### **Attributes of <csv-to-maps-transformer...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.
returnClass	string	no		The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.

ignoreBadInput	boolean	no		Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.
encoding	string	no		String encoding used for transformer output.
mimeType	string	no		The mime type, e.g. text/plain or application/json
delimiter	string	no		Delimiter used in CSV file. Default is comma.
mappingFile	string	no		Name of the "mapping file" used to describe the CSV file. See <a href="http://flatpack.sourceforge.net">http://flatpack.sourceforge.net</a> for details.
ignoreFirstRecord	boolean	no		Whether to ignore the first record. If the first record is a header, you should ignore it.
qualifier	string	no		The character used to escape text that contains the delimiter.

#### Child Elements of <csv-to-maps-transformer...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Resultset to maps transformer

Transforms a `java.sql.ResultSet` to a `List` of `Map` objects just like the default `ResultSetHandler`. Useful with Oracle stored procedures that return cursors (`ResultSets`).

#### Child Elements of <resultset-to-maps-transformer...>

Name	Cardinality	Description
------	-------------	-------------

Filters

Others

NOTE: Transports may have associated transformers, filters, etc. Provide listings for all of these as well.

#### Schema

You can view the full schema for the JDBC transport [here](#). The enterprise version of the schema docs is not available.

#### Javadoc API Reference

The Javadoc for this transport can be found [here](#). Refer to the EE distribution for the enterprise version of the jdbc transport javadocs.

#### Maven

The JDBC transport is implemented by the mule-transport-jdbc module. You can find the source for the jdbc transport under transports/jdbc.

If you are using maven to build your application, use the following dependency snippet to include the JDBC transport in your project:  
Community version:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-email</artifactId>
  <version>3.1.0</version>
</dependency>
```

Enterprise version:

```

<dependency>
    <groupId>com.mulesoft.muleesb.transports</groupId>
    <artifactId>mule-transport-jdbc-ee</artifactId>
    <version>3.1.0</version>
</dependency>

```

## Mule-Maven Dependencies

If you are building Mule ESB from source or including Mule artifacts in your Maven project, it may be necessary to add the 'mule-deps' repository to your Maven configuration. This repository contains third-party binaries which may not be in any other public Maven repository.

To add the 'mule-deps' repository to your Maven project, add the following to your pom.xml:

```

<repositories>
    <repository>
        <id>mule-deps</id>
        <name>Mule Dependencies</name>
        <url>http://dist.codehaus.org/mule/dependencies/maven2</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>

```

Your Rating: 

Results:  4 rates

## Best Practices

- Put your database connection and credential information in a separate properties file. This allows your port you configuration file to different environments. See [Example Configurations](#) or the [JDBC Transport Example](#) for an example on how this is done

[anchor:datasource]

### [Mule 3.2] Data Source Configuration

Data source configuration has become much simpler. Previously, a data source had to be configured with Spring:

```

<spring:bean id="dataSource" class="org.enhydra.jdbc.standard.StandardDataSource" destroy-method="shutdown">
    <spring:property name="driverName" value="com.mysql.jdbc.Driver"/>
    <spring:property name="url" value="jdbc:mysql://localhost/mule"/>
    <spring:property name="user" value="mysql"/>
    <spring:property name="password" value="secret"/>
</spring:bean>

```

Now this is greatly simplified:

```

<jdbc:mysql-data-source name="dataSource" database="mule" user="mysql" password="secret"/>

```

## Data sources

The following elements can be used with all the database-specific data sources listed below:

Attribute	Description
loginTimeout	Login timeout.

transactionIsolation	Transaction isolation level to set on the newly created <code>javax.sql.Connection</code> object.
----------------------	---

## Derby

Derby data sources are created as embedded data sources. So the definition of user and password is not required.

Example:

```
<jdbc:derby-data-source name="dataSource" database="mule" />
```

The following attributes are available on the `derby-data-source` element:

Attribute	Description
create	If <code>true</code> the database will be created upon first access. See the <a href="#">Derby documentation</a> for details.
database	Name of the database to connect to. This attribute cannot be used together with the <code>url</code> attribute.
name	Unique identifier of the data source. Use this name to reference the data source from the JDBC connector.
url	JDBC URL to use when connecting to the database. This attribute cannot be used together with the <code>database</code> attribute.

## MySQL

Example:

```
<jdbc:mysql-data-source name="dataSource" database="mule" user="mysql" password="secret" />
```

The following attributes are available on the `mysql-data-source` element:

Attribute	Description
database	Name of the database to connect to. This attribute cannot be used together with the <code>url</code> attribute.
host	Database host to connect to. This attribute cannot be used together with the <code>url</code> attribute.
name	Unique identifier of the data source. Use this name to reference the data source from the JDBC connector.
password	Password for connecting to the database. This attribute is required.
port	Database port to connect to. This attribute cannot be used together with the <code>url</code> attribute.
url	JDBC URL to use when connecting to the database. This attribute cannot be used together with the <code>database</code> , <code>host</code> or <code>port</code> attribute.
user	User for connecting to the database. This attribute is required.

## Oracle

Example:

```
<jdbc:oracle-data-source name="dataSource" user="scott" password="tiger" />
```

The following attributes are available on the `oracle-data-source` element:

Attribute	Description
host	Database host to connect to. This attribute cannot be used together with the <code>url</code> attribute.
instance	Oracle Instance to connect to. This attribute cannot be used together with the <code>url</code> attribute.
name	Unique identifier of the data source. Use this name to reference the data source from the JDBC connector.

password	Password for connecting to the database. This attribute is required.
port	Database port to connect to. This attribute cannot be used together with the <code>url</code> attribute.
url	JDBC URL to use when connecting to the database. This attribute cannot be used together with the <code>instance</code> , <code>host</code> or <code>port</code> attribute.
user	User for connecting to the database. This attribute is required.

## Postgresql

Example:

```
<jdbc:postgresql-data-source name="dataSource" database="mule" user="postgres" password="secret"/>
```

The following attributes are available on the `mysql-data-source` element:

Attribute	Description
database	Name of the database to connect to. This attribute cannot be used together with the <code>url</code> attribute.
host	Database host to connect to. This attribute cannot be used together with the <code>url</code> attribute.
name	Unique identifier of the data source. Use this name to reference the data source from the JDBC connector.
password	Password for connecting to the database. This attribute is required.
port	Database port to connect to. This attribute cannot be used together with the <code>url</code> attribute.
url	JDBC URL to use when connecting to the database. This attribute cannot be used together with the <code>database</code> , <code>host</code> or <code>port</code> attribute.
user	User for connecting to the database. This attribute is required.

Your Rating: 

Results:  1 rates

## JDBC Transport Configuration Reference

### JDBC Transport Configuration Reference

Following are the elements in the JDBC transport schema.

cache: Unexpected program error: java.lang.NullPointerException

#### Connector

##### Attributes of <connector...>

Name	Type	Required	Default	Description
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database. This is only applied to queries configured on inbound endpoints.
dataSource-ref	string	no		Reference to the JDBC DataSource object. This object is typically created using Spring. When using XA transactions, an XADataSource object must be provided.
queryRunner-ref	string	no		Reference to the QueryRunner object, which is the object that actually runs the Query. This object is typically created using Spring. Default is org.apache.commons.dbutils.QueryRunner.
resultSetHandler-ref	string	no		Reference to the ResultSetHandler object, which is the object that determines which java.sql.ResultSet gets handled. This object is typically created using Spring. Default is org.apache.commons.dbutils.handlers.MapListHandler, which steps through the ResultSet and stores records as Map objects on a List.
transactionPerMessage	boolean	no		Whether each database record should be received in a separate transaction. If false, there will be a single transaction for the entire result set. Default is true.

queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
--------------	-----	----	----	---

#### Child Elements of <connector...>

Name	Cardinality	Description
abstract-sqlStatementStrategyFactory	0..1	The factory that determines the execution strategy based on the SQL provided.
abstract-query	0..*	Defines a set of queries. Each query has a key and a value (SQL statement). Queries are later referenced by key.

cache: Unexpected program error: java.lang.NullPointerException

#### **Inbound Endpoint**

Receives or fetches data from a database. You can reference SQL select statements or call stored procedures on inbound endpoints. Statements on the inbound endpoint get invoked periodically according to the pollingInterval. Statements that contain an insert, update, or delete are not allowed.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
queryKey	string	no		The key of the query to use.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
abstract-query	0..*	

cache: Unexpected program error: java.lang.NullPointerException

#### **Outbound Endpoint**

You can reference any SQL statement or call a stored procedure on outbound endpoints. Statements on the outbound endpoint get invoked synchronously. SQL select statements or stored procedures may return output that is handled by the ResultSetHandler and then attached to the message as the payload.

#### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
queryKey	string	no		The key of the query to use.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
abstract-query	0..*	

cache: Unexpected program error: java.lang.NullPointerException

#### **Endpoint**

#### Attributes of <endpoint...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

queryTimeout	int	no	-1	The timeout in seconds that will be used as a query timeout for the SQL statement
queryKey	string	no		The key of the query to use.
pollingFrequency	long	no		The delay in milliseconds that will be used during two subsequent polls to the database.

#### Child Elements of <endpoint...>

Name	Cardinality	Description
abstract-query	0..*	

cache: Unexpected program error: java.lang.NullPointerException

#### Query

Assigns a name (key) to a query (value). Queries are then referenced by key, such as jdbc://myQuery for <jdbc:query key="myQuery" value="select \* from table"/>

cache: Unexpected program error: java.lang.NullPointerException

#### Transaction

Standard Mule transaction configuration. See [Transaction Management](#) for usage details.

cache: Unexpected program error: java.lang.NullPointerException

#### SqlStatementStrategyFactory

Override the default SqlStatementStrategyFactory. Determines the execution strategy based on the SQL provided.

#### Attributes of <sqlStatementStrategyFactory...>

Name	Type	Required	Default	Description
class	class name	no		
ref	string	no		

#### Enterprise-only Features

In addition to the configuration properties shown above, the Enterprise edition of the JDBC transport contains the following elements:

cache: Unexpected program error: java.lang.NullPointerException

#### Maps To Xml Transformer

Converts a List of Map objects to XML. The Map List is the same as what you get from using the default ResultSetHandler. The XML schema format is provided in the documentation.



#### Changes to maps-to-xml-transformer

The maps-to-xml-transformer (implemented in MapsToXMLTransformer) transforms a single Map or a List of Maps into a XML string representation.

The XML has the following structure:

### Structure of XML Output of maps-to-xml-transformer

```
<table>
<record>
<field name=FIELD_NAME type="java.lang.String">FIELD_VALUE</field>
</record>
</table>
```

NOTE: The result contains a record node for each Map contained in the source value.

cache: Unexpected program error: java.lang.NullPointerException

### ***Xml To Maps Transformer***

Converts XML to a List of Map objects. The Map List is the same as what you get from using the default ResultSetHandler. The XML schema format is provided in the documentation.

cache: Unexpected program error: java.lang.NullPointerException

### ***Last Record Map Lookup***

Looks up the last Map from a List of Map objects, and then looks up a particular entry in that Map. The key used for lookup is set on the idField property. Since Maps are equivalent to database records, the purpose of this transformer in a JDBC context is to find the last record ID in a batch.

#### Attributes of <last-record-map-lookup...>

Name	Type	Required	Default	Description
idField	string	no		Name of the key (database column) that identifies the ID column.

cache: Unexpected program error: java.lang.NullPointerException

### ***Last Record Map Lookup***

Looks up the last Map from a List of Map objects, and then looks up a particular entry in that Map. The key used for lookup is set on the idField property. Since Maps are equivalent to database records, the purpose of this transformer in a JDBC context is to find the last record ID in a batch.

#### Attributes of <last-record-map-lookup...>

Name	Type	Required	Default	Description
idField	string	no		Name of the key (database column) that identifies the ID column.

cache: Unexpected program error: java.lang.NullPointerException

### ***Maps To Csv Transformer***

Converts a List of Map objects to a CSV file. The Map List is the same as what you get from using the default ResultSetHandler.

#### Attributes of <maps-to-csv-transformer...>

Name	Type	Required	Default	Description
delimiter	string	no		Delimiter used in CSV file. Default is comma.
mappingFile	string	no		Name of the "mapping file" used to describe the CSV file. See <a href="http://flatpack.sourceforge.net">http://flatpack.sourceforge.net</a> for details.

ignoreFirstRecord	boolean	no		Whether to ignore the first record. If the first record is a header, you should ignore it.
qualifier	string	no		The character used to escape text that contains the delimiter.

cache: Unexpected program error: java.lang.NullPointerException

### Csv To Maps Transformer

Converts a CSV file to a List of Map objects. The Map List is the same as what you get from using the default ResultSetHandler.

#### Attributes of <csv-to-maps-transformer...>

Name	Type	Required	Default	Description
delimiter	string	no		Delimiter used in CSV file. Default is comma.
mappingFile	string	no		Name of the "mapping file" used to describe the CSV file. See <a href="http://flatpack.sourceforge.net">http://flatpack.sourceforge.net</a> for details.
ignoreFirstRecord	boolean	no		Whether to ignore the first record. If the first record is a header, you should ignore it.
qualifier	string	no		The character used to escape text that contains the delimiter.

cache: Unexpected program error: java.lang.NullPointerException

### Resultset To Maps Transformer

Transforms a `java.sql.ResultSet` to a List of Map objects just like the default ResultSetHandler. Useful with Oracle stored procedures that return cursors (ResultSets).

Your Rating: 

Results:  0 rates

## JDBC Transport Performance Benchmark Results

### JDBC Transport Performance Benchmark Results

This page describes the performance benchmark results for the Mule Enterprise JDBC transport.

#### Configuration

Mule	Enterprise 1.6 (default 512mb max heap size)
JDK	1.5.0.11
OS	Red Hat Enterprise 4.0
Mule CPUs	4-CPU Dell
Database	Oracle 10g (separate 4-CPU host)
Mule Configuration	See Examples in \$MULE_HOME/examples/jdbc. Used "Simple ETL" use case for this benchmark

#### Scenario Details

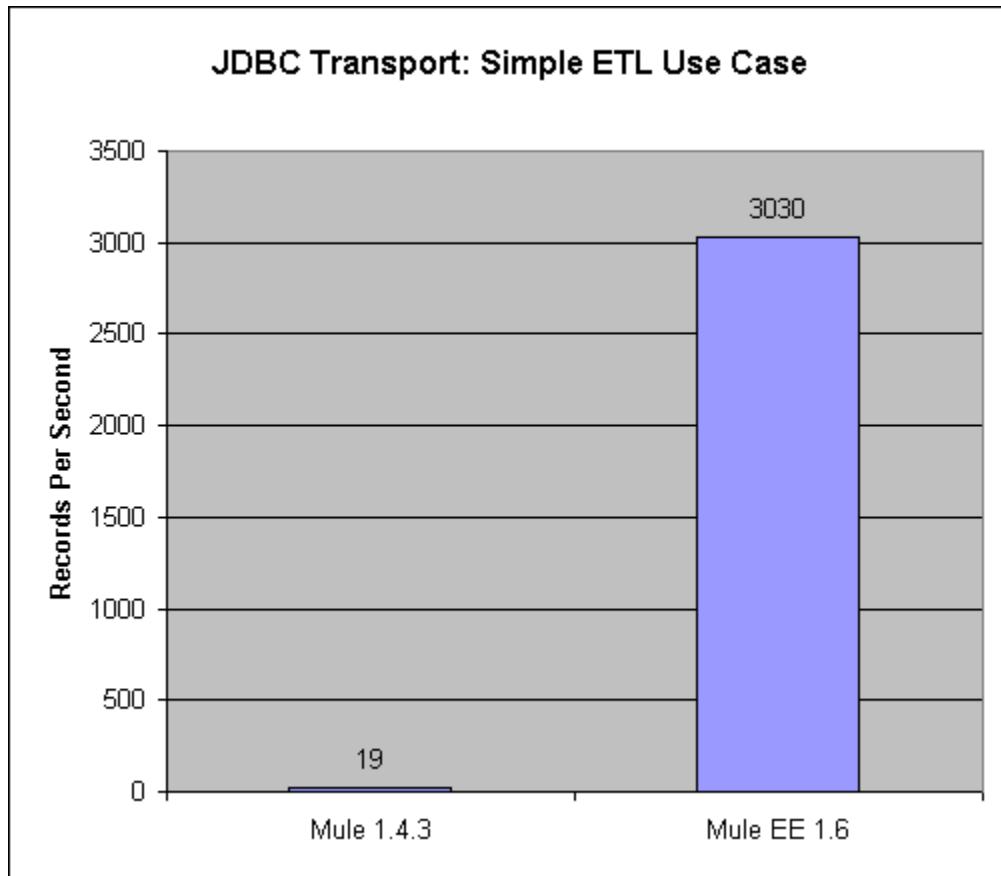
The ETL use case from the Examples directory was used for this benchmark. This example retrieves records from a table called `mule_source` and inserts them into a table called `mule_target`.

The scenario involves processing a backlog of 10 million records in the `mule_source` table. Records are read from the source table once every 1 second, at a batch size of 3000 records per read and 3000 records per commit.

#### Results

Mule took 55 minutes to complete processing of the 10 million record backlog. Therefore, with this configuration, the Mule Enterprise JDBC Transport could move over 10 million records an hour.

## **Comparison to Mule Community Edition**



Your Rating: 5 stars

Results: 0 rates

## **Jetty Transport Reference**

### **Jetty Transport**

[ [Connector](#) ] [ [Endpoints](#) ]

The Jetty transport provides support for exposing services over HTTP by embedding a light-weight Jetty server. The Jetty SSL Transport works the same way but over SSL. You can only define inbound endpoints with this transport.

The Javadoc for this transport can be found [here](#).

cache: Unexpected program error: java.lang.NullPointerException

### **Connector**

Allows Mule to expose Mule Services over HTTP using a Jetty HTTP server. A single Jetty server is created for each connector instance. One connector can serve many endpoints. Users should rarely need to have more than one Jetty connector. The Jetty connector can be configured using a Jetty XML config file, but the default configuration is sufficient for most scenarios.

#### **Attributes of <connector...>**

Name	Type	Required	Default	Description
configFile	string	no		The location of the Jetty config file to configure this connector with.
useContinuations	boolean	no		Whether to use continuations to free up connections in high load situations.

## **Child Elements of <connector...>**

Name	Cardinality	Description
------	-------------	-------------

For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jetty="http://www.mulesoft.org/schema/mule/jetty"
      xsi:schemaLocation="http://www.mulesoft.org/schema/mule/jetty
      http://www.mulesoft.org/schema/mule/jetty/3.0/mule-jetty.xsd http://www.mulesoft.org/schema/mule/core
      http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">
    <jetty:connector name="httpConnector" useContinuations="true" />
    ...

```

## **Endpoints**

Jetty endpoints are configured the same way as [HTTP endpoints](#). Note that only inbound endpoints can use the Jetty transport.

For example:

```
<jetty:endpoint name="serverEndpoint" host="localhost" port="60203" path="services/Foo" synchronous=
  "false" />
<model name="main">
  <service name="testComponent">
    <inbound>
      <inbound-endpoint ref="serverEndpoint" />
    </inbound>
    <test:component appendString="Received" />
  </service>
</model>
```

Your Rating: 

Results:  0 rates

## **Jetty SSL Transport**

### **Jetty SSL Transport**

The Jetty SSL transport works exactly the same way as the [HTTPS Transport Reference](#) with one additional optional attribute, `configFile`, which allows you to specify the location of the Jetty config file to configure this connector with.

For example, the following configuration specifies the HTTPS and Jetty-SSL connectors:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:https="http://www.mulesoft.org/schema/mule/https"
      xmlns:jetty="http://www.mulesoft.org/schema/mule/jetty-ssl"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/test/3.0/mule-test.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd

          http://www.mulesoft.org/schema/mule/https
          http://www.mulesoft.org/schema/mule/https/3.0/mule-https.xsd
          http://www.mulesoft.org/schema/mule/jetty-ssl
          http://www.mulesoft.org/schema/mule/jetty-ssl/3.0/mule-jetty-ssl.xsd">

    <https:connector name="httpConnector">
        <https:tls-client path="clientKeystore" storePassword="mulepassword" />
        <https:tls-key-store path="serverKeystore" keyPassword="mulepassword" storePassword="mulepassword" />
        <https:tls-server path="trustStore" storePassword="mulepassword" />
    </https:connector>
    <jetty:connector name="jettyConnector">
        <jetty:tls-client path="clientKeystore" storePassword="mulepassword" />
        <jetty:tls-key-store path="serverKeystore" keyPassword="mulepassword" storePassword="mulepassword" />
        <jetty:tls-server path="trustStore" storePassword="mulepassword" />
    </jetty:connector>
    <https:endpoint name="clientEndpoint" host="localhost" port="60202" synchronous="true" connector-ref="httpConnector" />
    <model name="main">
        <custom-service name="testComponent" class="org.mule.tck.testmodels.mule.TestSedaService">
            <inbound>
                <jetty:inbound-endpoint host="localhost" port="60202" synchronous="true" connector-ref="jettyConnector" />
            </inbound>
            <test:component appendString="Received" />
        </custom-service>
    </model>
</mule>

```

If you do not need this level of security, you can use the Jetty Transport Reference instead.

Your Rating:  Results:  0 rates

## JMS Transport Reference

### JMS Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Example Configurations](#) ] [ [Vendor-specific Configuration](#) ] [ [Reference](#) ] [ [Notes](#) ]

#### Introduction

JMS (Java Message Service) is a widely-used API for [Message Oriented Middleware](#). It allows communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.

JMS supports two models for messaging:

**Queues** - point-to-point

**Topics** - publish and subscribe

Mule's JMS transport allows you to easily send and receive messages to queues and topics for any message service which implements the JMS

specification.

## Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Art
JMS	JavaDoc SchemaDoc	✓	✓	✓	✓ (client ack, local, XA)	✗	✓	one-way, request-response	one-way	org.mule.transport.jms

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in Maven

## Namespace and Syntax

XML namespace:

```
xmlns:jms "http://www.mulesoft.org/schema/mule/jms"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd
```

Connector syntax:

```
<jms:connector name="myConnector" specification="1.1" connectionFactory-ref="myConnectionFactory"  
username="myuser" password="mypass"/>
```

Endpoint syntax:

```
<jms:outbound-endpoint queue="my.queue"/>  
<jms:inbound-endpoint topic="my.topic"/>
```

## Considerations

In the point-to-point or queuing model, a sender posts messages to a particular queue and a receiver reads messages from the queue. Here, the sender knows the destination of the message and posts the message directly to the receiver's queue. It is characterized by the following:

- Only one consumer gets the message
- The producer does not have to be running at the time the consumer consumes the message, nor does the consumer need to be running

at the time the message is sent

- Every message successfully processed is acknowledged by the consumer

The publish/subscribe model supports publishing messages to a particular message topic. Subscribers may register interest in receiving messages on a particular message topic. In this model, neither the publisher nor the subscriber know about each other. A good analogy for this is an anonymous bulletin board. The following are characteristics of this model:

- Multiple consumers (or none) will receive the message
- There is a timing dependency between publishers and subscribers. The publisher has to create a message topic for clients to subscribe.
- The subscriber has to remain continuously active to receive messages, unless it has established a durable subscription. In that case, messages published while the subscriber is not connected will be redistributed whenever it reconnects.

## Features

- Supports both versions of the JMS specification: 1.0.2b and 1.1
- Supports queues and topics, durable or non-durable subscriptions
- ConnectionFactory and Queues/Topics can be looked up via JNDI
- Supports local (JMS), distributed (XA), and multi-resource  transactions
- Tested with a variety of JMS providers
- Vendor-specific configuration available for popular providers

To see JMS with Mule in action (using ActiveMQ), take a look at the [Error Handler Example](#) (available in the full Mule distribution).



### WebSphere MQ

Mule Enterprise includes an enhanced transport for WebSphereMQ which is recommended if you are using WebSphereMQ as your JMS provider. 



### Mule MQ

If you are trying to select a JMS provider for your Mule application, you might consider [Mule MQ](#) which is an enterprise-class JMS implementation officially supported by MuleSoft. 

## Usage

### *Declaring the Namespace*

To use the JMS transport, you must first declare the JMS namespace in the header of your Mule configuration file. You can then configure the JMS connector and endpoints.

#### JMS namespace

```
<mule ...cut...
  xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
  xsi:schemaLocation=" ...cut...
    http://www.mulesoft.org/schema/mule/jms
    http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">
```

### *Configuring the Connector*

There are several attributes available on the connector, most of which are optional. Refer to the schema documentation below for complete information.

### Connector attributes

```
<jms:connector name="myConnector"
    acknowledgementMode="DUPS_OK_ACKNOWLEDGE"
    clientId="myClient"
    durable="true"
    noLocal="true"
    persistentDelivery="true"
    maxRedelivery="5"
    cacheJmsSessions="true"
    eagerConsumer="false"
    specification="1.1"
    numberOfConsumers="7"
    username="myuser"
    password="mypass" />
```

### Configuring the ConnectionFactory

One of the most important attributes is `connectionFactory-ref`. This is a reference to the `ConnectionFactory` object which will create new connections for your JMS provider. The object must implement the interface `javax.jms.ConnectionFactory`.

### ConnectionFactory

```
<spring:bean name="connectionFactory" class="com.foo.FooConnectionFactory"/>

<jms:connector name="jmsConnector1" connectionFactory-ref="connectionFactory" />
```

There are also a few attributes which allow you to look up the `ConnectionFactory` from a JNDI Context:

### ConnectionFactory from JNDI

```
<jms:connector name="jmsConnector"
    jndiInitialFactory="com.sun.jndi.ldap.LdapCtxFactory"
    jndiProviderUrl="ldap://localhost:10389/"
    jndiProviderProperties-ref="providerProperties"
    connectionFactoryJndiName="cn=ConnectionFactory,dc=example,dc=com" />
```

### Configuring the Endpoints

#### Queues

```
<jms:inbound-endpoint queue="my.queue" />

<jms:outbound-endpoint queue="my.queue" />
```

#### Topics

```
<jms:inbound-endpoint topic="my.topic" />

<jms:outbound-endpoint topic="my.topic" />
```

By default, Mule's subscription to a topic is non-durable (i.e., it will only receive messages while connected to the topic). You can make topic subscriptions durable by setting the `durable` attribute on the connector.

When using a durable subscription, the JMS server requires a durable name to identify each subscriber. By default, Mule generates the durable name in the format `mule.<connector name>.<topic name>`. If you want to specify the durable name yourself, you can do so using the `durableName` attribute on the endpoint.

## Durable Topic

```
<jms:connector name="jmsTopicConnector" durable="true"/>  
  
<jms:inbound-endpoint topic="some.topic" durableName="sub1" />  
<jms:inbound-endpoint topic="some.topic" durableName="sub2" />  
<jms:inbound-endpoint topic="some.topic" durableName="sub3" />
```



### Number of consumers

In the case of a topic, the number of consumers on the endpoint will be set to one. You can override this by setting `numberOfConcurrentTransactedReceivers` or `numberOfConsumers` on the connector.

## Transformers

The default transformers applied to JMS endpoints are as follows:

inbound = `JMSMessageToObject`

response = `ObjectToJMSMessage`

outbound = `ObjectToJMSMessage`

These will automatically transform to/from the standard JMS message types:

```
javax.jms.TextMessage - java.lang.String  
javax.jms.ObjectMessage - java.lang.Object  
javax.jms.BytesMessage - byte[]  
javax.jms.MapMessage - java.util.Map  
javax.jms.StreamMessage - java.io.InputStream
```

## Looking Up JMS Objects from JNDI

If you have configured a JNDI context on the connector, you can also look up queues/topics via JNDI using the `jndiDestinations` attribute. If a queue/topic cannot be found via JNDI, it will be created using the existing JMS session unless you also set the `forceJndiDestinations` attribute.

There are two different ways to configure the JNDI settings:

1. Using connector properties (deprecated):

```
<jms:connector name="jmsConnector"  
    jndiInitialFactory="com.sun.jndi.ldap.LdapCtxFactory"  
    jndiProviderUrl="ldap://localhost:10389/  
    connectionFactoryJndiName="cn=ConnectionFactory,dc=example,dc=com"  
    jndiDestinations="true"  
    forceJndiDestinations="true" />
```

2. Using a `JndiNameResolver`

A `JndiNameResolver` defines a strategy for lookup objects by name using JNDI. The strategy contains a `lookup` method that receives a name and returns the object associated to that name.

At the moment, there are two simple implementations of that interface:

**SimpleJndiNameResolver**: uses a JNDI context instance to search for the names. That instance is maintained opened during the full lifecycle of the name resolver.

**CachedJndiNameResolver**: uses a simple cache in order to store previously resolved names. A JNDI context instance is created for each request that is sent to the JNDI server and then the instance is freed. The cache can be cleaned up restarting the name resolver.

Default JNDI name resolver example: define the name resolver using the `default-jndi-name-resolver` tag and then add the appropriate properties to it.

```

<jms:activemq-connector name="jmsConnector"
    jndiDestinations="true"
    connectionFactoryJndiName="ConnectionFactory">
    <jms:default-jndi-name-resolver
        jndiInitialFactory="org.apache.activemq.jndi.ActiveMQInitialContextFactory"
        jndiProviderUrl="vm://localhost?broker.persistent=false&broker.useJmx=false"
        jndiProviderProperties-ref="providerProperties"/>
</jms:activemq-connector>

```

**Custom JNDI name resolver example:** define the name resolver using the custom-jndi-name-resolver tag, then add the appropriate property values using the Spring's property format.

```

<jms:activemq-connector name="jmsConnector"
    jndiDestinations="true"
    connectionFactoryJndiName="ConnectionFactory">
    <jms:custom-jndi-name-resolver class="org.mule.transport.jms.jndi.CachedJndiNameResolver">
        <spring:property name="jndiInitialFactory" value=
"org.apache.activemq.jndi.ActiveMQInitialContextFactory"/>
        <spring:property name="jndiProviderUrl"
            value="vm://localhost?broker.persistent=false&broker.useJmx=false"/>
        <spring:property name="jndiProviderProperties" ref="providerProperties"/>
    </jms:custom-jndi-name-resolver>
</jms:activemq-connector>

```

### Changes in JmsConnector

There are some property changes in the JmsConnector definition. Some properties are now deprecated as they should be defined in a JndiNameResolver and then using that JndiNameResolver in the JmsConnector.

Deprecated properties in JmsConnector:

- jndiContext
- jndiInitialFactory
- jndiProviderUrl
- jndiProviderProperties-ref

Added property:

- jndiNameResolver: used to set a proper JndiNameResolver. Can be set using the default-jndi-name-resolver or custom-jndi-name-resolver tags inside the JmsConnector definition.

### JMS Selectors

You can set a JMS selector as a filter on an inbound endpoint. The JMS selector simply sets the filter expression on the JMS consumer.

JMS Selector
<pre>&lt;jms:inbound-endpoint queue="important.queue"&gt;     &lt;jms:selector expression="JMSPriority=9"/&gt; &lt;/jms:inbound-endpoint&gt;</pre>

### JMS Header Properties

Once a JMS message is received by Mule, the standard JMS headers such as `JMSCorrelationID` and `JMSRedelivered` are made available as properties on the `MuleMessage` object.

Retrieving JMS Headers
<pre>String corrId = (String) muleMessage.getProperty("JMSCorrelationID"); boolean redelivered = muleMessage.getBooleanProperty("JMSRedelivered");</pre>

You can access any custom header properties on the message in the same way.



## Configuring Transactional Polling

The Enterprise version of the JMS transport can be configured for transactional polling using the `TransactedPollingJmsMessageReceiver`.

### Transactional Polling

```
<jms:connector ...cut...>
    <service-overrides transactedMessageReceiver=
"com.mulesoft.mule.transport.jms.TransactedPollingJmsMessageReceiver" />
</jms:connector>

<jms:inbound-endpoint queue="my.queue">
    <properties>
        <spring:entry key="pollingFrequency" value="5000" />
    </properties>
</jms:inbound-endpoint>
```

Each receiver polls with a 5 second interval

## Example Configurations

### Example configuration

```
<mule ...cut...
    xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
    xsi:schemaLocation="...cut...
    http://www.mulesoft.org/schema/mule/jms http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd">

    <spring:bean name="connectionFactory" class="com.foo.FooConnectionFactory"/>

    <jms:connector name="jmsConnector" connectionFactory-ref="connectionFactory" username="myuser"
password="mypass" />

    <flow name="MyFlow">
        <jms:inbound-endpoint queue="in" />
        <component class="com.foo.MyComponent" />
        <jms:outbound-endpoint queue="out" />
    </flow>
</mule>
```

Import the JMS schema namespace

### Example configuration with transactions

```
<mule ...cut...
  xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
  xsi:schemaLocation="...cut...
    http://www.mulesoft.org/schema/mule/jms http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd">

  <spring:bean name="connectionFactory" class="com.foo.FooConnectionFactory"/>

  <jms:connector name="jmsConnector" connectionFactory-ref="connectionFactory" username="myuser"
  password="mypass" />

  <flow name="MyFlow">
    <jms:inbound-endpoint queue="in">
      <jms:transaction action="ALWAYS_BEGIN" />
    </jms:inbound-endpoint>
    <component class="com.foo.MyComponent" />
    <jms:outbound-endpoint queue="out">
      <jms:transaction action="ALWAYS_JOIN" />
    </jms:outbound-endpoint>
  </flow>
</mule>
```

Local JMS transaction

### Example configuration with exception strategy

```
<mule ...cut...
  xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
  xsi:schemaLocation="...cut...
    http://www.mulesoft.org/schema/mule/jms http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd">

  <spring:bean name="connectionFactory" class="com.foo.FooConnectionFactory"/>

  <jms:connector name="jmsConnector" connectionFactory-ref="connectionFactory" username="myuser"
  password="mypass" />

  <flow name="MyFlow">
    <jms:inbound-endpoint queue="in">
      <jms:transaction action="ALWAYS_BEGIN" />
    </jms:inbound-endpoint>
    <component class="com.foo.MyComponent" />
    <jms:outbound-endpoint queue="out">
      <jms:transaction action="ALWAYS_JOIN" />
    </jms:outbound-endpoint>
    <default-exception-strategy>
      <commit-transaction exception-pattern="com.foo.ExpectedExceptionType" />
      <jms:outbound-endpoint queue="dead.letter">
        <jms:transaction action="JOIN_IF_POSSIBLE" />
      </jms:outbound-endpoint>
    </default-exception-strategy>
  </flow>
</mule>
```

Set exception-pattern="\*" to catch all exception types  
Implements a Dead letter queue for erroneous messages

### Example configuration using Service

```
<mule ...cut...
    <spring:bean name="connectionFactory" class="com.foo.FooConnectionFactory"/>

    <jms:connector name="jmsConnector" connectionFactory-ref="connectionFactory" username="myuser"
password="mypass" />

    <model>
        <service name="MyService">
            <inbound>
                <jms:inbound-endpoint queue="in" />
            </inbound>
            <component class="com.foo.MyComponent" />
            <outbound>
                <pass-through-router>
                    <jms:outbound-endpoint queue="out" />
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

New implementations are recommended to use [flows](#), but Mule 2.x users will be more familiar with `<service>`.

## Vendor-specific Configuration

Mule Enterprise includes an [enhanced transport for WebSphereMQ](#) which is recommended if you are using WebSphereMQ as your JMS provider.



Mule MQ is an enterprise-class JMS implementation officially supported by MuleSoft and has tight integration with Mule.



ActiveMQ is also widely-used with Mule and has [simplified configuration](#).

Information for configuring other JMS providers can be found here. Beware that some of this information may be out-of-date.

- FioranoMQ
- HornetQ
- JBoss MQ
- OpenJms
- Open MQ
- Oracle AQ
- SeeBeyond
- SonicMQ
- Sun JMS Grid
- SwiftMQ
- Tibco EMS
- WebLogic JMS

## Reference

### Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

#### Connector

The connector element configures a generic connector for sending and receiving messages over JMS queues.

#### Attributes of `<connector...>`

Name	Type	Required	Description
name	name (no spaces)	yes	

name	name (no spaces)	yes	
dynamicNotification	boolean	no	fa
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	fa
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
createMultipleTransactedReceivers	boolean	no	
numberOfConcurrentTransactedReceivers	integer	no	
connectionFactory-ref	string	no	
redeliveryHandlerFactory-ref	string	no	
acknowledgementMode	AUTO_ACKNOWLEDGE/CLIENT_ACKNOWLEDGE/DUPS_OK_ACKNOWLEDGE	no	All
clientId	string	no	
durable	boolean	no	
noLocal	boolean	no	

persistentDelivery	boolean		no	
honorQosHeaders	boolean		no	
maxRedelivery	integer		no	
cacheJmsSessions	boolean		no	
eagerConsumer	boolean		no	
specification	1.0.2b/1.1		no	1.
username	string		no	
password	string		no	
numberOfConsumers	integer		no	
jndiInitialFactory	string		no	
jndiProviderUrl	string		no	
jndiProviderProperties-ref	string		no	

connectionFactoryJndiName	string	no	
jndiDestinations	boolean	no	
forceJndiDestinations	boolean	no	
disableTemporaryReplyToDestinations	boolean	no	
embeddedMode	boolean	no	fa

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-jndi-name-resolver	0..1	A placeholder for jndi-name-resolver strategy elements.

#### Custom connector

The custom-connector element configures a custom connector for sending and receiving messages over JMS queues.

#### Inbound endpoint

The inbound-endpoint element configures an endpoint on which JMS messages are received.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description

name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
durableName	string	no		(As of 2.2.2) Allows the name for the durable topic subscription to be specified.
queue	string	no		The queue name. This attribute cannot be used with the topic attribute (the two are exclusive).
topic	string	no		The topic name. The "topic:" prefix will be added automatically. This attribute cannot be used with the queue attribute (the two are exclusive).
disableTemporaryReplyToDestinations	boolean	no		If this is set to false (the default), when Mule performs request/response calls a temporary destination will automatically be set up to receive a response from the remote JMS call.

**Child Elements of <inbound-endpoint...>**

Name	Cardinality	Description
mule:response	0..1	
mule:abstract-transaction	0..1	
mule:abstract-xa-transaction	0..1	
mule:abstract-security-filter	0..1	
mule:abstract-filter	0..1	
selector	0..1	

**Outbound endpoint**

The inbound-endpoint element configures an endpoint to which JMS messages are sent.

**Attributes of <outbound-endpoint...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
queue	string	no		The queue name. This attribute cannot be used with the topic attribute (the two are exclusive).
topic	string	no		The topic name. The "topic:" prefix will be added automatically. This attribute cannot be used with the queue attribute (the two are exclusive).
disableTemporaryReplyToDestinations	boolean	no		If this is set to false (the default), when Mule performs request/response calls a temporary destination will automatically be set up to receive a response from the remote JMS call.

**Child Elements of <outbound-endpoint...>**

Name	Cardinality	Description
mule:response	0..1	
mule:abstract-transaction	0..1	
mule:abstract-xa-transaction	0..1	
mule:abstract-security-filter	0..1	
mule:abstract-filter	0..1	
selector	0..1	

## Jmsmessage to object transformer

The jmsmessage-to-object-transformer element configures a transformer that converts a JMS message into an object by extracting the message payload.

### *Child Elements of <jmsmessage-to-object-transformer...>*

Name	Cardinality	Description
------	-------------	-------------

#### Object to jmsmessage transformer

The object-to-jmsmessage-transformer element configures a transformer that converts an object into one of five types of JMS messages, depending on the object passed in: java.lang.String -> javax.jms.TextMessage, byte[] -> javax.jms.BytesMessage, java.util.Map (primitive types) -> javax.jms.MapMessage, java.io.InputStream (or java.util.List of primitive types) -> javax.jms.StreamMessage, and java.lang.Serializable including java.util.Map, java.util.List, and java.util.Set objects that contain serializable objects (including primitives) -> javax.jms.ObjectMessage.

### *Child Elements of <object-to-jmsmessage-transformer...>*

Name	Cardinality	Description
------	-------------	-------------

#### Transaction

The transaction element configures a transaction. Transactions allow a series of operations to be grouped together so that they can be rolled back if a failure occurs. Set the action (such as ALWAYS\_BEGIN or JOIN\_IF\_POSSIBLE) and the timeout setting for the transaction.

### *Child Elements of <transaction...>*

Name	Cardinality	Description
------	-------------	-------------

#### Client ack transaction

The client-ack-transaction element configures a client acknowledgment transaction, which is identical to a transaction but with message acknowledgements. There is no notion of rollback with client acknowledgement, but this transaction can be useful for controlling how messages are consumed from a destination.

### *Child Elements of <client-ack-transaction...>*

Name	Cardinality	Description
------	-------------	-------------

#### Selector

### *Attributes of <selector...>*

Name	Type	Required	Default	Description
expression	string	yes		The expression to search for in the property.

### *Child Elements of <selector...>*

Name	Cardinality	Description
------	-------------	-------------

#### Property filter

The property-filter element configures a filter that allows you to filter messages based on a JMS property.

## XML Schema

Import the XML schema for this module as follows:

```
xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/jms
http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd"
```

Complete schema reference documentation.

## Javadoc

Javadoc for this transport can be found [here](#).

## Maven

If you are using Maven to build your application, use the following groupId/artifactId to include this module as a dependency:

```
<dependency>
<groupId>org.mule.transports</groupId>
<artifactId>mule-transport-jms</artifactId>
</dependency>
```

## Notes

The 1.0.2b specification has the limitation of only supporting queues **or** topics for each ConnectionFactory. If you need both, you will need to configure two separate connectors, one that references a QueueConnectionFactory, and another that references a TopicConnectionFactory. You can then use the connector-ref attribute to disambiguate the endpoints.

### Workaround for 1.0.2b spec.

```
<spring:bean name="queueConnectionFactory" class="com.foo.QueueConnectionFactory"/>
<spring:bean name="topicConnectionFactory" class="com.foo.TopicConnectionFactory"/>

<jms:connector name="jmsQueueConnector" connectionFactory-ref="queueConnectionFactory" />
<jms:connector name="jmsTopicConnector" connectionFactory-ref="topicConnectionFactory" />

<jms:outbound-endpoint queue="my.queue1" connector-ref="jmsQueueConnector"/>
<jms:outbound-endpoint queue="my.queue2" connector-ref="jmsQueueConnector"/>

<jms:inbound-endpoint topic="my.topic" connector-ref="jmsTopicConnector"/>
```

Your Rating: 

Results:  0 rates

## Open MQ Integration

### Open MQ Integration

This page describes how to integrate Mule with the Open MQ JMS broker, the community version of Sun Java System Message Queue. It includes examples for configuring a direct connection and a JNDI connection.

#### Direct Connection

To create a direct connection, simply specify the connection factory and JMS connector as follows in your Mule configuration file:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <spring:bean name="connectionFactory" class="com.sun.messaging.ConnectionFactory"/>

    <jms:connector name="JMSConnector"
                   connectionFactory-ref="connectionFactory"
                   specification="1.1"/>

```

### **JNDI Connection**

To create a JNDI connection, take the following steps:

1. Configure Open MQ using the instructions from Sun [here](#). You must use the Open MQ or Message Queue Administration Console to define the connectionFactory (Factory type topic/queue connection factory) and to configure any Open MQ JMS endpoints you will specify in your Mule configuration file.
2. Configure the namespaces, JMS connector, Spring beans, and endpoints in your Mule configuration file as follows, ensuring that you include all the tags shown below:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector" connectionFactory-ref="openMQ" specification="1.1">
        <spring:property name="jmsSupport" ref="jndiJmsSupport" />
    </jms:connector>

    <spring:beans>
        <spring:bean name="jndiJmsSupport" class="org.mule.transport.jms.Jms102bSupport">
            <spring:constructor-arg ref="jmsConnector" />
        </spring:bean>

        <spring:bean name="context" class="javax.naming.InitialContext">
            <spring:constructor-arg type="java.util.Hashtable">
                <spring:props>
                    <spring:prop key="java.naming.factory.initial">
com.sun.jndi.fscontext.RefFSContextFactory
                    </spring:prop>
                    <spring:prop key="java.naming.provider.url">file:///C:/pawan/openMQ/mq
                </spring:prop>
                </spring:props>
            </spring:constructor-arg>
        </spring:bean>

        <spring:bean name="openMQ" class="org.springframework.jndi.JndiObjectFactoryBean">
            <spring:property name="jndiName" value="MyTopicConnectionFactory" />
            <spring:property name="jndiEnvironment">
                <spring:props>
                    <spring:prop key="java.naming.factory.initial">
com.sun.jndi.fscontext.RefFSContextFactory
                    </spring:prop>
                    <spring:prop key="specifications">1.1</spring:prop>
                    <spring:prop key="java.naming.provider.url">file:///C:/Temp</spring:prop>
                </spring:props>
            </spring:property>
        </spring:bean>
    </spring:beans>

    <endpoint name="MyEndPoint" address="jms://topic:my_topic" connector-ref="jmsConnector"/>
    ...
</mule>

```

3. You need to put imq.jar and fscontext.jar from the \$OPENMQ\_HOME/lib on your classpath.

Your Rating: 

Results:  1 rates

## Fiorano Integration

### Configuring the JMS Connector for FioranoMQ 2007

FioranoMQ is a High Performance Enterprise Communication Backbone. To use the FioranoMQ connector, you configure the JMS namespace and connector as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

<jms:connector name="FioranoJMSConnector"
               connectionFactoryJndiName="PrimaryCF"
               jndiInitialFactory="fiorano.jms.runtime.naming.FioranoInitialContextFactory"
               specification="1.1"
               jndiProviderUrl="http://localhost:1856"
               username="anonymous"
               password="anonymous">

    <!-- A few optional values for the factory -->
    <spring:property key="connectionFactoryProperties">
        <spring:map>
            <spring:entry key="clientID" value="sampleClientID"/>
            <spring:entry key="ConnectURL" value="http://localhost:1856"/>
            <spring:entry key="BackupConnectURLs" value="http://localhost:1956"/>
        </spring:map>
    </spring:property>
</jms:connector>
...

```

You will need the following jars on your classpath:

- FioranoMQ2007/fmq/lib/client/all/fmq-client.jar
- FioranoMQ2007/framework/lib/all/fiorano-framework.jar

### **Sample Usage**

The following steps illustrate modifying the "Echo" sample shipped with Mule. Instead of using System.out in the outbound router, we will write the output onto a Topic in FioranoMQ using the above configuration.

Modify the outbound router in the echo-config.xml under examples\echo\conf to use a Topic:

```
<jms:outbound-endpoint topic="muleTopic" />
```

Start the durable connection sample available in FioranoMQ from a command prompt in fmq/samples/PubSub/DurableSubscribers as shown below:

```
runClient DurableSubscriber -clientid sampleClientID -topicName muleTopic
```

Now on starting Mule with the above echo-config.xml file we can push messages onto the topic and consequently to the subscriber. The durable connection property can also be tested by killing the subscriber, pumping in more messages and then again starting the subscriber.

Your Rating:  Results:  0 rates

## **JBoss Jms Integration**

### **JBoss JMS Integration**

You configure a JBoss JMS connector for Mule as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:jbossts="http://www.mulesoft.org/schema/mule/jbossts"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/jbossts
          http://www.mulesoft.org/schema/mule/jbossts/3.0/mule-jbossts.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/stdio/3.0/mule-stdio.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/spring-beans-3.0.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <jbossts:transaction-manager/>

    <configuration>
        <default-dispatcher-threading-profile maxThreadsActive="50" maxThreadsIdle="25"
            threadTTL="60000"/>
        <default-receiver-threading-profile maxThreadsActive="50" maxThreadsIdle="25"
            threadTTL="60000"/>
        <default-service-threading-profile maxThreadsActive="50" maxThreadsIdle="25"
            threadTTL="60000"/>
    </configuration>

    <jms:connector name="jms-connector" jndiInitialFactory="org.jnp.interfaces.NamingContextFactory"
        jndiProviderUrl="jnp://127.0.0.1:1099"
        connectionFactoryJndiName="java:/QueueConnectionFactory" jndiDestinations="true"
        forceJndiDestinations="true" createMultipleTransactedReceivers="true"
        numberOfConcurrentTransactedReceivers="10" disableTemporaryReplyToDestinations="true">
        <!--retry:forever-policy frequency="2000"-->
    </jms:connector>

    <stdio:connector name="stdioConnector" promptMessage="Please enter message: " outputMessage=
    "Received message: " messageDelayTime="3000"/>

    <jms:object-to-jmsmessage-transformer name="ObjectToJMSMessageTransformer"/>

    <jms:jmsmessage-to-object-transformer name="JMSMessageToObjectTransformer"/>

    <model>
        <service name="stdioToQueue">
            <inbound>
                <stdio:inbound-endpoint system="IN" exchange-pattern="request-response"/>
            </inbound>
            <outbound>
                <pass-through-router>
                    <jms:outbound-endpoint queue="queue/testQueue" exchange-pattern="request-response">
                </pass-through-router>
            </outbound>
        </service>

        <service name="queueToStdio">
            <inbound>
                <jms:inbound-endpoint queue="queue/testQueue" exchange-pattern="request-response"/>
            </inbound>
            <outbound>
                <pass-through-router>

```

```
<stdio:outbound-endpoint system="OUT" exchange-pattern="request-response" />
</pass-through-router>
</outbound>
</service>
```

```
</model>
</mule>
```

The JNDI provider and JBoss properties are specified in Spring.



If you use user credentials to connect to JBoss MQ, make sure that the user has the 'guest' role assigned to it. This will ensure that there are no issues if temporary topics or queues are used.

Your Rating: 5

Results: 0 rates

## SeeBeyond JMS Server Integration

### SeeBeyond JMS Server Integration

The following configuration is for the SeeBeyond ICAN IQManager JMS Server. Note the values in [ ] (square brackets), which should be replaced by values relevant to your installation. Port 18006 is the default, which you can change in the SeeBeyond designer.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
                   jndiInitialFactory="com.stc.is.naming.NamingContextFactory"
                   jndiProviderUrl="[ServerName]:18006"
                   connectionFactoryJndiName="/jms/connectionfactory/queue/[LogicalHostName]_[JMS
iqManager Name]"/>
    </jms:connector>
    ...

```

For a topic, the connectionFactoryJndiName would be /jms/connectionfactory/topic/[LogicalHostName]\_[JMS iqManager Name].

You will need the following files from the Java API Kit on your classpath:

- com.stc.jmsis.jar
- fscontext.jar
- providerutil.jar
- jms.jar
- jta.jar
- log4j.jar
- log4j.properties

Your Rating: 5

Results: 0 rates

## Sun JMS Grid Integration

### Sun JMS Grid Integration

The following configuration demonstrates how to configure the JMS Transport Reference in Mule to use the Sun JMS Grid server.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector" specification="1.1"
                   connectionFactoryJndiName="QueueConnectionFactory"
                   jndiInitialFactory="com.spirit.directory.SpiritVMDirectoryContextFactory"
                   <spring:property name="jndiProviderProperties">
                     <spring:map>
                       <spring:entry key="driverName" value="WMSEmbedded"/>
                     </spring:map>
                   </spring:property>
    </jms:connector>
    ...
  
```

Your Rating: 

Results:  0 rates

## Tibco EMS Integration

### TIBCO EMS Integration

The following example demonstrates how to configure Mule to use the TIBCO Enterprise Message Server (EMS) with authentication in place.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
                   jndiProviderUrl="tibjmsnaming://host:port"
                   connectionFactoryJndiName="QueueConnectionFactory"
                   username="username"
                   password="password"
                   jndiDestinations="true"
                   forceJndiDestinations="true"
                   jndiInitialFactory="com.tibco.tibjms.naming.TibjmsInitialContextFactory"
                   specification="1.1">
      <spring:property name="jndiProviderProperties">
        <spring:map>
          <spring:entry key="java.naming.security.principal" value="jndiUsername"/>
          <spring:entry key="java.naming.security.credentials" value="jndiPassword"/>
        </spring:map>
      </spring:property>
    </jms:connector>
    ...
  
```

Note that when you use `tibjmsnaming` as the protocol in your `jndiProviderUrl`, you can also specify TCP or SSL with the JNDI property

```
com.tibco.tibjms.naming.security_protocol.
```

For XA transactions, you must create an XA Connection Factory from the TIBCO administration-tool (tibemsadmin) as follows:

```
> create factory XAQueueConnectionFactory xaqueue url=tcp://7222
```

Your Rating: ★★★★★

Results: ★★★★★ 0 rates

## SonicMQ Integration

### SonicMQ Integration

The following configuration was tested with versions 6.1 and 7.0 of SonicMQ.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsSonicMQConnector"
        jndiInitialFactory="com.sonicsw.jndi.mfcontext.MFContextFactory"
        specification="1.1"
        connectionFactoryJndiName="sonic-cf"
        jndiProviderUrl="tcp://localhost:2506"
        username="Administrator"
        password="Administrator">

        <spring:property key="connectionFactoryProperties">
            <spring:map>
                <spring:entry key="clientID" value="clientIDString"/>
                <spring:entry key="connectID" value="connectIDString"/>
                <spring:entry key="connectionURLs" value="somURLStrings here"/>
                <spring:entry key="defaultUser" value="userString"/>
                <spring:entry key="defaultPassword" value="passwordString"/>
                <spring:entry key="prefetchCount" value="10"/>
                <spring:entry key="prefetchThreshold" value="10"/>
                <spring:entry key="faultTolerant" value="true"/>
                <spring:entry key="persistentDelivery" value="true"/>
                <spring:entry key="loadBalancing" value="true"/>
                <spring:entry key="sequential" value="false"/>
            </spring:map>
        </spring:property>

        <spring:property key="jndiProviderProperties">
            <spring:map>
                <spring:entry key="com.sonicsw.jndi.mfcontext.domain" value="Domain1"/>
                <spring:entry key="java.naming.security.principal" value="Administrator"/>
                <spring:entry key="java.naming.security.credentials" value="Administrator"/>
                <!-- optional, default is 30sec -->
                <spring:entry key="com.sonicsw.jndi.mfcontext.idleTimeout" value="5000"/>
            </spring:map>
        </spring:property>

    </jms:connector>
    ...
```

Your Rating:  5

Results:  0 rates

## OpenJMS Integration

### OpenJMS Integration

The following example configuration describes how to configure a Mule JMS connector for OpenJMS. You will probably need to change the connectionFactoryJndiName to one that is configured from your OpenJMS configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
                   jndiInitialFactory="org.exolab.jms.jndi.InitialContextFactory"
                   jndiProviderUrl="tcp://localhost:3035"
                   connectionFactoryJndiName="QueueConnectionFactory"/>
    ...

```

Your Rating:  5

Results:  0 rates

## HornetQ Integration

### HornetQ integration

To integrate with HornetQ you have to configure a connection factory in Spring and reference it when creating the JMS connector. The configuration differs depending if you want to connect to a stand-alone instance of HornetQ or one that is set up as a cluster.

#### *Connecting to a single HornetQ instance*

```

<mule xmlns="http://www.mulesource.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesource.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.mulesource.org/schema/mule/core
          http://www.mulesource.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesource.org/schema/mule/jms
          http://www.mulesource.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/spring-beans-3.0.xsd">

    <spring:bean name="connectionFactory" class="org.hornetq.jms.client.HornetQConnectionFactory">
        <spring:constructor-arg>
            <spring:bean class="org.hornetq.api.core.TransportConfiguration">
                <spring:constructor-arg value=
"org.hornetq.core.remoting.impl.netty.NettyConnectorFactory"/>
                <spring:constructor-arg>
                    <spring:map key-type="java.lang.String" value-type="java.lang.Object">
                        <spring:entry key="port" value="5445"></spring:entry>
                    </spring:map>
                </spring:constructor-arg>
            </spring:bean>
            <spring:constructor-arg>
        </spring:bean>
    </spring:constructor-arg>
</spring:bean>

    <jms:connector name="hornetq-connector" username="guest" password="guest"
                  specification="1.1" connectionFactory-ref="connectionFactory" />
</mule>

```

### **Connecting to a HornetQ cluster**

```

<mule xmlns="http://www.mulesource.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jms="http://www.mulesource.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.mulesource.org/schema/mule/core
          http://www.mulesource.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesource.org/schema/mule/jms
          http://www.mulesource.org/schema/mule/jms/3.0/mule-jms.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/spring-beans-3.0.xsd">

    <spring:bean name="connectionFactory" class="org.hornetq.jms.client.HornetQConnectionFactory">
        <spring:property name="discoveryAddress" value="231.7.7.7"/>
        <spring:property name="discoveryPort" value="9876"/>
        <spring:property name="discoveryRefreshTimeout" value="1000"/>
        <!-- If you want the client to failover when its server is cleanly shutdown -->
        <spring:property name="failoverOnServerShutdown" value="true"/>
        <!-- period in milliseconds between subsequent reconnection attempts. The default value is
        2000 milliseconds-->
        <spring:property name="retryInterval" value="1000"/>
        <!-- allows you to implement an exponential backoff between retry attempts -->
        <spring:property name="retryIntervalMultiplier" value="2.0"/>
        <!-- A value of -1 signifies an unlimited number of attempts. The default value is 0. -->
        <spring:property name="reconnectAttempts" value="-1"/>
        <!-- interesting for blocked receivers: If you're using JMS it's defined by the
        ClientFailureCheckPeriod attribute on a HornetQConnectionFactory instance -->
        <spring:property name="clientFailureCheckPeriod" value="1000"/>
        <!-- allow the client to loadbalance when creating multiple sessions from one sessionFactory
    -->
        <spring:property name="connectionLoadBalancingPolicyClassName" value=
        "org.hornetq.api.core.client.loadbalance.RandomConnectionLoadBalancingPolicy"/>
    </spring:bean>

    <jms:connector name="hornetq-connector" username="guest" password="guest"
                  specification="1.1" connectionFactory-ref="connectionFactory"/>
</mule>

```

## Required libraries

You need to have the following jars on the classpath

- hornetq-core-client.jar
- hornetq-jms.jar
- netty.jar

Your Rating: 

Results:  0 rates

## WebLogic JMS Integration

### WebLogic JMS Integration

[ Configuration for WebLogic 8.x and Earlier ] [ Configuration for WebLogic 9.x ] [ Configuring Security ]

If you are using a WebLogic version prior to 10.3, copy the `weblogic.jar` file to `$MULE_HOME/lib/user`. For WebLogic versions 10.3 and up, you must generate a `wlfullclient.jar` file from your WebLogic installation as follows:

1. Go to the `server/lib` directory of your WebLogic installation.
2. Run this command to generate the client JAR:

```
java -jar wljarbuilder.jar
```

3. Copy the generated `wlfullclient.jar` file to the `$MULE_HOME/lib/user` directory.



### JNDI destinations syntax

If Mule fails to look up topics or queues in WebLogic's JNDI, but the JNDI tree lists them as available, try replacing JNDI subcontext delimiters with dots, so `tracker/topic/PriceUpdates` becomes `tracker.topic.PriceUpdates`.

## Configuration for WebLogic 8.x and Earlier

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
        jndiProviderUrl="t3://localhost:7001"
        connectionFactoryJndiName="javax.jms.QueueConnectionFactory"
        jndiDestinations="true"
        forceJndiDestinations="true"
        jndiInitialFactory="weblogic.jndi.WLInitialContextFactory"
        specification="1.0.2b"/>
```

## Configuration for WebLogic 9.x

For WebLogic 9.x, the configuration is almost the same. The only differences are:

- Supported JMS specification level is 1.1 (1.0.2b should still work, however)
- The unified JMS connection factory can be used as a result of the above. The following example demonstrates using the default factories available out of the box.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
        jndiProviderUrl="t3://localhost:7001"
        connectionFactoryJndiName="weblogic.jms.ConnectionFactory"
        jndiDestinations="true"
        forceJndiDestinations="true"
        jndiInitialFactory="weblogic.jndi.WLInitialContextFactory"
        specification="1.1"/>
```

## Configuring Security

The following example shows how to configure security on WebLogic 9.x using Spring:

```

<spring:bean name="jmsProperties" class="java.util.HashMap">
    <spring:constructor-arg>
        <spring:map>
            <spring:entry key="java.naming.security.principal" value="secureUser"/>
            <spring:entry key="java.naming.security.credentials" value="password"/>
            <spring:entry key="java.naming.security.authentication" value="simple"/>
        </spring:map>
    </spring:constructor-arg>
</spring:bean>

<jms:connector ...>
    jndiProviderProperties-ref="jmsProperties"
/>

```

If you are using the enterprise edition of Mule ESB 2.2.3 or later, and you want to override the authorization at the endpoint level, you do the following:

- Specify a custom JMS connector that uses the `com.mulesoft.mule.transport.jms.weblogic.EeWeblogicJmsConnector` class
- Create a transformer that deletes the security properties
- Call that transformer on the endpoint where you want to override the authorization, and then specify the new properties.

For example:

```

<!--
JNDI security props have to be deleted so they aren't propagated to remote destinations in the
message
-->
<message-properties-transformer name="stripJndiProps">
    <delete-message-property key="java.naming.security.principal"/>
    <delete-message-property key="java.naming.security.credentials"/>
    <delete-message-property key="java.naming.security.authentication"/>
</message-properties-transformer>

<jms:object-to-jmsmessage-transformer name="obj2jms"/>
<jms:jmsmessage-to-object-transformer name="jms2obj"/>

<jms:custom-connector name="weblogicConnector">
    class="com.mulesoft.mule.transport.jms.weblogic.EeWeblogicJmsConnector"
    jndiInitialFactory="weblogic.jndi.WLInitialContextFactory"
    connectionFactoryJndiName="weblogic.jms.ConnectionFactory"
    jndiDestinations="true"
    forceJndiDestinations="true"
    specification="1.1"
    numberOfConsumers="8"
    jndiProviderProperties-ref="jmsSecure1Properties"
    disableTemporaryReplyToDestinations="true">
</jms:custom-connector>

<model name="SecureJMSTesting">
    <service name="SecureJMS">
        <inbound>
            <jms:inbound-endpoint queue="jms.SecuredQueue1"
                transformer-refs="jms2obj stripJndiProps">
                <properties>
                    <spring:entry key="java.naming.security.principal" value="user1"/>
                    <spring:entry key="java.naming.security.credentials" value="password1"/>
                    <spring:entry key="java.naming.security.authentication" value="simple"/>
                </properties>
                <jms:transaction action="BEGIN_OR_JOIN"/>
            </jms:inbound-endpoint>
        </inbound>
    </service>
</model>

```

## SwiftMQ Integration

### SwiftMQ Integration

This page describes how to use SwiftMQ with Mule.

#### **Configuring a Mule JMS Connector**

The best approach for integrating SwiftMQ is via JNDI. You will specify the following attributes:

Attribute	Description	Recommended Value
jndiInitialFactory	InitialContext factory	com.swiftmq.jndi.InitialContextFactoryImpl
jndiProviderUrl	JNDI Provider URL	smqp://localhost:4001/timeout=10000
jndiDestinations	JNDI lookup of queues/topics	true
forceJndiDestinations	Forces a JNDI exception if a destination was not found in JNDI	true
specification	Version of the JMS specification	1.1
connectionFactoryJndiName	Name of the JMS connection factory to use	ConnectionFactory

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.0/mule-jms.xsd">

    <jms:connector name="jmsConnector"
                   connectionFactoryJndiName="ConnectionFactory"
                   jndiDestinations="true"
                   forceJndiDestinations="true"
                   jndiInitialFactory="com.swiftmq.jndi.InitialContextFactoryImpl"
                   jndiProviderUrl="smqp://localhost:4001/timeout=10000"
                   specification="1.1"/>
    ...

```

After you have configured the connector, copy `swiftnmq.jar` into the Mule `lib/user` directory and start the SwiftMQ Router. You can now use SwiftMQ from Mule.

#### **Configuring the Loan Broker ESB Example with SwiftMQ**

The following example shows you how to modify the [Loan Broker example](#) to use SwiftMQ. The only change necessary is to modify the JMS connector in both example configuration files. With a SwiftMQ Router running on the local host, the connector would look like this:

```
<jms:connector name="jmsConnector"
               connectionFactoryJndiName="ConnectionFactory"
               jndiDestinations="true"
               forceJndiDestinations="true"
               jndiInitialFactory="com.swiftmq.jndi.InitialContextFactoryImpl"
               jndiProviderUrl="smqp://localhost:4001/timeout=10000"
               specification="1.1"/>
```

The Loan Broker ESB example uses the following JMS queues (Mule syntax):

```
jms://esb.loan.quotes  
jms://esb.credit.agency  
jms://esb.lender.service  
jms://esb.banks
```

SwiftMQ does not allow dots '.' in queue names. Instead, use underscores '\_' in SwiftMQ's `routerconfig.xml`:

```
<swiftlet name="sys$queuemanager">  
  <queue-controllers>  
    <queue-controller name="01" persistence-mode="non_persistent" predicate="tmp$%" />  
    <queue-controller name="02" predicate="sys$%" />  
    <queue-controller name="03" predicate="swiftnmq%" />  
    <queue-controller name="04" predicate="rt$%" />  
    <queue-controller name="05" predicate="unroutable%" />  
    <queue-controller name="06" predicate="%%%" />  
    <queue-controller name="07" predicate="%" />  
  </queue-controllers>  
  <queues>  
    <queue name="esb_banks" />  
    <queue name="esb_credit_agency" />  
    <queue name="esb_lender_service" />  
    <queue name="esb_loan_quotes" />  
  </queues>  
</swiftlet>
```

To match with the Loan Broker ESB example's JMS queue names, define JNDI aliases in SwiftMQ's `routerconfig.xml`:

```
<swiftlet name="sys$jndi">  
  <aliases>  
    <alias name="esb.banks" map-to="esb_banks@router1" />  
    <alias name="esb.credit.agency" map-to="esb_credit_agency@router1" />  
    <alias name="esb.lender.service" map-to="esb_lender_service@router1" />  
    <alias name="esb.loan.quotes" map-to="esb_loan_quotes@router1" />  
  </aliases>  
  <jndi-replications/>  
  <remote-queues/>  
</swiftlet>
```

You now rebuild the Loan Broker ESB example with Ant or Maven so that the configuration changes can take effect, then start the SwiftMQ Router and the Loan Broker ESB example.

Note that the @ sign can be escaped with %40 in the Mule URI, so for an alternate configuration you can use the following:

```

<endpoint name="LoanBrokerRequestsREST" address="jetty:rest://localhost:8080/loanbroker"/>
<vm:endpoint name="LoanBrokerRequests" path="loan.broker.requests"/>
<jms:endpoint name="LoanQuotes" address="jms://esb_loan_quotes%40router1"/>
<jms:endpoint name="CreditAgencyGateway" address="jms://esb_credit_agency%40router1"/>
<!-- here we're telling Mule to invoke a remote Ejb directly (not host a
proxy service for the remote object as with the other example in
mule-config-with-ejb-container.xml example)
-->
<ejb:endpoint name="CreditAgency" host="localhost" port="1099" object="local/CreditAgency" method=
"getCreditProfile" />
<!-- endpoint name="CreditAgency" address=
"ejb://localhost:1099/local/CreditAgency?method=getCreditProfile" / -->
<endpoint name="LenderGateway" address="jms://esb.lender.service" />
<endpoint name="LenderService" address="vm://lender.service" />
<endpoint name="BankingGateway" address="jms://esb.banks%40router1" />
<endpoint name="Bank1" address="axis:http://localhost:10080/mule/TheBank1?method=getLoanQuote"
synchronous="true" />
<endpoint name="Bank2" address="axis:http://localhost:20080/mule/TheBank2?method=getLoanQuote"
synchronous="true" />
<endpoint name="Bank3" address="axis:http://localhost:30080/mule/TheBank3?method=getLoanQuote"
synchronous="true" />
<endpoint name="Bank4" address="axis:http://localhost:40080/mule/TheBank4?method=getLoanQuote"
synchronous="true" />
<endpoint name="Bank5" address="axis:http://localhost:50080/mule/TheBank5?method=getLoanQuote"
synchronous="true" />

```

Keep in mind that a SwiftMQ JNDI alias also decouples a queue from its physical location. You can move a queue to another router without affecting clients. So it's always best practice to avoid physical queue names.

Your Rating: 

Results:  0 rates

## ActiveMQ Integration

### ActiveMQ Integration

#### *Introduction*

Apache ActiveMQ is a popular open source messaging provider which is easy to integrate with Mule. ActiveMQ supports the JMS 1.1 and J2EE 1.4 specifications and is released under the Apache 2.0 License.

#### *Usage*

To configure ActiveMQ connector with most common settings, use `<jms:activemq-connector>` or `<jms:activemq-xa-connector>` (for XA transaction support) element in your Mule configuration, e.g:

```

<?xml version="1.0" encoding="UTF-8" ?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:jms="http://www.mulesoft.org/schema/mule/jms"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/jms
          http://www.mulesoft.org/schema/mule/jms/3.1/mule-jms.xsd">

    <jms:activemq-connector name="jmsConnector"
                           brokerURL="tcp://localhost:61616"/>
    <jms:activemq-xa-connector name="jmsXAConnector"
                             brokerURL="tcp://localhost:61616"/>
    ...

```

Mule will initialize the ActiveMQ connector with default instance of ActiveMQ connection factory and establish a TCP connection to the remote standalone broker running on local host and listening on port 61616.

Use failover:// protocol to connect to the cluster of brokers, and pass additional ActiveMQ options as URI parameters, e.g.:

```
<jms:activemq-xa-connector name="jmsFailoverConnector"  
    brokerURL="failover:(tcp://primary:61616,tcp://secondary:61616)?randomize=false"/>
```

To create an embedded instance of ActiveMQ broker, i.e. broker running on the same Java VM as Mule, use vm:// protocol, e.g.:

```
<jms:activemq-connector name="jmsConnector" brokerURL="vm://localhost"/>
```

You may also use additional connector attributes (See "Configuration Reference" for more details)

Sometimes it might be necessary to explicitly configure an instance of ActiveMQ connection factory, for example, to set redelivery policy, or other ActiveMQ-specific features that are not exposed through Mule connector parameters. To create custom ActiveMQ connection factory instance, first configure it using Spring, e.g.:

```
<bean name="connectionFactory"  
    class="org.apache.activemq.ActiveMQConnectionFactory">  
    <!-- to support XA transactions, use org.apache.activemq.ActiveMQXAConnectionFactory instead -->  
    <property name="brokerURL"  
        value="tcp://activemqserver:61616"/>  
  
    <property name="redeliveryPolicy">  
        <bean class="org.apache.activemq.RedeliveryPolicy">  
            <property name="initialRedeliveryDelay"  
                value="20000"/>  
            <property name="redeliveryDelay"  
                value="20000"/>  
            <property name="maximumRedeliveries"  
                value="10"/>  
        </bean>  
    </property>  
</bean>
```

then reference this bean in <jms:activemq-connector>, e.g.:

```
<jms:activemq-connector name="jmsConnector" connectionFactory-ref="connectionFactory"/>
```

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Activemq connector

The activemq-connector element configures an ActiveMQ version of the JMS connector.

Attributes of <activemq-connector...>

Name	Type	Required	D
name	name (no spaces)	yes	
name	name (no spaces)	yes	

dynamicNotification	boolean	no	fa
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	fa
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
createMultipleTransactedReceivers	boolean	no	
numberOfConcurrentTransactedReceivers	integer	no	
connectionFactory-ref	string	no	
redeliveryHandlerFactory-ref	string	no	
acknowledgementMode	AUTO_ACKNOWLEDGE/CLIENT_ACKNOWLEDGE/DUPS_OK_ACKNOWLEDGE	no	All
clientId	string	no	
durable	boolean	no	
noLocal	boolean	no	

persistentDelivery	boolean		no	
honorQosHeaders	boolean		no	
maxRedelivery	integer		no	
cacheJmsSessions	boolean		no	
eagerConsumer	boolean		no	
specification	1.0.2b/1.1		no	1.
username	string		no	
password	string		no	
numberOfConsumers	integer		no	
jndiInitialFactory	string		no	
jndiProviderUrl	string		no	
jndiProviderProperties-ref	string		no	
connectionFactoryJndiName	string		no	
jndiDestinations	boolean		no	
forceJndiDestinations	boolean		no	
disableTemporaryReplyToDestinations	boolean		no	

embeddedMode	boolean	no	fa
brokerURL	string	no	

Child Elements of <activemq-connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-jndi-name-resolver	0..1	A placeholder for jndi-name-resolver strategy elements.

#### Activemq xa connector

The activemq-xa-connector element configures an ActiveMQ version of the JMS connector with XA transaction support.

Attributes of <activemq-xa-connector...>

Name	Type	Required	Default
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	false
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
name	name (no spaces)	yes	

name	name (no spaces)	yes	
dynamicNotification	boolean	no	fa
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
createMultipleTransactedReceivers	boolean	no	
numberOfConcurrentTransactedReceivers	integer	no	
connectionFactory-ref	string	no	
redeliveryHandlerFactory-ref	string	no	
acknowledgementMode	AUTO_ACKNOWLEDGE/CLIENT_ACKNOWLEDGE/DUPS_OK_ACKNOWLEDGE	no	All
clientId	string	no	
durable	boolean	no	
noLocal	boolean	no	
persistentDelivery	boolean	no	
honorQosHeaders	boolean	no	
maxRedelivery	integer	no	
cacheJmsSessions	boolean	no	

eagerConsumer	boolean	no	
specification	1.0.2b/1.1	no	1.
username	string	no	
password	string	no	
numberOfConsumers	integer	no	
jndiInitialFactory	string	no	
jndiProviderUrl	string	no	
jndiProviderProperties-ref	string	no	
connectionFactoryJndiName	string	no	
jndiDestinations	boolean	no	
forceJndiDestinations	boolean	no	
disableTemporaryReplyToDestinations	boolean	no	
embeddedMode	boolean	no	fa
brokerURL	string	no	

Child Elements of <activemq-xa-connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.

abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-jndi-name-resolver	0..1	A placeholder for jndi-name-resolver strategy elements.

Your Rating: 

Results:  5 rates

## MuleMQ Integration

### Mule MQ Integration

#### Introduction

Mule MQ is an enterprise-class Java Message Service (JMS) implementation with official support from MuleSoft.

For more information, refer to the [Mule MQ Documentation](#).

#### Usage

To use Mule MQ with the [JMS Transport Reference](#) you need to:

1. Copy the Mule MQ client JAR `nJMSClient.jar` from the Mule MQ lib folder to the `<MULE_HOME>/lib/user` folder.
2. Configure the JMS connector.

Use the `<jms:mulemq-connector>` tag to configure the Mule MQ connector in your Mule configuration file. Since the `<jms:mulemq-connector>` tag extends `<jms:connector>`, all the attributes that one can set on the JMS connector can also be set on the Mule MQ connector. The following attributes are required:

```
<jms:mulemq-connector name="jmsConnector"
                      brokerURL="nsp://localhost:9000"
                      specification="1.1"
...
/>
```



If you want to use XA transactions, use the `<jms:mulemq-xa-connector>` instead of the standard `<jms:mulemq-connector>`. The `mulemq-xa-connector` has the same extra attributes as the default connector.

#### Connecting using JNDI

1. First configure JNDI managed object using the Mule MQ JNDI admin
1. Then connect using the managed connection factory:

```
<jms:connector name="muleMQJmsConnector"
               jndiInitialFactory="com.pcbsys.nirvana.nSpace.NirvanaContextFactory"
               jndiProviderUrl="nsp://172.16.10.148:9000"
               connectionFactoryJndiName="ConnectionFactory"
               specification="1.1"
...
/>
```

#### Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

#### Mulemq connector

The mulemq-connector element configures a MuleMQ version of the JMS connector.

Attributes of <mulemq-connector...>

Name	Type	Required	Default
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	false
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	false
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
createMultipleTransactedReceivers	boolean	no	
numberOfConcurrentTransactedReceivers	integer	no	
connectionFactory-ref	string	no	
redeliveryHandlerFactory-ref	string	no	
acknowledgementMode	AUTO_ACKNOWLEDGE/CLIENT_ACKNOWLEDGE/DUPS_OK_ACKNOWLEDGE	no	All

clientId	string	no	
durable	boolean	no	
noLocal	boolean	no	
persistentDelivery	boolean	no	
honorQosHeaders	boolean	no	
maxRedelivery	integer	no	
cacheJmsSessions	boolean	no	
eagerConsumer	boolean	no	
specification	1.0.2b/1.1	no	1.
username	string	no	
password	string	no	
numberOfConsumers	integer	no	
jndiInitialFactory	string	no	

jndiProviderUrl	string	no	
jndiProviderProperties-ref	string	no	
connectionFactoryJndiName	string	no	
jndiDestinations	boolean	no	
forceJndiDestinations	boolean	no	
disableTemporaryReplyToDestinations	boolean	no	
embeddedMode	boolean	no	fa
brokerURL	string	no	
bufferOutput	string	no	qu
syncWrites	boolean	no	fa
syncBatchSize	integer	no	50
syncTime	integer	no	20
globalStoreCapacity	integer	no	50

maxUnackedSize	integer	no	1000
useJMSEngine	boolean	no	true
queueWindowSize	integer	no	1000
autoAckCount	integer	no	500
enableSharedDurable	boolean	no	false
randomiseRNames	boolean	no	true
messageThreadPoolSize	integer	no	300
discOnClusterFailure	boolean	no	true
initialRetryCount	integer	no	2
muleMqMaxRedelivery	integer	no	1000
retryCommit	boolean	no	false
enableMultiplexedConnections	boolean	no	false

Child Elements of <mulemq-connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.

abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-jndi-name-resolver	0..1	A placeholder for jndi-name-resolver strategy elements.

#### **Mulemq xa connector**

The mulemq-xa-connector element configures a MuleMQ version of the JMS XA connector.

Attributes of <mulemq-xa-connector...>

Name	Type	Required	Default
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	false
validateConnections	boolean	no	true
dispatcherPoolFactory-ref	string	no	
name	name (no spaces)	yes	
name	name (no spaces)	yes	
dynamicNotification	boolean	no	false
validateConnections	boolean	no	true

dispatcherPoolFactory-ref	string	no	
createMultipleTransactedReceivers	boolean	no	
numberOfConcurrentTransactedReceivers	integer	no	
connectionFactory-ref	string	no	
redeliveryHandlerFactory-ref	string	no	
acknowledgementMode	AUTO_ACKNOWLEDGE/CLIENT_ACKNOWLEDGE/DUPS_OK_ACKNOWLEDGE	no	All
clientId	string	no	
durable	boolean	no	
noLocal	boolean	no	
persistentDelivery	boolean	no	
honorQosHeaders	boolean	no	
maxRedelivery	integer	no	

cacheJmsSessions	boolean		no	
eagerConsumer	boolean		no	
specification	1.0.2b/1.1		no	1.
username	string		no	
password	string		no	
numberOfConsumers	integer		no	
jndiInitialFactory	string		no	
jndiProviderUrl	string		no	
jndiProviderProperties-ref	string		no	
connectionFactoryJndiName	string		no	
jndiDestinations	boolean		no	
forceJndiDestinations	boolean		no	
disableTemporaryReplyToDestinations	boolean		no	
embeddedMode	boolean		no	fa
brokerURL	string		no	

bufferOutput	string	no	qu
syncWrites	boolean	no	fa
syncBatchSize	integer	no	50
syncTime	integer	no	20
globalStoreCapacity	integer	no	50
maxUnackedSize	integer	no	10
useJMSEngine	boolean	no	true
queueWindowSize	integer	no	10
autoAckCount	integer	no	50
enableSharedDurable	boolean	no	false
randomiseRNames	boolean	no	true
messageThreadPoolSize	integer	no	30

discOnClusterFailure	boolean	no	true
initialRetryCount	integer	no	2
muleMqMaxRedelivery	integer	no	10
retryCommit	boolean	no	false
enableMultiplexedConnections	boolean	no	false

Child Elements of <mulemq-xa-connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
abstract-jndi-name-resolver	0..1	A placeholder for jndi-name-resolver strategy elements.

Your Rating: 

Results:  0 rates

## Mule MSMQ Transport Reference



### Mule MSMQ Transport Reference



The Mule MSMQ Transport is currently in beta.

[ [Setting Up the Mule MSMQ Transport](#) ] [ [Configuring the MSMQ Connector](#) ] [ [Configuring an MSMQ Endpoint](#) ] [ [Configuring Message Properties Programmatically](#) ] [ [Configuring Retry Policies](#) ] [ [Configuring Logging](#) ]

The Mule Enterprise transport for Microsoft® Message Queuing (MSMQ) allows you to dispatch messages through an MSMQ endpoint. MSMQ provides guaranteed message delivery, efficient routing, security, and priority-based messaging. It can be used to implement solutions for both asynchronous and synchronous messaging scenarios.

This page describes how to configure and use this transport and its connector. It assumes that you already understand how to configure and use MSMQ. For complete information on MSMQ, see the [Microsoft Message Queuing](#) in the Microsoft documentation.

#### Notes:

- The Mule MSMQ transport is currently in beta and is available only with Mule Enterprise Edition version 2.2.4 or later.
- The transport has been tested with MSMQ version 3.
- MSMQ is designed to be run on Windows only, and MSMQ uses trusted certificates from Internet Explorer to authenticate messages that are not in a domain. For authentication to work well, the client must be running Windows. If you are using a Linux test server, you must exclude the authentication test case. For more information, see [Securing Your MSMQ Enterprise](#) in the Microsoft Message Queuing documentation.

## Setting Up the Mule MSMQ Transport

To use the Mule MSMQ transport, do the following:

- Contact MuleSoft technical support to receive a copy of the Mule MSMQ distribution.
- Extract the Mule MSMQ ZIP file to your existing Mule 2.2.4 home directory.
- Add the MSMQ namespace to the Mule XML configuration file(s) where you will configure the MSMQ connector and endpoints. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:msmq="http://www.mulesoft.org/schema/mule/ee/msmq"
    xsi:schemaLocation="
        http://www.mulesoft.org/schema/mule/ee/msmq
        http://www.mulesoft.org/schema/mule/ee/msmq/3.0/mule-msmq-ee.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">

</mule>
```

- If you are operating from a Linux/Unix or Mac platform, make sure the `/etc/hosts` contain the NetBios name of the MSMQ server as well as the appropriate IP address.

You are now ready to configure the MSMQ connector and endpoints in your Mule XML configuration file.

## Configuring the MSMQ Connector

When specifying an MSMQ endpoint, the default connector is used unless you specify a connector that you have explicitly configured. The connector properties reflect the properties you can set on the MSMQ queues themselves.



Configuring the connector makes changes to the underlying MSMQ queue residing on the server. Therefore, other applications that will be connecting to the same queue can be affected by the changes.

Following are the properties you can set on an MSMQ connector:

Property	Description	Type	Valid Values	Required?
Username	Username on the server where MSMQ is running	String		YES if not set on endpoint

Passwd	Password on the server where MSMQ is running	String		YES if not set on endpoint
connection-type	Connection type being used: ex DIRECT=OS	String		NO
domain	The domain to log into. For workgroup, use "localhost"	String		YES if not set on endpoint
log-dcom-to-file	The Jinterop library will log to file when this property is set to true	Boolean		NO (defaults to false)
log-dcom-level	The Jinterop logging level	String	SEVERE, ALL, CONFIG, OFF, INFO, WARNING, FILE, FINER, FINEST	NO (defaults to SEVERE)
authenticate	The authenticate property specifies whether the queue only accepts authenticated messages	String	MQ_AUTHENTICATE_NONE, MQ_AUTHENTICATE	NO
base-priority	This property specifies a single base priority for all messages sent to a public queue	Integer		NO
deny-shared-receive	Specifies who can access the queue	String	MQ_DENY_NONE, MQ_DENY_RECEIVE_SHARE	No (defaults to MQ_DENY_NONE)
encryption-required	The private level property specifies whether the queue accepts private (encrypted) messages, non-private messages, or both.	String	MQ_PRIV_LEVEL_NONE, MQ_PRIV_LEVEL_OPTIONAL, MQ_PRIV_LEVEL_BODY	No
label	The label property specifies a description of the queue.	String		No
maximum-journal-size	Maximum size (in kilobytes) of the queue journal	Integer		No
maximum-queue-size	This property specifies the maximum storage size (in kilobytes) of the queue	Integer		No
multicast-address	Specifies the multicast address associated with the queue	String		No
use-journal-queue	Enables or disables journaling	Boolean		No
default-properties-to-send	A map of key value pairs where the key denotes the property name and the value denotes the value of the property. All properties set in this map will be set to all messages being routed through endpoints using the connector.	Map of key-value pairs		No

For example:

```
<msmq:connector name="msmqConnector" connection-type="DIRECT:OS" username="${uname}"  
passwd="${passwd}" domain="${domain}">  
    <msmq:default-properties-to-send  
        key="MsmqPriority" value="3"  
        key="MsmqEncryption" value="MQMSG_PRIV_LEVEL_BODY_ENHANCED"  
    />  
</msmq:connector>
```

You could then specify this connector when configuring your endpoints.

## Configuring an MSMQ Endpoint

The properties configurable on the endpoint reflect the MSMQ message properties. All messages routed via the endpoint will be modified such that the properties configured on the endpoint in the Mule XML configuration file will be set on the MSMQ message before dispatching it to the queue. Properties set on the endpoint will override any properties also set using the `<default-properties-to-send>` element on the connector.

Following are the properties you can configure on an MSMQ endpoint:

Property	Description	Type	Valid Values	Required
----------	-------------	------	--------------	----------

Username	Username on the server where MSMQ is running	String		YES if not set on the connector
passwd	Password on the server where MSMQ is running	String		YES if not set on the connector
connection-type	Connection type being used: ex DIRECT=OS	String		NO
domain	The domain to log into. For workgroup, use "localhost"	String		YES if not set on the connector
hash-algorithm	The hash-algorithm property specifies the hashing algorithm used by Message Queuing when authenticating messages.	Boolean	MQMSG_CALG_MD2, MQMSG_CALG_MD4, MQMSG_CALG_MD5, MQMSG_CALG_SHA1	NO
encryption-algorithm	Specifies the encryption algorithm used to encrypt the message body of a private message	String	MQMSG_CALG_RC2, MQMSG_CALG_RC4	No
priority	The priority property specifies the message's priority. A low number indicates a low priority message	Integer		No
administration-queue	Specifies the queue used for Message Queuing-generated acknowledgment messages. This object is passed to the queue manager on the target computer	String		No
label	The Label property of the MSMQMessage object specifies a description of the message	String		No
app-specific	The AppSpecific property of the MSMQMessage object specifies application-generated information, such as single integer values or application-defined message classes	Integer		No
acknowledge-types	Specifies the type of acknowledgment messages that Message Queuing will post (in the administration queue) when acknowledgments are requested. This property is a space separated list of acknowledgments	List of white space separated Strings	MSMQ_ACKNOWLEDGMENT_NONE, MSMQ_ACKNOWLEDGMENT_POS_ARRIVAL, MSMQ_ACKNOWLEDGMENT_POS_RECEIVE, MSMQ_ACKNOWLEDGMENT_NEG_ARRIVAL, MSMQ_ACKNOWLEDGMENT_NEG_RECEIVE, MSMQ_ACKNOWLEDGMENT_NACK_REACH_QUEUE, MSMQ_ACKNOWLEDGMENT_FULL_REACH_QUEUE, MSMQ_ACKNOWLEDGMENT_NACK_RECEIVE, MSMQ_ACKNOWLEDGMENT_FULL_RECEIVE	NO
authentication-provider-name	The name of the cryptographic provider used to generate the digital signature attached to the message	String	Microsoft Base Cryptographic Provider, v1.0, Microsoft Enhanced Cryptographic Provider, v1.0	No

authentication-provider-type	The type of cryptographic provider used to generate the digital signature attached to the message. The only supported value of this property is PROV_RSA_FULL, which is the default value.	Integer		No
connector-type	The ConnectorTypeGuid property of the MSMQMessage object indicates that some message properties that are typically set by Message Queuing were set by the sending application. Typically, this property is set by a connector application and applications sending application-encrypted messages.	String		No
time-to-be-received	A time limit (in seconds) for the message to be retrieved from the target queue. This includes the time spent getting to the destination queue plus the time spent waiting in the queue before it is retrieved by an application.	Integer		NO
time-to-reach-queue	Specifies a time limit (in seconds) for the message to reach the queue	Integer		No
auth-level	Whether the message should be authenticated and what type of digital signature is used	String	MQMSG_AUTH_LEVEL_NONE, MQMSG_AUTH_LEVEL_ALWAYS, MQMSG_AUTH_LEVEL_MSMQ10, MQMSG_AUTH_LEVEL_SIG10, MQMSG_AUTH_LEVEL_MSMQ20, MQMSG_AUTH_LEVEL_SIG20, MQMSG_AUTH_LEVEL_SIG30	No
encryption	The private level property of the MSMQMessage object specifies privacy level of the message	String	MQMSG_PRIV_LEVEL_NONE, MQMSG_PRIV_LEVEL_BODY, MQMSG_PRIV_LEVEL_BODY_BASE, MQMSG_PRIV_LEVEL_BODY_ENHANCED	No
use-journal-queue	The Journal property of the MSMQMessage object specifies whether Message Queuing stores copies of the message as it is routed to the destination queue	Boolean		No
use-dead-letter-queue	Negative source journaling is requested. The message is stored in the applicable dead-letter queue on failure, but no copy is stored in the computer journal on success	Boolean		No
use-tracing	The Trace property of the MSMQMessage object specifies how Message Queuing traces the route of the message	Boolean		No

destination-symmetric-key	Specifies the path of the file where the symmetric key used to encrypt the message can be loaded from.	String		No
digital-signature	The path to the application-generated signature that is attached to the message. In most cases, Signature is set by the Message Queuing runtime when the sending application requests authentication	String		No
sender-certificate	Path to file that represents the user certificate. The user certificate is used to authenticate messages	String		No
sender-id	Path to file that represents the identifier of the sending user. Typically, Message Queuing sets this property when the message is sent			No
correlation-id	Path to file contains a 20-byte correlation identifier used to identify the message			No

## Configuring Message Properties Programmatically

As described above, you can override the MSMQ message properties in the configuration file using `<default-properties-to-send>` on the connector or by overriding them at the endpoint. You can also override properties programmatically by setting them on the Mule message. The properties are the same as the endpoint properties listed above with the following changes:

- Prefix each property with MSMQ and capitalize the first letter.
- Property names that contain the symbol "-" in the above table should be changed by deleting the "-" and capitalizing the next letter.
- All MSMQ property names that can be set on the Mule message are defined in the `org.mule.transport.msmq.MsmqConstants` interface.

For example, you can override the `label` property on the endpoint by setting `MsmqConstants.MSMQLabel` on the Mule message.

To set properties on the Mule message, you add the property names and values to a hash map, and then create a new message with those properties as shown in the following example:

```
Map props = new HashMap();
props.put(MsmqConstants.MSMQ_AUTH_LEVEL,
MqMsgAuthLevel.convertAuthLevelToString(MqMsgAuthLevel.MQMSG_AUTH_LEVEL_NONE));
client.dispatch("vm://fromTestCase", new DefaultMuleMessage(MESSAGE, props));
```

Constants such as `MQMSG_CALG_SHA1` are defined in their respective class under the package `org.mule.transport.msmq.constants`. These classes also provide helper methods to convert the appropriate integer value to their respective String representation. For example, the following line converts the value denoting no encryption (0) to its appropriate string representation (`MQMSG_PRIV_LEVEL_NONE`):

```
MqMsgPrivLevel.convertPrivLevelToString(MqMsgPrivLevel.MQMSG_PRIV_LEVEL_NONE)
```

Properties that require a path to a file when configured in the XML file (such as `sender-certificate`, `correlation-id`, and `digital-signature`) are treated differently when set directly on the Mule message. If a String type is found as the value of one of these properties, the behavior is exactly the same (assume that the string refers to a file containing the information). The MSMQ transport also allows these properties to be assigned the direct byte array value, in which case the transport will not load any data from any file, but if the type is `byte[]`, it will simply forward the value directly.

## Configuring Retry Policies

The MSMQ transport supports **retry policies** as long as there is only one connector per MSMQ server. Internally, a reconnection causes the dispatcher and receiver classes to be disconnected, so connections to a server other than the one being re-connected will cause data loss. A retry

is triggered by DCOM exceptions such as "MSMQ server was restarted," "Connection with server lost," or "Connection with server timed out."

Message bursts of 400+ messages are liable to trigger a "Java.net.Binding" exception (thrown from the JIInterop library) due to an issue in the library itself. Current network performance constraints mean that this issue only occurs when the Mule connector is deployed on the same server as MSMQ. A workaround for this situation is to throttle speed when operating locally so that the connector will not fail. However, rather than fail, the connector is designed to sleep for 80 seconds and then continue working. You can also configure a retry policy on it so that any failures are automatically caught and connections are re-attempted.

## Configuring Logging

All JIInterop logs are treated independently from the Mule logging mechanism, since the library has its own logging mechanism. However, you can configure the logging within the MSMQ connector by modifying the `log-dcom-to-file` property, which directs the logging to a file as well as the console. This file is under `/tmp/j-interop0.log` on Linux/Unix based systems and under the `temp` directory on Windows systems. This property can only be set once per Mule instance, as multiple connectors with different settings for this property will override each other.

Your Rating: 

Results:  0 rates

## Mule WMQ Transport Reference



### Mule WMQ Transport Reference

[ Setting Up the Mule WMQ Transport ] [ Using the Mule WMQ Transport ] [ Defining WMQ XA Connector ] [ Inbound Message Handling ] [ Outbound Message Handling ] [ Retrieving the Connection Factory from JNDI ] [ Transformers ] [ Transactions ] [ Configuring Retry Policies ] [ Known Limitations ] [ Configuration Reference ] [ Connector ] [ Xa Connector ] [ Inbound Endpoint ] [ Outbound Endpoint ] [ Endpoint ] [ Message To Object Transformer ] [ Object To Message Transformer ] [ Transaction ]

The Mule Enterprise transport for IBM® WebSphere® MQ (Mule WMQ transport) provides the following functionality:

- Bi-directional request-response messaging between Websphere MQ and Mule JMS using native MQ point-to-point.
- Synchronous and asynchronous request-response communications using `JMSReplyTo`.
- Support for local, multi-resource, and XA transactions
- Support for native MQ message types
- WebSphere MQ-specific configuration syntax that removes the complexity of MQ configuration
- Supports async request-response messaging pattern transparently in both JMS and native MQ message types
- Support for connection retry policies (self-healing) with or without transactions
- Support for IBM-specific headers
- JNDI destination support including JNDI connections
- Per endpoint control over target clients (native or JMS compliant)
- Certified on WMQ v6 and WMQ v7 (use WebSphere MQ 6.0.2.8 client JARs in both cases)

This page describes how to configure and use this transport and its connector. It assumes that you already understand how to configure and use WebSphere MQ. For complete information on WebSphere MQ, refer to the [WebSphere MQ Information Center](#).

The Mule WMQ transport is available only with Mule Enterprise Edition version 1.6 or later and, as of 2.2, contains many critical performance and reliability improvements as well as native support for WebSphere MQ-specific features. Note that the community JMS transport does not provide full support for WebSphere MQ and is not recommended for use with production WebSphere MQ installations.

### Setting Up the Mule WMQ Transport

In Mule 2.x, before you can use the Mule WMQ transport, you must copy the following JARs from the `java/lib` directory under your WebSphere MQ home directory into the `lib/user` directory under your Mule home directory:

- `com.ibm.mq.jar`
- `com.ibm.mqgetclient.jar` (if using `remote XA` transactions)
- `com.ibm.mq.jmqi.jar` (WebSphere MQ version 7 only)
- `com.ibm.mqjms.jar`
- `dhbcore.jar`

Additionally, if you are using local bindings (`transportType="BINDINGS_MQ"`), you must set the following environment variable:

```
export LD_LIBRARY_PATH=/opt/mqm/java/lib
```

Set this variable to the location of the Java library directory that shipped with the WebSphere MQ server installation. For more information about the bindings types, see the WebSphere MQ documentation.

## Using the Mule WMQ Transport

By default, messages sent from Mule to WebSphere MQ are sent in native format, and JMS (RFH2) headers are suppressed. This configuration is applied transparently to the configuration below by the connector appending a parameter to the WebSphere MQ destination URI (`?targetClient=1`). To force JMS behavior on the receiving WebSphere MQ (that is, to use non-native format), use the following attribute setting in the WMQ connector declaration:

```
<wmq:connector name="..."  
...  
    targetClient="JMS_COMPLIANT"  
...  
/>
```

**Note:** The `targetClient` attribute must be set to `JMS_COMPLIANT` when the message payload is an object.

The following configuration example configures a service named `test` that performs request-reply processing. It picks up messages from the native WebSphere MQ endpoint (`wmq://REQUEST.QUEUE`) and forwards them to another native WebSphere MQ endpoint, `wmq://RESPONSE.QUEUE`. The Responder service waits while Mule processes the messages asynchronously and appends the string `RESPONSE OK` when it receives messages on the response queue. All JMS semantics apply, and settings such as `replyTo` and `QoS` properties are read from the message properties, or defaults are used (according to the JMS specification).

Note the following additional points about the configuration:

- The `wmq` URI scheme for endpoints indicates that the WebSphere MQ transport should be used.
- The `queueManager` property in the WMQ connector declaration matches the WebSphere MQ QueueManager set up previously.
- The local queues `REQUEST.QUEUE` and `RESPONSE.QUEUE` were set up previously using the `rwmqsc` utility. If you were running on a remote queue, you could specify the channel with the `channel` attribute.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:wmq="http://www.mulesoft.org/schema/mule/ee/wmq"
      xmlns:test="http://www.mulesoft.org/schema/mule/test"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/test
          http://www.mulesoft.org/schema/mule/test/3.1/mule-test.xsd
          http://www.mulesoft.org/schema/mule/ee/wmq
          http://www.mulesoft.org/schema/mule/ee/wmq/3.1/mule-wmq-ee.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <wmq:connector name="wmqConnector"
        hostName="winter" port="1414"
        queueManager="HELLO.QMGR"
        transportType="CLIENT_MQ_TCPIP"
        specification="1.1"
        disableTemporaryReplyToDestinations="true"
        username=""
        password=""
        numberOfConsumers="1">
    </wmq:connector>

    <model>
        <service name="test">
            <inbound>
                <wmq:inbound-endpoint queue="REQUEST.QUEUE" exchange-pattern="request-response"/>
            </inbound>

            <test:component/>

            <outbound>
                <pass-through-router>
                    <wmq:outbound-endpoint queue="MIDDLE.QUEUE"/>
                    <reply-to address="wmq://RESPONSE.QUEUE"/>
                </pass-through-router>
            </outbound>

            <async-reply failOnTimeout="true" timeout="5000">
                <wmq:inbound-endpoint queue="RESPONSE.QUEUE"/>
                <single-async-reply-router>
                    <wmq:message-info-mapping/>
                </single-async-reply-router>
            </async-reply>
        </service>

        <service name="Responder">
            <inbound>
                <wmq:inbound-endpoint queue="MIDDLE.QUEUE"/>
            </inbound>

            <test:component appendString=" RESPONSE OK"/>

        </service>
    </model>
</mule>

```

If you were running on a remote queue, you could use the WebSphere MQ utility `amqsget` to verify that the message was received on the remote queue.

## Defining WMQ XA Connector

Defining CF via Spring is optional, you can simply define WMQ XA-enabled connector as follows:

```
<wmq:xa-connector ...>
```

It will instantiate the XA CF under the hood, no reference to explicitly defined CF is required.

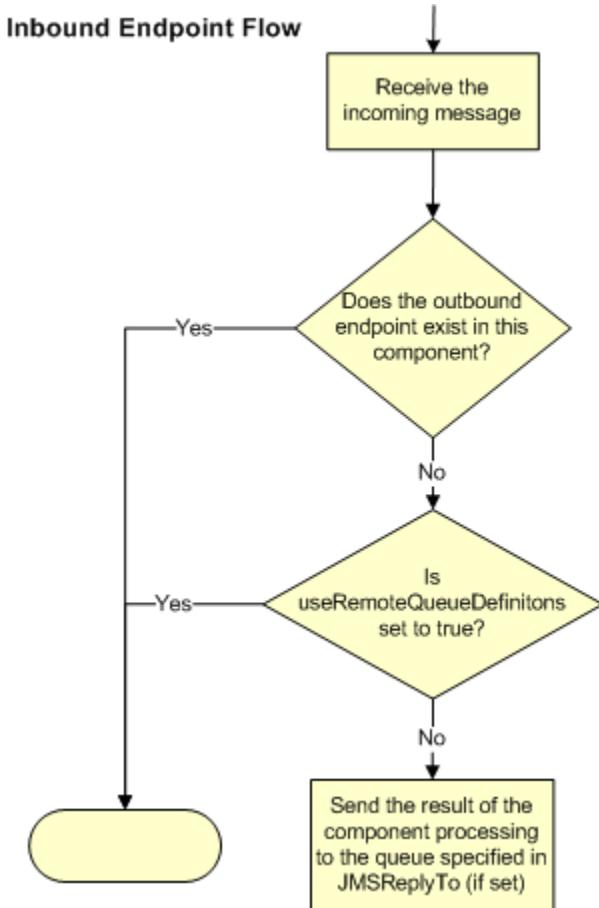
However, in some situations you need to define CF explicitly and then reference it in the connector definition. If that is the case, then the CF class has to be MQXAConnectionFactory, if XA transactions are used. Then WMQ connector has to reference this bean, e.g.:

```
<spring:bean id="mqXAFactory" class="com.ibm.mq.jms.MQXAConnectionFactory">
...
</spring:bean>

<wmq:xa-connector ... connectionFactory-ref="mqXAFactory">
```

## Inbound Message Handling

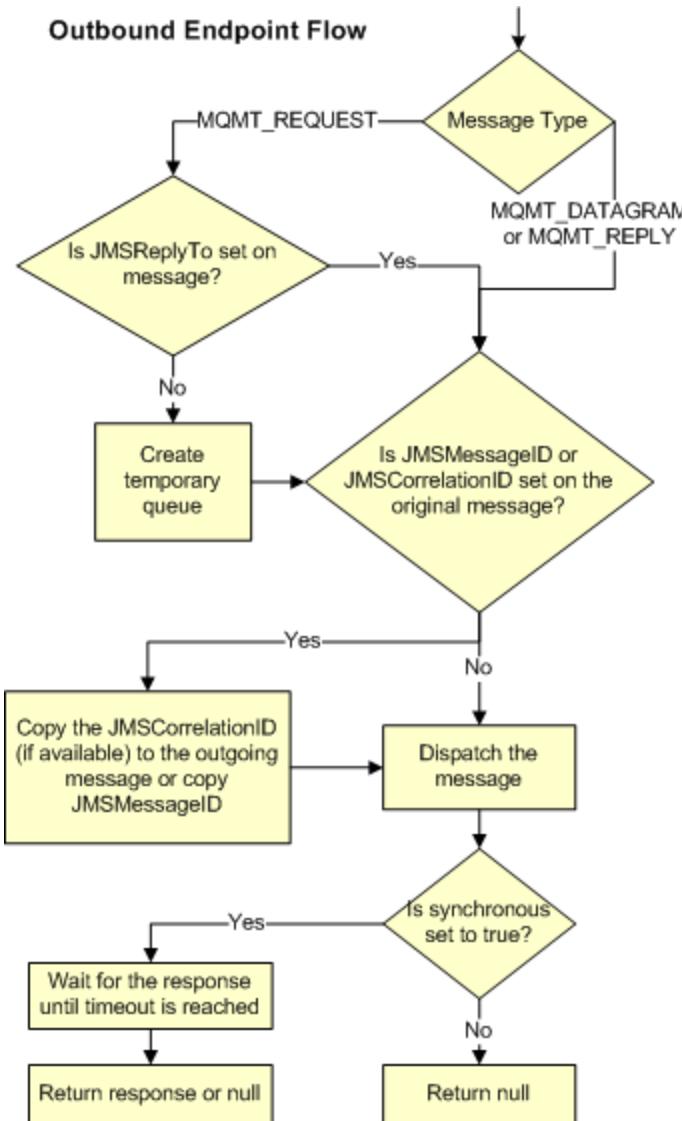
The inbound messages are received by the connector and delivered to the component. If the `useRemoteQueueDefinitions` connector attribute is not set to true and the inbound message type is `MQMT_REQUEST`, the message returned by the component will be sent to the queue specified in the `JMSReplyTo` property of the original message. However, if the outbound WebSphere MQ endpoint exists in the component, it will override the `replyto` handler functionality. By default, `useRemoteQueueDefinitions` is set to false.



## Outbound Message Handling

The outbound endpoint behavior depends on the WebSphere MQ message type. If the message type is `MQMT_REPLY` or `MQMT_DATAGRAM`, other properties will be copied over from the original message and the message will be dispatched to the queue.

If the message type is `MQMT_REQUEST`, the connector will check for the existence of the `JMSReplyTo` setting on the message. If it is not set, the temporary queue will be created. If the endpoint is synchronous, the connector will wait for a response. The timeout can be set using the `responseTimeout` setting. If a response is received by the connector, it will be returned by the component.



## Retrieving the Connection Factory from JNDI

To support the case where a JNDI registry has been configured to store the connection factory, the connector declaration should include the following parameters. This is the same as the regular JMS transport documented [here](#).

```

<wmq:connector ...
  jndiInitialFactory="com.sun.jndi.ldap.LdapCtxFactory"
  jndiProviderUrl="ldap://localhost:10389/"
  connectionFactoryJndiName="cn=ConnectionFactory,dc=example,dc=com"
  
```

## Transformers

The WMQ transport provides a transformer for converting a `com.ibm.jms.JMSMessage` or sub-type into an object by extracting the message payload. It also provides a transformer to convert the object back to a message. You use the `<message-to-object-transformer>` and `<object-to-message-transformer>` elements to configure these transformers. Note that object payloads work only when `targetClient` is set to `JMS_COMPLIANT`.

## Transactions

You can configure single-resource (local), multi-resource, and XA transactions on WMQ transport endpoints using the standard transaction configuration elements. For example, you might configure an XA transaction on an outbound endpoint as follows:

```

<jbossts:transaction-manager/>

<wmq:xa-connector name="wmqConnector" hostName="winter" ...>
...
<outbound>
  <pass-through-router>
    <wmq:outbound-endpoint queue="out">
      <xa-transaction action="ALWAYS_BEGIN" />
    </wmq:outbound-endpoint>
  </pass-through-router>
</outbound>
...

```

Note that if you are using XA transactions, and you are connecting to a queue that requires the queue manager to connect to a remote resource, you must use the extended transactional client from WebSphere MQ (`mqlclient.jar`). For more information, see [What is an extended transactional client?](#) in the WebSphere MQ 7 help.

For more information on using transactions, see [Transaction Management](#).

## Configuring Retry Policies

The WMQ transport supports [retry policies](#). As of Mule 2.2.3, you can configure the timeout value on the connector as follows:

```

<wmq:connector name="wmqConnector" ...>
  <spring:property name="connectionLostTimeout" value="3000"/>
  <ee:retry-forever-policy frequency="3000" />
</wmq:connector>

```

The example that ships with the Mule WMQ transport allows you to test retry policies. For complete information, see the `readme` file in the WMQ distribution.

## Known Limitations

Following are the features that have not been fully tested with the Mule WMQ transport or are not supported:

- Remote queues (tested only in previous releases)
- Exit handler support (not tested)
- Topics (not tested)
- MQMT\_REPORT message type support (not supported)
- Native WMQ connection pool support (not supported)
- SSL connection support (not supported)
- Data compression over channels for performance throughput gain (not supported)

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Connector

The default WebSphere MQ connector.

#### Attributes of `<connector...>`

Name	Type	Required	Default	Description
queueManager	string	no		The name of the QueueManager to use.
hostName	string	no		The host name of the QueueManager to use.
port	port number	no		The port of the QueueManager to use.
temporaryModel	string	no		The temporary destination model to use when creating temporary destinations from this connector.

ccsId	integer	no		The WebSphere MQ CCS ID.
transportType		no		Whether to use a local binding or client/server TCP binding. Possible values are: BINDINGS_MQ, CLIENT_MQ_TCPIP, DIRECT_HTTP, DIRECT_TCPIP, and MQJD.
channel	string	no		The name of the channel used to communicate with the QueueManager.
propagateMQEvents	boolean	no		
useRemoteQueueDefinitions	boolean	no		When using remote queue definitions, WMQ uses the JMSReplyTo property to channel responses. When set to true this property will cause Mule to ignore ReplyTo queue destinations and not interfere with WMQ's remote queue mechanism. By default this is set to false. This also means that by using WMQ's remote queue definitions it is not possible to use some of Mule's request/response patterns when this property is true.
receiveExitHandler	class name	no		The fully qualified class name of the receive exit handler implementation.
receiveExitHandlerInit	class name	no		An initialization parameter for the receive exit handler.
sendExitHandler	class name	no		The fully qualified class name of the send exit handler implementation.
sendExitHandlerInit	class name	no		An initialization parameter for the send exit handler.
securityExitHandler	class name	no		The fully qualified class name of the security exit handler implementation.
securityExitHandlerInit	class name	no		An initialization parameter for the security exit handler.
targetClient		no		Specifies whether this is in JMS or non-JMS format. Possible values are: JMS_COMPLIANT or NONJMS_MQ (default).

cache: Unexpected program error: java.lang.NullPointerException

## Xa Connector

The WebSphere MQ connector for XA transactions.

### Attributes of <xa-connector...>

Name	Type	Required	Default	Description
queueManager	string	no		The name of the QueueManager to use.
hostName	string	no		The host name of the QueueManager to use.
port	port number	no		The port of the QueueManager to use.
temporaryModel	string	no		The temporary destination model to use when creating temporary destinations from this connector.
ccsId	integer	no		The WebSphere MQ CCS ID.
transportType		no		Whether to use a local binding or client/server TCP binding. Possible values are: BINDINGS_MQ, CLIENT_MQ_TCPIP, DIRECT_HTTP, DIRECT_TCPIP, and MQJD.
channel	string	no		The name of the channel used to communicate with the QueueManager.
propagateMQEvents	boolean	no		
useRemoteQueueDefinitions	boolean	no		When using remote queue definitions, WMQ uses the JMSReplyTo property to channel responses. When set to true this property will cause Mule to ignore ReplyTo queue destinations and not interfere with WMQ's remote queue mechanism. By default this is set to false. This also means that by using WMQ's remote queue definitions it is not possible to use some of Mule's request/response patterns when this property is true.

receiveExitHandler	class name	no		The fully qualified class name of the receive exit handler implementation.
receiveExitHandlerInit	class name	no		An initialization parameter for the receive exit handler.
sendExitHandler	class name	no		The fully qualified class name of the send exit handler implementation.
sendExitHandlerInit	class name	no		An initialization parameter for the send exit handler.
securityExitHandler	class name	no		The fully qualified class name of the security exit handler implementation.
securityExitHandlerInit	class name	no		An initialization parameter for the security exit handler.
targetClient		no		Specifies whether this is in JMS or non-JMS format. Possible values are: JMS_COMPLIANT or NONJMS_MQ (default).

cache: Unexpected program error: java.lang.NullPointerException

## Inbound Endpoint

An endpoint on which WMQ messages are received.

### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
queue	string	no		The queue name.

cache: Unexpected program error: java.lang.NullPointerException

## Outbound Endpoint

An endpoint to which WMQ messages are sent.

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
queue	string	no		The queue name.
disableTemporaryReplyToDestinations	boolean	no		If this is set to false (the default), when Mule performs request/response calls a temporary destination will automatically be set up to receive a response from the remote WMQ call.
correlationId	string	no		A client can use the correlation ID header field to link one message to another. A typical use case is to link a response message with its request message. The CorrelationID must be a 24-byte String. WebSphere will pad shorter values with zeroes so that the fixed length is always 24 bytes. Because each message sent by a WMQ provider is assigned a message ID value, it is convenient to link messages via the message ID. All message ID values must start with the 'ID:' prefix.

messageType		no		Indicates the message type. Each of the message types have specific behavior associated with them. The following message types are defined: MQMT_REQUEST: The message requires a reply. Specify the name of the reply queue using the <ReplyTo> element of outbound routers. Mule handles the underlying configuration. MQMT_DATAGRAM: The message does not require a reply. MQMT_REPLY: The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the <ReplyTo> configured on the outbound router. Mule automatically configures the request to control how to set the MessageId and CorrelationId of the reply. MQMT_REPORT: The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Sends the message to the queue indicated by the <ReplyTo> configuration of the message descriptor of the original message.
characterSet	integer	no		If set, this property overrides the coded character set property of the destination queue or topic.
persistentDelivery	boolean	no		If set to true, the JMS provider logs the message to stable storage as it is sent so that it can be recovered if delivery is unsuccessful. A client marks a message as persistent if the application will have problems if the message is lost in transit. A client marks a message as non-persistent if an occasional lost message is tolerable. Clients use delivery mode to tell a JMS provider how to balance message transport reliability/throughput. Delivery mode only covers the transport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a PERSISTENT delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site-specific message retention policy may be dropped. A message is guaranteed to be delivered once and only once by a JMS provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.
timeToLive	long	no		Define the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system. Time to live is set to zero (forever) by default.
priority	substitutablePriorityNumber	no		Sets the message priority. JMS defines a ten-level priority value with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. JMS does not require that a provider strictly implement priority ordering of messages. However, it should do its best to deliver expedited messages ahead of normal messages.
targetClient		no		Specifies whether this is in JMS or non-JMS format. Possible values are: JMS_COMPLIANT or NONJMS_MQ (default).

cache: Unexpected program error: java.lang.NullPointerException

## Endpoint

A global WMQ endpoint definition. Note that global endpoints are like endpoint factories from which new endpoints can be created. As such this endpoint has a union of inbound and outbound endpoint properties. Depending on how this endpoint is used the unneeded properties will be ignored.

### Attributes of <endpoint...>

Name	Type	Required	Default	Description
queue	string	no		The queue name.
disableTemporaryReplyToDestinations	boolean	no		If this is set to false (the default), when Mule performs request/response calls a temporary destination will automatically be set up to receive a response from the remote WMQ call.
correlationId	string	no		A client can use the correlation ID header field to link one message to another. A typical use case is to link a response message with its request message. The CorrelationID must be a 24-byte String. WebSphere will pad shorter values with zeroes so that the fixed length is always 24 bytes. Because each message sent by a WMQ provider is assigned a message ID value, it is convenient to link messages via the message ID. All message ID values must start with the 'ID:' prefix.
messageType		no		Indicates the message type. Each of the message types have specific behavior associated with them. The following message types are defined: MQMT_REQUEST: The message requires a reply. Specify the name of the reply queue using the <ReplyTo> element of outbound routers. Mule handles the underlying configuration. MQMT_DATAGRAM: The message does not require a reply. MQMT_REPLY: The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the <ReplyTo> configured on the outbound router. Mule automatically configures the request to control how to set the MessageId and CorrelationId of the reply. MQMT_REPORT: The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Sends the message to the queue indicated by the <ReplyTo> configuration of the message descriptor of the original message.
characterSet	integer	no		If set, this property overrides the coded character set property of the destination queue or topic.

persistentDelivery	boolean	no		If set to true, the JMS provider logs the message to stable storage as it is sent so that it can be recovered if delivery is unsuccessful. A client marks a message as persistent if the application will have problems if the message is lost in transit. A client marks a message as non-persistent if an occasional lost message is tolerable. Clients use delivery mode to tell a JMS provider how to balance message transport reliability/throughput. Delivery mode only covers the transport of the message to its destination. Retention of a message at the destination until its receipt is acknowledged is not guaranteed by a PERSISTENT delivery mode. Clients should assume that message retention policies are set administratively. Message retention policy governs the reliability of message delivery from destination to message consumer. For example, if a client's message storage space is exhausted, some messages as defined by a site-specific message retention policy may be dropped. A message is guaranteed to be delivered once and only once by a JMS provider if the delivery mode of the message is persistent and if the destination has a sufficient message retention policy.
timeToLive	long	no		Define the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system. Time to live is set to zero (forever) by default.
priority	substitutablePriorityNumber	no		Sets the message priority. JMS defines a ten-level priority value with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. JMS does not require that a provider strictly implement priority ordering of messages. However, it should do its best to deliver expedited messages ahead of normal messages.
targetClient		no		Specifies whether this is in JMS or non-JMS format. Possible values are: JMS_COMPLIANT or NONJMS_MQ (default).

cache: Unexpected program error: java.lang.NullPointerException

### Message To Object Transformer

Converts a com.ibm.jms.JMSMessage or sub-type into an object by extracting the message payload.

cache: Unexpected program error: java.lang.NullPointerException

### Object To Message Transformer

Converts an object back into a com.ibm.jms.JMSMessage.

cache: Unexpected program error: java.lang.NullPointerException

### Transaction

Transactions allow a series of operations to be grouped together so that they can be rolled back if a failure occurs. Set the action (such as ALWAYS\_BEGIN or JOIN\_IF\_POSSIBLE) and the timeout setting for the transaction.

Your Rating: 

Results:  0 rates

## Multicast Transport Reference

### Multicast Module Reference

## Introduction

The [Multicast](#) transport allow sending messages to or receiving messages from groups of multicast sockets. It is implemented on top of [UDP](#) and is highly scalable since knowledge of the receivers is not required.

## Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
Multicast	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>	✓	✓	✓	✗	✗	✗	one-way, request-response	request-response	org.mule.transport.multicast.MulticastTransport

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name and artifact name for this transport in [Maven](#)

## Namespace and Syntax

XML namespace:

```
xmlns:multicast="http://www.mulesoft.org/schema/mule/multicast"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/multicast
http://www.mulesoft.org/schema/mule/multicast/3.1/mule-multicast.xsd
```

Connector syntax:

```
<multicast:connector name="mcConnector" receiveBufferSize="1024" sendBufferSize="1024"
    timeout="0" keepSendSocketOpen="false" broadcast="false"
    timeToLive="127" loopback="true' />
```

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```
<multicast:inbound-endpoint host="localhost" port="65433"/>
```

## 2. Non-prefixed URI:

```
<inbound-endpoint address="multicast://localhost:65433" />
```

See the sections below for more information.

## Considerations

IP Multicasting is a service provided by IP (the internet protocol layer), that allows one-to-many communication. The most common use of IP Multicasting is to send UDP datagrams to multiple sockets located on different systems. Mule supports this with the Multicast transport. Note that, except for the communication being many-to-one instead of one-to-one, the Multicast transport is very similar to the [UDP transport](#) and the same considerations should be observed.

As shown in the examples below, the Multicast transport has two purposes:

- Send messages to a group of IP Multicasting sockets
- Read messages sent to a group of IP Multicasting sockets

## Features

The Multicasting module allows a Mule application both to send and receive IP Multicasting datagrams, and to declaratively customize the following features of IP Multicasting (with the standard name for each feature, where applicable):

- The timeout for sending or receiving messages (SO\_TIMEOUT).
- Whether to allow sending broadcast messages (SO\_BROADCAST).
- Whether to close a socket after sending a message.
- The maximum size of messages that can be received.
- The time to live for the packets that are sent
- Whether to loop packets back to the local socket

Multicast endpoints can be used in one of two ways:

- To receive an IP Multicasting datagram, create an inbound Multicast endpoint.
- To send an IP Multicasting datagram, create an outbound Multicast endpoint.

## Usage

To use Multicast endpoints

- Add the MULE Multicast namespace to your configuration:
  - Define the multicast prefix using xmlns:multicast="http://www.mulesoft.org/schema/mule/multicast"
  - Define the schema location with <http://www.mulesoft.org/schema/mule/multicast> [/3.1/mule-multicast.xsd](#)
- Define one or more connectors for Multicast endpoints.
  - Create a Multicast connector:

```
<multicast:connector name="multicastConnector" />
```
- Create Multicast endpoints.
  - Datagrams will be received on inbound endpoints. The bytes in the datagram will become the message payload.
  - Datagrams will be sent to outbound endpoints. The bytes in the message payload will become the datagram.
  - Both kinds of endpoints are identified by a host name and a port. The host name, in this case, is one of the standard IP multicast addresses defined [here](#). When a datagram is sent to a multicasting host/port combination, all sockets subscribed to that host/port receive the message.

Multicast endpoints are always one-way.

## Example Configurations

### Mule Flow

### Copy datagrams from one port to another in a flow

```
<multicast:connector name="connector"/>

<flow name="copy">
    <multicast:inbound-endpoint host="224.0.0.0" port="4444" exchange-pattern="one-way" />
    <pass-through-router>
        <multicast:outbound-endpoint host="224.0.0.0" port="5555" exchange-pattern="one-way" />
    </pass-through-router>
</flow>
```

### Mule Service

### Copy datagrams from one port to another in a service

```
<multicast:connector name="connector"/>

<service name="copy">
    <inbound>
        <multicast:inbound-endpoint host="224.0.0.0" port="4444" exchange-pattern="one-way" />
    </inbound>
    <outbound>
        <multicast:outbound-endpoint host="224.0.0.0" port="5555" exchange-pattern="one-way" />
    </outbound>
</service>
```

The connector uses all default properties. The inbound endpoint receives multicasting datagrams and copies them to the outbound endpoint, which will copy them to a different multicasting group.

## Configuration Options

Multicast Connector attributes

Name	Description	Default
broadcast	set this to true to allow sending to broadcast ports	false
keepSendSocketOpen	Whether to keep the the socket open after sending a message	false
loopback	Whether to loop messages back to the socket that sent them	false
receiveBufferSize	This is the size of the largest (in bytes) datagram that can be received.	16 Kbytes
sendBufferSize	The size of the network send buffer	16 Kbytes
timeout	the timeout used for both sending and receiving	system default
timeToLive	how long the packet stays active. This is a number between 1 and 225	System default

## Configuration Reference

### *Element Listing*

cache: Unexpected program error: java.lang.NullPointerException

## Multicast Transport

The Multicast transport can dispatch Mule events using IP multicasting.

### *Connector*

## ***Inbound endpoint***

Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <inbound-endpoint...>

Name	Cardinality	Description

## ***Outbound endpoint***

Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <outbound-endpoint...>

Name	Cardinality	Description

## ***Endpoint***

Attributes of <endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <endpoint...>

Name	Cardinality	Description

## **Schema**

The schema for the Multicast module appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/multicast"

Targeting Schemas (1):

[mule-multicast.xsd](#)

Targeting Components:

4 global elements, 3 complexTypes, 1 attribute group

Schema Summary	
mule-multicast.xsd	<p>The Multicast transport can dispatch Mule events using IP multicasting.</p> <p>Target Namespace: <a href="http://www.mulesoft.org/schema/mule/multicast">http://www.mulesoft.org/schema/mule/multicast</a></p> <p>Defined Components:</p> <p>4 global elements, 3 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form:</p>

	<p>Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:</p> <p><a href="http://www.mulesoft.org/schema/mule/multicast/3.1/mule-multicast.xsd">http://www.mulesoft.org/schema/mule/multicast/3.1/mule-multicast.xsd</a>; see XML source</p> <p>Imports Schemas (4):</p> <p><a href="#">mule-schemadoc.xsd</a>, <a href="#">mule-udp.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p>
--	---

All Element Summary	
connector	<p>Type: <a href="#">anonymous</a> (extension of <a href="#">udp:udpConnectorType</a>)</p> <p>Content: complex, 15 <a href="#">attributes</a>, attr. wildcard, 5 elements</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-connector</a></p> <p>Defined: globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Includes: definitions of 2 <a href="#">attributes</a></p> <p>Used: never</p>
endpoint	<p>Type: <a href="#">globalEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-global-endpoint</a></p> <p>Defined: globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: never</p>
inbound-endpoint	<p>Type: <a href="#">inboundEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-inbound-endpoint</a></p> <p>Defined: globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: never</p>
outbound-endpoint	<p>Type: <a href="#">outboundEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-outbound-endpoint</a></p> <p>Defined: globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: never</p>

Complex Type Summary	
<a href="#">globalEndpointType</a>	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: at 1 <a href="#">location</a></p>
<a href="#">inboundEndpointType</a>	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: at 1 <a href="#">location</a></p>
<a href="#">outboundEndpointType</a>	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Used: at 1 <a href="#">location</a></p>

Attribute Group Summary	
<a href="#">addressAttributes</a>	<p>Content: 2 <a href="#">attributes</a></p> <p>Defined: globally in <a href="#">mule-multicast.xsd</a>; see XML source</p> <p>Includes:definitions of 2 <a href="#">attributes</a></p> <p>Used: at 3 <a href="#">locations</a></p>

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this module can be found here: [Multicast](#)

## Maven

The Multicast Module can be included with the following dependency:

```

<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-multicast</artifactId>
  <version>3.1.0</version>
</dependency>

```

## Extending the Multicast Transport

### Best Practices

### Notes

Before Mule 3.1.1, there were two different attributes for setting timeout on Multicast connectors, `sendTimeout` and `receiveTimeout`. It was necessary to set them to the same value. Now there is only `timeout` for either send or receive.

Your Rating:  Results:  0 rates

## POP3 Transport Reference

### POP3 Transport Reference

#### Introduction

The POP3 transport can be used for receiving messages from POP3 inboxes. The POP3S tranport connects to POP3 mailboxes using the `javax.mail` API.



#### Warning

POP3 Polling May Fail if geronimo-mail is in the CLASSPATH.  
For information on this problem, consult [MULE-4875](#).

TLS/SSL connections are made on behalf of an entity, which can be anonymous or identified by a certificate. The key store provides the certificates and associated private keys necessary for identifying the entity making the connection. Additionally, connections are made to trusted systems. The public certificates of trusted systems are stored in a trust store, which is used to verify that the connection made to a remote system matches the expected identity.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
POP3	JavaDoc SchemaDoc								one-way	one-way
POP3S	JavaDoc SchemaDoc								one-way	one-way

#### Legend

Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here

are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in Maven

## Namespace and Syntax

XML namespace:

```
xmlns:pop3 "http://www.mulesoft.org/schema/mule/pop3"
xmlns:pop3s "http://www.mulesoft.org/schema/mule/pop3s"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/pop3 http://www.mulesoft.org/schema/mule/pop3/3.1/mule-pop3.xsd
http://www.mulesoft.org/schema/mule/pop3s http://www.mulesoft.org/schema/mule/pop3s/3.1/mule-pop3s.xsd
```

Connector syntax:

```
<pop3:connector name="pop3Connector" backupEnabled="true" backupFolder="backup" checkFrequency="90000"
    deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED"/>
<pop3s:connector name="pop3sConnector" backupEnabled="true" backupFolder="backup" checkFrequency="90000"
    deleteReadMessages="false" mailboxFolder="INBOX" moveToFolder="PROCESSED"/>
<pop3s:tls-client path="clientKeystore" storePassword="mulepassword" />
<pop3s:tls-trust-store path="greenmail-truststore" storePassword="password" />
</pop3s:connector>
```

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```
<pop3:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>
<pop3s:inbound-endpoint user="bob" password="password" host="localhost" port="65433"/>
```

2. Non-prefixed URI:

```
<inbound-endpoint address="pop3://bob:password@localhost:65433"/>
<inbound-endpoint address="pop3s://bob:password@localhost:65433"/>
```

See the sections below for more information.

## Features

- Simple to configure email access on inbound endpoints: including authentication information and check frequency
- Automate the handling of email attachments
- Automatically back up messages to a specified folder
- Automatically delete read messages
- Easy to configure tls security

## Usage

If you want to include the POP3 email transport in your configuration, these are the namespaces you need to define:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:imap="http://www.mulesoft.org/schema/mule/imap"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/pop3 http://www.mulesoft.org/schema/mule/imap/3.1/mule-pop3.xsd">
  ...

```

For the secure version, you would use the following:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:pop3s="http://www.mulesoft.org/schema/mule/pop3s"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
    http://www.mulesoft.org/schema/mule/pop3s
    http://www.mulesoft.org/schema/mule/pop3s/3.1/mule-pop3s.xsd">

```

Then you need to configure your connector and endpoints as described below.

### **Configuration Example**

Say you had a business and wanted to take orders through email attachments. After you receive the email, you want to save the order attachments so they could be picked up by your order fulfillment process. The following Mule configuration checks an email box for emails, and saves the attachments to the local disk, where it can be picked up from a separate fulfillment process:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:pop3="http://www.mulesoft.org/schema/mule/pop3"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:file="http://www.mulesoft.org/schema/mule/file"
    xmlns:email="http://www.mulesoft.org/schema/mule/email"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
        http://www.mulesoft.org/schema/mule/pop3 http://www.mulesoft.org/schema/mule/pop3/3.1/mule-pop3.xsd
        http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <pop3:connector name="pop3Connector" />

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <model name="test">
        <service name="incoming-orders">
            <inbound>
                <pop3:inbound-endpoint user="bob" password="password" host="mailServer"
                    port="110" transformer-refs="returnAttachments"/>
            </inbound>
            <outbound>
                <list-message-splitter-router>
                    <file:outbound-endpoint path=".//received" outputPattern=
"##[function:datestamp].dat">
                        <expression-transformer>
                            <return-argument expression="payload.inputStream" evaluator="groovy" />
                        </expression-transformer>
                    </file:outbound-endpoint>
                </list-message-splitter-router>
            </outbound>
        </service>
    </model>
</mule>

```

The built-in transformer is declared on `returnAttachments` and gets the list of email attachments. This transformer is then applied to the pop3 inbound endpoint defined at `user="bob"`. Then we define a list `list-message-splitter-router` on `outbound` which will iterate through all of the email attachments. Next we define a file `outbound` endpoint which will write the attachment to the `'./received'` directory with a datestamp as the file name on `##[function:datestamp].dat`. `expression-transformer` defines a simple groovy expression which gets the `inputStream` of the attachment to write the file.

Here is the flow-based version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:pop3="http://www.mulesoft.org/schema/mule/pop3"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:file="http://www.mulesoft.org/schema/mule/file"
    xmlns:email="http://www.mulesoft.org/schema/mule/email"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
        http://www.mulesoft.org/schema/mule/pop3 http://www.mulesoft.org/schema/mule/pop3/3.1/mule-pop3.xsd
        http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <pop3:connector name="pop3Connector" />

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <flow name="incoming-orders">
        <pop3:inbound-endpoint user="bob" password="password" host="mailServer"
            port="110" transformer-refs="returnAttachments"/>
        <collection-splitter/>
        <file:outbound-endpoint path="./received" outputPattern="#{function:datestamp}.dat">
            <expression-transformer>
                <return-argument expression="payload.inputStream" evaluator="groovy" />
            </expression-transformer>
        </file:outbound-endpoint>
    </flow>
</mule>

```

Here is the secure version of the same configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:pop3s="http://www.mulesoft.org/schema/mule/pop3s"
    xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
    xmlns:file="http://www.mulesoft.org/schema/mule/file"
    xmlns:email="http://www.mulesoft.org/schema/mule/email"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/file http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
        http://www.mulesoft.org/schema/mule/pop3s http://www.mulesoft.org/schema/mule/pop3s/3.1/mule-pop3s.xsd
        http://www.mulesoft.org/schema/mule/email http://www.mulesoft.org/schema/mule/email/3.1/mule-email.xsd
        http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">

    <pop3s:connector name="pop3sConnector">
        <pop3s:tls-client path="clientKeystore" storePassword="mulepassword" />
        <pop3s:tls-trust-store path="greenmail-truststore" storePassword="password" />
    </pop3s:connector>

    <expression-transformer name="returnAttachments">
        <return-argument evaluator="attachments-list" expression="*" optional="false"/>
    </expression-transformer>

    <file:connector name="fileName">
        <file:expression-filename-parser/>
    </file:connector>

    <model name="test">
        <service name="incoming-orders">
            <inbound>
                <pop3s:inbound-endpoint user="bob" password="password" host="mailServer"
                    port="110" transformer-refs="returnAttachments"/>
            </inbound>
            <outbound>
                <list-message-splitter-router>
                    <file:outbound-endpoint path=".received" outputPattern=
"##[function:datetimestamp].dat">
                        <expression-transformer>
                            <return-argument expression="payload.inputStream" evaluator="groovy" />
                        </expression-transformer>
                    </file:outbound-endpoint>
                </list-message-splitter-router>
            </outbound>
        </service>
    </model>
</mule>

```

The pop3s connector has tls client and server keystore information as defined on . The built-in transformer is declared on and gets the list of email attachments. This transformer is then applied to the inbound endpoint on . Then we define a list list-message-splitter-router on which will iterate through all of the email attachments. Next we define a file outbound endpoint which will write the attachment to the './received' directory with a timestamp as the file name on . defines a simple groovy expression which gets the inputStream of the attachment to write the file.

## Configuration Reference

### Connectors

The POP3 connector supports all the common connector attributes and properties and the following additional attributes:

Attribute	Description	Default	Required
backupEnabled	Whether to save copies to the backup folder	False	No
backupFolder	The folder where messages are moved after they have been read.		No

checkFrequency	Period (ms) between poll connections to the server.	60000	Yes
mailboxFolder	The remote folder to check for email.	INBOX	No
deleteReadMessages	Whether to delete messages from the server when they have been downloaded. If set to false, the messages are set to defaultProcessMessageAction attribute value.	true	No
moveToFolder	The remote folder to move mail to once it has been read. It is recommended that 'deleteReadMessages' is set to false when this is used. This is very useful when working with public email services such as GMail where marking messages for deletion doesn't work. Instead set the @moveToFolder=GMail/Trash.		No
defaultProcessMessageAction	The action performed if the deleteReadMessages attribute is set to false. Valid values are: ANSWERED, DELETED, DRAFT, FLAGGED, RECENT, SEEN, USER, and NONE	SEEN	No

For the secure version, the following elements are also required:

Element	Description
tls-client	Configures the client key store with the following attributes: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the keystore that contains public certificates and private keys for identification</li> <li>storePassword: The password used to protect the keystore</li> <li>class: The type of keystore used (a Java class name)</li> </ul>
tls-trust-store	Configures the trust store. The attributes are: <ul style="list-style-type: none"> <li>path: The location (which will be resolved relative to the current classpath and file system, if possible) of the trust store that contains public certificates of trusted servers</li> <li>storePassword: The password used to protect the trust store</li> </ul>

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:spring="http://www.springframework.org/schema/beans"
       xmlns:pop3="http://www.mulesoft.org/schema/mule/pop3"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
           http://www.mulesoft.org/schema/mule/pop3
           http://www.mulesoft.org/schema/mule/pop3/3.0/mule-pop3.xsd">

    <pop3:connector name="pop3Connector" backupEnabled="true" backupFolder="newBackup"
    checkFrequency="1234"
        mailboxFolder="newMailbox" deleteReadMessages="false"/>

    ...

```

Secure version:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core/2.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:pop3s="http://www.mulesoft.org/schema/mule/pop3s"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/pop3s
          http://www.mulesoft.org/schema/mule/pop3s/3.0/mule-pop3s.xsd">

<pop3s:connector name="pop3sConnector">
    <pop3s:tls-client path="clientKeystore" storePassword="mulepassword" />
    <pop3s:tls-trust-store path="greenmail-truststore" storePassword="password" />
</pop3s:connector>
<model name="test">
    <service name="relay">
        <inbound>
            <pop3s:inbound-endpoint user="bob" password="password" host="localhost" port="65436" />
        </inbound>
    ...

```

## Endpoints

POP3 and POP3S endpoints include details about connecting to a POP3 mailbox. You configure the endpoints just as you would with any other transport, with the following additional attributes:

- user: the user name of the mailbox owner
- password: the password of the user
- host: the IP address of the POP3 server, such as www.mulesoft.com, localhost, or 127.0.0.1
- port: the port number of the POP3 server. If not set for the POP3S connector, the default port is 995.

For example:

```
<pop3:inbound-endpoint user="bob" password="foo" host="pop.gmail.com" checkFrequency="3000" />
```

or if using a POP3S connector:

```
<pop3s:inbound-endpoint user="bob" password="foo" host="pop.gmail.com" checkFrequency="3000" />
```

You can also define the endpoints using a URI syntax:

```
<inbound-endpoint address="pop3://bob:foo@pop.gmail.com:995"/>
<inbound-endpoint address="pop3s://bob:foo@pop.gmail.com:995"/>
```

This will log into the bob mailbox on pop.gmail.com using password foo (using the default port 995 for the POP3S endpoint).

**cache: Unexpected program error: java.lang.NullPointerException**

## Transformers

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description
------	-------------

<code>email-to-string-transformer</code>	Converts an email message to string format.
<code>string-to-email-transformer</code>	Converts a string message to email format.
<code>object-to-mime-transformer</code>	Converts an object to MIME format.
<code>mime-to-bytes-transformer</code>	Converts a MIME message to a byte array.
<code>bytes-to-mime-transformer</code>	Converts a byte array message to MIME format.

Here is how you define transformers in your Mule configuration file:

```
<email:bytes-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name=""
returnClass="" xsi:type="" />
<email:email-to-string-transformer encoding="" ignoreBadInput="" mimeType="" name=""
returnClass="" xsi:type="" />
<email:mime-to-bytes-transformer encoding="" ignoreBadInput="" mimeType="" name=""
returnClass="" xsi:type="" />
<email:object-to-mime-transformer encoding="" ignoreBadInput="" mimeType="" name=""
returnClass="" />
useInboundAttachments="true" useOutboundAttachments="true"/>
{Note}Need to explain attributes somewhere; can we pull them in from xsd?{Note}
<email:string-to-email-transformer encoding="" ignoreBadInput="" mimeType="" name=""
returnClass="" xsi:type="" />
```

Each transformer supports all the common transformer attributes and properties:

**cache:** Unexpected program error: java.lang.NullPointerException

### Transformer

A reference to a transformer defined elsewhere.

#### Attributes of <transformer...>

Name	Type	Required	Default	Description
<code>name</code>	<code>name (no spaces)</code>	<code>no</code>		<i>Identifies the transformer so that other elements can reference it. Required if the transformer is defined at the global level.</i>
<code>returnClass</code>	<code>string</code>	<code>no</code>		<i>The class of the message generated by the transformer. This is used if transformers are auto-selected and to validate that the transformer returns the correct type. Note that if you need to specify an array type you need postfix the class name with '[]'. For example, if you want return a an Orange[], you set the return class to 'org.mule.tck.testmodels.fruit.Orange[]'.</i>
<code>ignoreBadInput</code>	<code>boolean</code>	<code>no</code>		<i>Many transformers only accept certain classes. Such transformers are never called with inappropriate input (whatever the value of this attribute). If a transformer forms part of a chain and cannot accept the current message class, this flag controls whether the remaining part of the chain is evaluated. If true, the next transformer is called. If false the chain ends, keeping the result generated up to that point.</i>
<code>encoding</code>	<code>string</code>	<code>no</code>		<i>String encoding used for transformer output.</i>
<code>mimeType</code>	<code>string</code>	<code>no</code>		<i>The mime type, e.g. text/plain or application/json</i>
<code>ref</code>	<code>string</code>	<code>yes</code>		<i>The name of the transformer to use.</i>

#### Child Elements of <transformer...>

Name	Cardinality	Description
------	-------------	-------------

The object-to-mime-transformer has the following attributes:

Attribute	Description	Default Value
useInboundAttachments	Whether to transform inbound attachment in the input message into MIME parts.	true
useOutboundAttachments	Whether to transform outbound attachment in the input message into MIME parts.	true

To use these transformers, make sure you include the 'email' namespace in your mule configuration.

Your Rating:  Results:  1 rates

## Filters

Filters can be set on an endpoint to filter out any unwanted messages. The Email transport provides a couple of filters that can either be used directly or extended to implement custom filtering rules.

Filter	Description
org.mule.providers.email.filters.AbstractMailFilter	A base filter implementation that must be extended by any other mail filter.
org.mule.providers.email.filters.MailSubjectRegExFilter	Applies a regular expression to a Mail Message subject.

This is how you define the MailSubjectRegExFilter in your Mule configuration:

```
<message-property-filter pattern="to=barney@mule.org"/>
```

The 'pattern' attribute is a regular expression pattern. This is defined as java.util.regex.Pattern.

Your Rating:  Results:  1 rates

## Exchange patterns / features of the transport

(see [transport matrix](#))

## Schema Reference

You can view the full schema for POP3 email transport [here](#). The secure version is [here](#)

## Java API Reference

The Javadoc for this transport can be found [here](#).

## Maven module

The email transports are implemented by the mule-transport-email module. You can find the source for the email transport under transports/email.

If you are using maven to build your application, use the following dependency snippet to include the email transport in your project:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-email</artifactId>
</dependency>
```

## Mule-Maven Dependencies

If you are building Mule ESB from source or including Mule artifacts in your Maven project, it may be necessary to add the 'mule-deps' repository to your Maven configuration. This repository contains third-party binaries which may not be in any other public Maven repository.

To add the 'mule-deps' repository to your Maven project, add the following to your pom.xml:

```

<repositories>
  <repository>
    <id>mule-deps</id>
    <name>Mule Dependencies</name>
    <url>http://dist.codehaus.org/mule/dependencies/maven2</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>

```

Your Rating:  Results:  4 rates

## Limitations

The following known limitations affect email transports:

- Retry policies do not work with email transports
- Timeouts are not supported in email transports
- Can't send same object to different email users
- MailSubjectRegExFilter cannot handle mails with attachments

Your Rating:  Results:  2 rates



### Escape Your Credentials

If you use a URI-style endpoint and you include the user name and password, escape any characters that are illegal for URIs, such as the @ character. For example, if the user name is user@myco.com, you should enter it as user%40myco.com.

Your Rating:  Results:  1 rates

## Quartz Transport Reference

### Quartz Transport Reference

[ About Cron Expressions ] [ About Jobs ] [ Connector ] [ Outbound Endpoint ] [ Inbound Endpoint ] [ Endpoint ] [ Abstract Job ] [ Abstract Inbound Job ] [ Event Generator Job ] [ Endpoint Polling Job ] [ Scheduled Dispatch Job ] [ Custom Job ] [ Custom Job From Message ]

The Quartz transport provides support for scheduling events and for triggering new events. An inbound quartz endpoint can be used to trigger inbound events that can be repeated, such as every second. Outbound quartz endpoints can be used to schedule an existing event to fire at a later date. Users can create schedules using cron expressions, and events can be persisted in a database.

This transport makes use of the [Quartz Project at Open Symphony](#). The Quartz site has more generic information about how to work with Quartz.

### About Cron Expressions

A cron expression is a string comprised by six or seven fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows:

Field Name	Mandatory	Allowed Values	Allowed Special Chars
Seconds	YES	0-59	, - * /
Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of Month	YES	1-31	, - * ? / L W C
Month	YES	1-12 or JAN-DEC	, - * /
Day of Week	YES	1-7 or SUN-SAT	, - * ? / L C #

Year	NO	empty, 1970-2099	, - * /
------	----	------------------	---------

Cron expressions can be as simple as this:

\* \* \* \* ? \*

or more complex, like this:

0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010.

Following are some examples:

Expression	Behavior
0 0 12 * * ?	Fire at 12pm (noon) every day
0 15 10 ? * *	Fire at 10:15am every day
0 15 10 * * ?	Fire at 10:15am every day
0 15 10 * * ? *	Fire at 10:15am every day
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day

If you are not familiar with cron syntax, here is a good tutorial.

## About Jobs

Jobs are used to perform an action when a time trigger occurs from the Quartz transport. Mule provides a number of jobs for generating and scheduling events. These are detailed below. Users can also write their own jobs and hook them in using the custom-job type included with Mule.

cache: Unexpected program error: java.lang.NullPointerException

## Connector

The Quartz connector is used to configure the default behavior for Quartz endpoints that reference the connector. Note if there is only one Quartz connector configured, all Quartz endpoints will use that connector.

### Attributes of <connector...>

Name	Type	Required	Default	Description
scheduler-ref	string	no		Provides an implementation of the Quartz Scheduler interface. If no value is provided, a scheduler is retrieved from the StdSchedulerFactory. If no properties are provided, the getDefaultScheduler method is called. Otherwise, a new factory instance is created using the given properties, and a scheduler is retrieved using the getScheduler method.

### Child Elements of <connector...>

Name	Cardinality	Description
factory-property	0..*	Set a property on the factory (see scheduler-ref).

For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:quartz="http://www.mulesoft.org/schema/mule/quartz"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/quartz
          http://www.mulesoft.org/schema/mule/quartz/3.0/mule-quartz.xsd">

    <quartz:connector name="quartzConnector1" scheduler-ref="myScheduler">
        <factory-property key="org.quartz.scheduler.instanceName" value="MuleScheduler1"/>
        <factory-property key="org.quartz.threadPool.class" value="org.quartz.simpl.SimpleThreadPool"/>
        <factory-property key="org.quartz.threadPool.threadCount" value="3"/>
        <factory-property key="org.quartz.scheduler.rmi.proxy" value="false"/>
        <factory-property key="org.quartz.scheduler.rmi.export" value="false"/>
        <factory-property key="org.quartz.jobStore.class" value="org.quartz.simpl.RAMJobStore"/>
    </quartz:connector>
    ...

```

cache: Unexpected program error: java.lang.NullPointerException

## Outbound Endpoint

An outbound Quartz endpoint allows existing events to be stored and fired at a later time/date. If you are using a persistent event store, the payload of the event must implement `java.io.Serializable`. You configure an `org.quartz.Job` implementation on the endpoint to tell it what action to take. Mule has some default jobs, but you can also write your own.

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
jobName	string	no		The name to associate with the job on the endpoint. This is only really used internally when storing events.
cronExpression	string	no		The cron expression to schedule events at specified dates/times. This attribute or repeatInterval is required. A cron expression is a string comprised by 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows: Field Name Mandatory Allowed Values Allowed Special Chars Seconds YES 0-59, - * / Minutes YES 0-59, - * / Hours YES 0-23, - * / Day of Month YES 1-31, - * ? / L W C Month YES 1-12 or JAN-DEC, - * / Day of Week YES 1-7 or SUN-SAT, - * ? / L C # Year NO Empty, 1970-2099, - * / Cron expressions can be as simple as this: * * * * ? * or more complex, like this: 0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010 Some examples: 0 0 12 * * ? Fire at 12pm (noon) every day 0 15 10 ? * * ? Fire at 10:15am every day 0 15 10 * * ? Fire at 10:15am every day 0 15 10 * * ? * Fire at 10:15am every day 0 15 10 * * ? 2005 Fire at 10:15am every day during the year 2005 0 * 14 * * ? Fire every minute starting at 2pm and ending at 2:59pm, every day 0 0/5 14 * * ? Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
repeatInterval	long	no		The number of milliseconds between two events. This attribute or cronExpression is required.
repeatCount	integer	no		The number of events to be scheduled. This value defaults to -1, which means that the events will be scheduled indefinitely.
startDelay	long	no		The number of milliseconds that will elapse before the first event is fired.

### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
abstract-job	1..1	A placeholder for Quartz jobs that can be set on the endpoint.

cache: Unexpected program error: java.lang.NullPointerException

## Inbound Endpoint

A Quartz inbound endpoint can be used to generate events. It is most useful when you want to trigger a service at a given interval (or cron expression) rather than have an external event trigger the service.

#### **Attributes of <inbound-endpoint...>**

Name	Type	Required	Default	Description
jobName	string	no		The name to associate with the job on the endpoint. This is only really used internally when storing events.
cronExpression	string	no		The cron expression to schedule events at specified dates/times. This attribute or repeatInterval is required. A cron expression is a string comprised by 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows: Field Name Mandatory Allowed Values Allowed Special Chars Seconds YES0-59, - * / Minutes YES0-59, - * / Hours YES0-23, - * / Day of Month YES1-31, - * ? / L W C Month YES1-12 or JAN-DEC, - * / Day of Week YES1-7 or SUN-SAT, - * ? / L C # Year NOEmpty, 1970-2099, - * / Cron expressions can be as simple as this: * * * * ? * or more complex, like this: 0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010 Some examples: 0 0 12 * * ? Fire at 12pm (noon) every day 0 15 10 * * ? Fire at 10:15am every day 0 15 10 * * ? Fire at 10:15am every day 0 15 10 * * ? Fire every minute starting at 2pm and ending at 2:59pm, every day 0 0/5 14 * * ? Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
repeatInterval	long	no		The number of milliseconds between two events. This attribute or cronExpression is required.
repeatCount	integer	no		The number of events to be scheduled. This value defaults to -1, which means that the events will be scheduled indefinitely.
startDelay	long	no		The number of milliseconds that will elapse before the first event is fired.

#### **Child Elements of <inbound-endpoint...>**

Name	Cardinality	Description
abstract-job	1..1	A placeholder for Quartz jobs that can be set on the endpoint.

cache: Unexpected program error: java.lang.NullPointerException

#### **Endpoint**

A global endpoint that can be used as a template to create inbound and outbound Quartz endpoints. Common configuration can be set on a global endpoint and then referenced using the @ref attribute on the local endpoint. Note that because jobs sometimes only work on inbound or outbound endpoints, they have to be set on the local endpoint.

#### **Attributes of <endpoint...>**

Name	Type	Required	Default	Description
stateful	boolean	no		Determines if the job is persistent. If so, the job detail state will be persisted for each request. More importantly, each job triggered will execute sequentially. If the Job takes longer than the next trigger the next job will wait for the current job to execute.
jobName	string	no		The name to associate with the job on the endpoint. This is only really used internally when storing events.
cronExpression	string	no		The cron expression to schedule events at specified dates/times. This attribute or repeatInterval is required. A cron expression is a string comprised by 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of the allowed special characters for that field. The fields are as follows: Field Name Mandatory Allowed Values Allowed Special Chars Seconds YES0-59, - * / Minutes YES0-59, - * / Hours YES0-23, - * / Day of Month YES1-31, - * ? / L W C Month YES1-12 or JAN-DEC, - * / Day of Week YES1-7 or SUN-SAT, - * ? / L C # Year NOEmpty, 1970-2099, - * / Cron expressions can be as simple as this: * * * * ? * or more complex, like this: 0 0/5 14,18,3-39,52 ? JAN,MAR,SEP MON-FRI 2002-2010 Some examples: 0 0 12 * * ? Fire at 12pm (noon) every day 0 15 10 * * ? Fire at 10:15am every day 0 15 10 * * ? Fire at 10:15am every day 0 15 10 * * ? Fire every minute starting at 2pm and ending at 2:59pm, every day 0 0/5 14 * * ? Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day

repeatInterval	long	no		The number of milliseconds between two events. This attribute or cronExpression is required.
repeatCount	integer	no		The number of events to be scheduled. This value defaults to -1, which means that the events will be scheduled indefinitely.
startDelay	long	no		The number of milliseconds that will elapse before the first event is fired.

#### **Child Elements of <endpoint...>**

Name	Cardinality	Description
abstract-job	0..1	A placeholder for Quartz jobs that can be set on the endpoint.

cache: Unexpected program error: java.lang.NullPointerException

#### **Abstract Job**

A placeholder for Quartz jobs that can be set on the endpoint.

#### **Attributes of <abstract-job...>**

Name	Type	Required	Default	Description
groupName	string	no		The group name of the scheduled job
jobGroupName	string	no		The job group name of the scheduled job.

cache: Unexpected program error: java.lang.NullPointerException

#### **Abstract Inbound Job**

A placeholder for Quartz jobs that can be set on inbound endpoints only.

#### **Attributes of <abstract-inbound-job...>**

Name	Type	Required	Default	Description
groupName	string	no		The group name of the scheduled job
jobGroupName	string	no		The job group name of the scheduled job.

cache: Unexpected program error: java.lang.NullPointerException

#### **Event Generator Job**

An inbound endpoint job that will trigger a new event for the service according to the schedule on the endpoint. This is useful for periodically triggering a service without the need for an external event to occur.

#### **Child Elements of <event-generator-job...>**

Name	Cardinality	Description
payload	0..1	The payload of the newly created event. The payload can be a reference to a file, fixed string, or object configured as a Spring bean. If this value is not set, an event will be generated with an org.mule.transport.NullPayload instance.

Following is an example:

▶ Click here to expand...

```

<service name="testService1">
  <description>
    This configuration will create an inbound event for testService1 at 12 noon every day. The event payload will always have the same value 'foo'.
  </description>

  <inbound>
    <quartz:inbound-endpoint name="qEP1" cronExpression="0 0 12 * * ?" jobName="job1" connector-ref="quartzConnector1">
      <quartz:event-generator-job>
        <quartz:payload>foo</quartz:payload>
      </quartz:event-generator-job>
    </quartz:inbound-endpoint>
  </inbound>
</service>

<service name="testService2">
  <description>
    This configuration will create an inbound event for testService2 every 1 second indefinitely. The event payload will always have the same value, which the contents of the file 'payload-data.txt'. The file can be on the classpath or on the local file system.
  </description>

  <inbound>
    <quartz:inbound-endpoint name="qEP2" repeatCount="10" repeatInterval="1000" jobName="job2" connector-ref="quartzConnector1">
      <quartz:event-generator-job>
        <quartz:payload file="payload-data.txt"/>
      </quartz:event-generator-job>
    </quartz:inbound-endpoint>
  </inbound>
</service>

```

cache: Unexpected program error: java.lang.NullPointerException

## Endpoint Polling Job

An inbound endpoint job that can be used to periodically read from an external source (via another endpoint). This can be useful for triggering time-based events from sources that do not support polling or for simply controlling the rate in which events are received from the source.

### ***Child Elements of <endpoint-polling-job...>***

Name	Cardinality	Description
job-endpoint	0..1	A reference to another configured endpoint from which events will be received.

For example:

► Click here to expand...

```

<service name="testService5">
  <description>
    The endpoint polling Job will try and perform a 'request' on any Mule endpoint. If a result is received it will be handed off to this 'testService5' service for processing. The trigger will fire every 5 minutes starting at 2pm and ending at 2:55pm, every day. during this period the job will check the file directory /N/drop-data/in every 5 minutes to see if any event data is available. The request will timeout after 4 seconds if there are no files in the directory.
  </description>

  <inbound>
    <quartz:inbound-endpoint name="qEP5" cronExpression="0 0/5 14 * * ?" jobName="job5"
      connector-ref="quartzConnector1">
      <quartz:endpoint-polling-job>
        <quartz:job-endpoint address="file:///N/drop-data/in" timeout="4000"/>
      </quartz:endpoint-polling-job>
    </quartz:inbound-endpoint>
  </inbound>
</service>

```

cache: Unexpected program error: java.lang.NullPointerException

## Scheduled Dispatch Job

An outbound job that will schedule a job for dispatch at a later time/date. The event will get dispatched using the configured endpoint reference.

### **Child Elements of <scheduled-dispatch-job...>**

Name	Cardinality	Description
job-endpoint	0..1	The endpoint used to dispatch the scheduled event. The preferred approach is to create a global endpoint and reference it using the ref attribute. However, you can also use the address attribute to define a URI endpoint (which supports expressions). You can use the timeout attribute to specify an arbitrary time-out value associated with the endpoint that can be used by jobs that block waiting to receive events.

For example:

► Click here to expand...

```

<service name="testService6">
  <description>
    This outbound Quartz endpoint will receive an event after the component has processed it and store it in the event store. When the trigger kicks in at 10:15am everyday it will dispatch the event on the endpoint referenced as 'scheduledDispatchEndpoint'. Since the 'repeatCount' is set to 0, the event will only be sent out once.
  </description>

  <inbound>
    <inbound-endpoint address="test://inbound6"/>
  </inbound>
  <test:component/>
  <outbound>
    <pass-through-router>
      <quartz:outbound-endpoint name="qEP6" repeatCount="0" cronExpression="0 15 10 * * ?"
        jobName="job6" connector-ref="quartzConnector1">
        <quartz:scheduled-dispatch-job>
          <quartz:job-endpoint ref="scheduledDispatchEndpoint"/>
        </quartz:scheduled-dispatch-job>
      </quartz:outbound-endpoint>
    </pass-through-router>
  </outbound>
</service>

```

cache: Unexpected program error: java.lang.NullPointerException

## Custom Job

A custom job can be configured on inbound or outbound endpoints. You can create and configure your own job implementation and use it on a Quartz endpoint. A custom job can be configured as a bean in the XML configuration and referenced using this job.

### Attributes of <custom-job...>

Name	Type	Required	Default	Description
groupName	string	no		The group name of the scheduled job
jobGroupName	string	no		The job group name of the scheduled job.
job-ref	string	no		The bean name or ID of the custom job to use when this job gets executed.

For example:

► Click here to expand...

```
<service name="testService5">
    <description>
        The endpoint polling Job will try and perform a 'request' on any Mule endpoint. If a result is received it will be handed off to this 'testService5' service for processing. The trigger will fire every 5 minutes starting at 2pm and ending at 2:55pm, every day. during this period the job will check the file directory /N/drop-data/in every 5 minutes to see if any event data is available. The request will timeout after 4 seconds if there are no files in the directory.
    </description>

    <inbound>
        <quartz:inbound-endpoint name="qEP5" cronExpression="0 0/5 14 * * ?" jobName="job5"
            connector-ref= "quartzConnector1">
            <quartz:endpoint-polling-job>
                <quartz:job-endpoint address="file:///N/drop-data/in" timeout="4000"/>
            </quartz:endpoint-polling-job>
        </quartz:inbound-endpoint>
    </inbound>
</service>
```

cache: Unexpected program error: java.lang.NullPointerException

## Custom Job From Message

Allows a job to be stored on the current message. This can only be used on outbound endpoints. When the message is received, the job is read and the job is added to the scheduler with the current message. This allows for custom scheduling behavior determined by the message itself. Usually the service or a transformer would create the job on the message based on application-specific logic. Any Mule-supported expressions can be used to read the job from the message. Typically, you add the job as a header, but an attachment could also be used.

### Attributes of <custom-job-from-message...>

Name	Type	Required	Default	Description
groupName	string	no		The group name of the scheduled job
jobGroupName	string	no		The job group name of the scheduled job.

For example:

► Click here to expand...

```

<service name="testService3">
    <description>
        This configuration will process a message and find a Job configured as a header called 'jobConfig' on the current message. We're using the test component here, but a real implementation will need to set a custom {{org.quartz.Job}} implementation as a header on the current message. Note that other expressions could be used to extract the job from an attachment or even a property within the payload itself.
    </description>

    <inbound>
        <inbound-endpoint address="test://inbound3"/>
    </inbound>
    <test:component/>
    <outbound>
        <pass-through-router>
            <quartz:outbound-endpoint name="qEP3" repeatInterval="1000" jobName="job3"
                connector-ref="quartzConnector1">
                <quartz:custom-job-from-message evaluator="header" expression="jobConfig"/>
            </quartz:outbound-endpoint>
        </pass-through-router>
    </outbound>
</service>

```

Your Rating: 

Results:  0 rates

## RMI Transport Reference

### RMI Transport Reference

The RMI transport can be used to send and receive Mule events over JRMP. This transport has a dispatcher that invokes an RMI method and a polling receiver that repeatedly does the same.

You configure the RMI transport as follows:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:rmi="http://www.mulesoft.org/schema/mule/rmi"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.mulesoft.org/schema/mule/core
        http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
        http://www.mulesoft.org/schema/mule/rmi
        http://www.mulesoft.org/schema/mule/rmi/3.0/mule-rmi.xsd">
    <!-- specify the actual class of the JNDI factory you want to use -->
    <spring:bean name="jndiFactory" class="org.mule.transport.rmi.MuleRMIFactory"/>

    <spring:bean name="jndiContext" factory-bean="jndiFactory" factory-method="create"/>

    <rmi:connector name="rmi" jndiContext-ref="jndiContext" securityPolicy="rmi.policy"/>

    <rmi:endpoint name="hello" host="localhost" port="1099" object="HelloServer"
        method="hello" methodArgumentTypes="java.lang.String"/>

```

The connector looks for the `method` and `methodArgumentTypes`. It uses the payload as the argument.

### JNP Connector

If you want to use the Java naming protocol to bind to remote objects, you can use the JNP connector instead simply by using the `jnp` namespace.

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:jnp="http://www.mulesoft.org/schema/mule/jnp"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.mulesoft.org/schema/mule/core
        http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
        http://www.mulesoft.org/schema/mule/jnp
        http://www.mulesoft.org/schema/mule/jnp/3.0/mule-jnp.xsd">

    <!-- specify the actual class of the JNDI factory you want to use -->
    <spring:bean name="jndiFactory" class="org.mule.transport.rmi.MuleRMIFactory"/>
    <spring:bean name="jndiContext" factory-bean="jndiFactory" factory-method="create"/>

    <jnp:connector name="jnp" jndiContext-ref="jndiContext" securityPolicy="rmi.policy"/>

    <jnp:endpoint name="Sender2" host="localhost" port="1099" object="MatchingUMO" method=
    "reverseString"/>
    ...

```

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

## Connector

### Attributes of <connector...>

Name	Type	Required	Default	Description
pollingFrequency	long	no		Period (ms) between polling connections.
securityManager-ref	string	no		Bean reference to the security manager that should be used.
securityPolicy	string	no		The security policy (file name) used to enable connections.
serverClassName	string	no		The target class name.
serverCodebase	string	no		The target method.

cache: Unexpected program error: java.lang.NullPointerException

## Endpoint

### Attributes of <endpoint...>

Name	Type	Required	Default	Description
host	string	no		The endpoint host name.
port	port number	no		The port number to use when a connection is made.
object	string	no		The class name of the object that is being invoked over RMI.
method	string	no		The name of the method to invoke.
methodArgumentTypes	string	no		Comma separated argument types of the method to invoke. For example, "java.lang.String".

Your Rating: ★★★★★

Results: ★★★★★ 0 rates

## Servlet Transport Reference

## Servlet Transport Reference

The Servlet transport provides integration with a servlet implementation. The implementing servlet does the work of receiving a request, and the Servlet transport then hands off the request to any receivers registered with it. There is no notion of a dispatcher for this connector, as it is triggered by a request and may or may not return a response. You specify the servlet URL as part of the connector configuration and then specify the endpoints just like any other [HTTP endpoints](#).

The Javadoc for this transport can be found [here](#). For more information about using Mule with servlet containers, see [Embedding Mule in a Java Application or Webapp](#). For information on using the RESTlet connector, click [here](#).

cache: Unexpected program error: java.lang.NullPointerException

### Connector

Servlet connector is a channel adapter between Mule and a servlet engine. It allows the MuleReceiverServlet to look up components interested in requests it receives via the servlet container.

#### Attributes of <connector...>

Name	Type	Required	Default	Description
servletUrl		no		The real URL on which the servlet container is bound. If this is not set, the WSDL may not be generated correctly when using CXF bound to a servlet endpoint.
useCachedHttpServletRequest	boolean	no	false	Whether to use a cached http servlet request

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:javax="http://www.mulesoft.org/schema/mule/servlet"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/servlet
          http://www.mulesoft.org/schema/mule/servlet/3.0/mule-servlet.xsd">

    <servlet:connector name="servletConnector" servletUrl="http://localhost:8888" />
    ...

```

You can also specify the servlet URL as part of the endpoint:

```
<servlet:inbound-endpoint path="http://localhost:8888" />
```

### Endpoints

Servlet endpoints accept the same attributes and child elements as HTTP endpoints. For more information, see [HTTP Transport Reference](#).

cache: Unexpected program error: java.lang.NullPointerException

### Http Request To Parameter Map

The <http-request-to-parameter-map> transformer returns a simple Map of the parameters sent with the HTTP Request. If the same parameter is given more than once, only the first value for it will be in the Map.

cache: Unexpected program error: java.lang.NullPointerException

### Http Request To Input Stream

The <http-request-to-input-stream> transformer converts an HttpServletRequest into an InputStream.

cache: Unexpected program error: java.lang.NullPointerException

## Http Request To Byte Array

The <http-request-to-byte-array> transformer converts an HttpServletRequest into an array of bytes by extracting the payload of the request.

Your Rating:  Results:  1 rates

## SFTP Transport Reference

### SFTP Transport Reference

[ Introduction ] [ Namespace and Syntax ] [ Considerations ] [ Features ] [ Usage ] [ Example Configurations ] [ Configuration Reference ] [ Schema ] [ Javadoc API Reference ] [ Maven ] [ Best Practices ] [ Notes ]

#### Introduction

The SFTP Transport allows files to be read and written to and from directories over SFTP. Unlike the VFS Transport, it can handle large files because it streams message payloads. The SFTP transport can be used as an inbound or an outbound endpoint. Files can be retrieved from an SFTP server and processed as Mule messages, or Mule messages can be uploaded as files to an SFTP server.

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Art
SFTP	<a href="#">JavaDoc</a> <a href="#">SchemaDoc</a>							one-way, request-response	one-way	<a href="#">org.mule.transport.sftp</a>

#### Namespace and Syntax

XML namespace:

```
xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/sftp http://www.mulesoft.org/schema/mule/sftp/3.1/mule-sftp.xsd
```

Connector syntax:

```
<sftp:connector name="sftp-default" archiveDir="" archiveTempReceivingDir="/tmp/get"
archiveTempSendingDir="/tmp/send" autoDelete="true"
duplicateHandling="throwException" fileAge="60000"
identityFile="/home/test/.ssh/id_rsa" keepFileOnError="false"
outputPattern="#{header:originalfilename}" passphrase="passPhrase"
pollingFrequency="10000" sizeCheckWaitTime="30000"
tempDirInbound="/tmp/inbound" tempDirOutbound="/tmp/outbound"
useTempFileTimestampSuffix="true" />
```

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```

<sftp:inbound-endpoint user="mule" password="test123" path="/tmp/sftp"
    host="myhost.com"
    pollingFrequency="500"
    name="inboundEndpoint1"/>
<sftp:outbound-endpoint user="mule" password="test123" path="/tmp/sftp"
    host="myhost.com" name="outboundEndpoint2"
    keepFileOnError="false"/>

```

## 2. Non-prefixed URI:

```

<inbound-endpoint address="sftp://mule:test123@myhost.com/tmp/sftp"/>
<outbound-endpoint address="sftp://mule:test123@myhost.com/tmp/sftp"/>

```

See the sections below for more information.

## Considerations

You would use the SFTP transport if you need to download from or upload to a secured resource accessible via SFTP. This transport does not currently support transactions or retry policies. Some uses for the SFTP transport are downloading data into a database and picking up files and uploading them via SFTP. You can use this transport to implement the file transfer Enterprise Integration Pattern. As explained in the EIP book, the file transfer pattern allows you to loosely couple two applications together, with delays in processing time. If your integration is time-sensitive, you may want to look at implementing the messaging pattern with the [JMS trasnport](#) which can give you closer to real-time processing.

Using the SFTP transport allows you to optionally use streaming support for larger files and asynchronously chain other endpoints with an SFTP endpoint. It also allows you to use Mule ESBs robust error handling in your Mule application.

The examples on this page will show you how to define SFTP inbound and outbound endpoints in your Mule application.

## Features

- Streaming support of resources
- For inbound endpoints, poll the resource at a specified interval
- For outbound endpoints, choices on how to handle duplicate files: throw and exception, overwrite, append a sequence number to the file name

## Usage

If you want to include the SFTP transport in your configuration, these are the namespaces you need to define:

```

<?xml version="1.0" encoding="utf-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
    xsi:schemaLocation="
        http://www.mulesoft.org/schema/mule/core
        http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/sftp
        http://www.mulesoft.org/schema/mule/sftp/3.1/mule-sftp.xsd">

```

Then define a connector:

```

<sftp:connector name="sftp-default" />

```

Finally define an inbound and/or outbound endpoint.

- Use an inbound endpoint if you want new files found on the sftp site to trigger a Mule flow or service
- Use an outbound endpoint if you want to upload files to an sftp site. These files typically start as Mule messages and are converted to files.

```

<sftp:inbound-endpoint
    name="inboundEndpoint1"
    connector-ref="sftp"
    address="sftp://user:password@host/~/data1"/>
<sftp:outbound-endpoint
    address="sftp://user:password@host/~/data"
    outputPattern="#{function:count}-#{function:systime}.dat"/>

```

#### Rules for using the Module/Transport

On the connector, you define the connection pool size, and your inbound/outbound temporary directories. The endpoint is where you define the authentication information, polling frequency, file name patterns, etc. See below for the full list of configuration options.

one-way and request-response exchange patterns are supported. If an exchange pattern is not defined, 'one-way' is the default.

This is a polling Transport. The inbound endpoint for SFTP uses polling to look for new files. The default is to check every second, but it can be changed via the 'pollingFrequency' attribute on the inbound endpoint.

Streaming is supported by the SFTP transport and is enabled by default. Retries and transactions are not currently supported.

### Example Configurations

The following example saves any files found on a remote sftp server to a local directory. This demonstrates using an sftp inbound endpoint and a file outbound endpoint.

#### Mule Flow

##### Downloading files from SFTP using a Flow

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/sftp
          http://www.mulesoft.org/schema/mule/sftp/3.1/mule-sftp.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <!-- This placeholder bean will let you import the properties from the sftp.properties file. -->
    <spring:bean id="property-placeholder" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:sftp.properties"/>
    </spring:bean>

    <flow name="sftp2file">
        <sftp:inbound-endpoint host="${sftp.host}" port="${sftp.port}" path="/home/test/sftp-files"
        user="${sftp.user}" password="${sftp.password}">
            <file:filename-wildcard-filter pattern="*.txt,*.xml"/>
        </sftp:inbound-endpoint>
        <file:outbound-endpoint path="/tmp/incoming" outputPattern="#{header:originalFilename}"/>
    </flow>
</mule>

```

#### Mule Service

## Downloading files from SFTP using a Service

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/sftp
          http://www.mulesoft.org/schema/mule/sftp/3.1/mule-sftp.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <!-- This placeholder bean will let you import the properties from the db.properties file. -->
    <spring:bean id="property-placeholder" class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:sftp.properties"/>
    </spring:bean>

    <model name="sftp2file">
        <service name="sftp2file-service">
            <inbound>
                <sftp:inbound-endpoint host="${sftp.host}" port="${sftp.port}" path=
"/home/test/sftp-files" user="${sftp.user}" password="${sftp.password}">
                    <file:filename-wildcard-filter pattern="*.txt,*.xml"/>
                </sftp:inbound-endpoint>
            </inbound>
            <outbound>
                <pass-through-router>
                    <file:outbound-endpoint path="/tmp/incoming" outputPattern=
"#*[header:originalFilename]" />
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>
```

A properties file which holds the sftp server login credentials is defined on . Next a sftp inbound endpoint is declared on which checks the '/home/test/sftp-files' directory for new files every one second by default. defines a file filter which will only send files ending with .txt or .xml to the outbound endpoint. Any conforming files found on the inbound endpoint are then written to the '/tmp/incoming' local directory with the same file name it had on the sftp server .

The following example uploads files found in a local directory to an SFTP server. This demonstrates using a file inbound endpoint and an sftp outbound endpoint.

### Mule Flow

## Uploading files via SFTP using a Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/sftp
          http://www.mulesoft.org/schema/mule/sftp/3.1/mule-sftp.xsd
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <!-- This placeholder bean will let you import the properties from the sftp.properties file. -->
    <spring:bean id="property-placeholder" class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:sftp.properties"/>
    </spring:bean>

    <flow name="file2sftp">
        <file:inbound-endpoint path="/tmp/outgoing">
            <file:filename-wildcard-filter pattern="*.txt,*.xml"/>
        </file:inbound-endpoint>
        <sftp:outbound-endpoint host="${sftp.host}" port="${sftp.port}" path="/home/test/sftp-files"
user="${sftp.user}" password="${sftp.password}"/>
    </flow>
</mule>
```

## Mule Service

## Uploading files via SFTP using a Service

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:sftp="http://www.mulesoft.org/schema/mule/sftp"
      xmlns:file="http://www.mulesoft.org/schema/mule/file"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/sftp
          http://www.mulesoft.org/schema/mule/file
          http://www.mulesoft.org/schema/mule/file/3.1/mule-file.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

    <!-- This placeholder bean will let you import the properties from the sftp.properties file. -->
    <spring:bean id="property-placeholder" class=
"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <spring:property name="location" value="classpath:sftp.properties"/>
    </spring:bean>
    <model name="file2sftp">
        <service name="file2sftp-service">
            <inbound>
                <file:inbound-endpoint path="/tmp/outgoing">
                    <file:filename-wildcard-filter pattern="*.txt,*.xml"/>
                </file:inbound-endpoint>
            </inbound>
            <outbound>
                <pass-through-router>
                    <sftp:outbound-endpoint host="${sftp.host}" port="${sftp.port}" path=
"/home/test/sftp-files" user="${sftp.user}" password="${sftp.password}"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>

```

A properties file which holds the stfp server login credentials is defined on . Next a file inbound endpoint is declared on which checks the '/tmp/outgoing' directory for new files every one second by default. defines a file filter which will only send files ending with .txt or .xml to the outbound endpoint. Any conforming files found on the inbound endpoint are then written to the '/home/test/sftp-files' remote sftp directory with the same file name it had on the local filesystem .

exchange patterns / features of the transport  
(see [transport matrix](#))

### Configuration Reference

#### **Element Listing**

cache: Unexpected program error: java.lang.NullPointerException

#### **Connector**

SFTP connectivity

#### **Attributes of <connector...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.

dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
maxConnectionPoolSize	integer	no		Required: No Default: disabled If the number of active connections is specified, then a connection pool will be used with active connections up to this number. Use a negative value for no limit. If the value is zero no connection pool will be used.
pollingFrequency	long	no		Required: no Default: 1000 ms The frequency in milliseconds that the read directory should be checked. Note that the read directory is specified by the endpoint of the listening component.
autoDelete	boolean	no		Required: no Default: true Whether to delete the file after successfully reading it.
fileAge	long	no		Required: no Default: disabled Minimum age (in ms) for a file to be processed. This can be useful when consuming large files. It tells Mule to wait for a period of time before consuming the file, allowing the file to be completely written before the file is processed. WARNING: The fileAge attribute will only work properly if the servers where Mule and the sftp-server runs have synchronized time. NOTE: See attribute sizeCheckWaitTime for an alternate method of determining if a incoming file is ready for processing.
sizeCheckWaitTime	long	no		Required: no Default: disabled Wait time (in ms) between size-checks to determine if a file is ready to be processed. Disabled if not set or set to a negative value. This feature can be useful to avoid processing not yet completely written files (e.g. consuming large files). It tells Mule to do two size checks waiting the specified time between the two size calls. If the two size calls return the same value Mule consider the file ready for processing. NOTE: See attribute fileAge for an alternate method of determining if a incoming file is ready for processing.
archiveDir	string	no		Required: no Default: disabled Archives a copy of the file in the specified directory on the file system where mule is running. The archive folder must have been created before Mule is started and the user Mule runs under must have privileges to read and write to the folder.
archiveTempReceivingDir	string	no		Required: no Default: disabled If specified then the file to be archived is received in this folder and then moved to the archiveTempSendingDir while sent further on to the outbound endpoint. This folder is created as a subfolder to the archiveDir. NOTE: Must be specified together with the archiveTempSendingDir and archiveDir attributes.
archiveTempSendingDir	string	no		Required: no Default: disabled If specified then the file to be archived is sent to the outbound endpoint from this folder. This folder is created as a subfolder to the archiveDir. After the file is consumed by the outbound endpoint or the component itself (i.e. when the underlying InputStream is closed) it will be moved to the archive folder. NOTE: Must be specified together with the archiveTempReceivingDir and archiveDir attributes.
outputPattern	string	no		Required: no Default: the message id, e.g. ee241e68-c619-11de-986b-adeb3d6db038 The pattern to use when writing a file to disk. This can use the patterns supported by the filename-parser configured for this connector, by default the <a href="#">Legacy Filename Parser</a> is used. See section <a href="#">Child Elements to File Connector</a> for information on how to override the default parser.

keepFileOnError	boolean	no		Required: no Default: true If true the file on the inbound-endpoint will not be deleted if an error occurs when writing to the outbound-endpoint. NOTE: This assumes that both the inbound and outbound endpoints are using the SFTP-Transport.
duplicateHandling	duplicateHandlingType	no		Required: no Default: throwException Determines what to do if a file already exist on the outbound endpoint with the specified name. throwException: Will throw an exception if a file already exists overwrite: Will overwrite an existing file addSeqNo: Will add a sequence number to the target filename making the filename unique, starting with 1 and incrementing the number until a unique filename is found The default behavior is to throw an exception.
identityFile	string	no		Required: no Default: disabled An identityFile location for a PKI private key.
passphrase	string	no		Required: no Default: disabled The passphrase (password) for the identityFile if required.
tempDirInbound	string	no		Required: No Default: disabled If specified then Mule tries to create the temp-directory in the endpoint folder if it doesn't already exist. Ensure that the user Mule is configured to use to access the sftp server has privileges to create a temp folder if required! For inbound endpoints: A temporary directory on the ftp-server from where the download takes place. The file will be moved (locally on the sftp-server) to the tempDir, to mark that a download is taking place, before the download starts. NOTE: A file in the tempDir of an inbound endpoint is always correct (has only been moved locally on the sftp-server) and can therefore be used to restart a failing file transfer.
tempDirOutbound	string	no		Required: No Default: disabled If specified then Mule tries to create the temp-directory in the endpoint folder if it doesn't already exist. Ensure that the user Mule is configured to use to access the sftp server has privileges to create a temp folder if required! For outbound endpoints: A temporary directory on the sftp-server to first upload the file to. When the file is fully uploaded the file is moved to its final destination. The tempDir will be created as a sub directory to the endpoint. NOTE: A file in the tempDir of an outbound endpoint might not be correct (since the upload takes place to this folder) and can therefore NOT be used to restart a failing file transfer.
useTempFileTimestampSuffix	boolean	no		Required: No Default: disabled Used together with the tempDir - attribute to give the files in the tempDir a guaranteed unique name based on the local time when the file was moved to the tempDir.

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
file:abstract-filenameParser	0..1	

cache: Unexpected program error: java.lang.NullPointerException

#### Inbound endpoint

**Attributes of <inbound-endpoint...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.

connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
path	string	no		A file location.
user	string	no		A username.
password	string	no		A password.
host	string	no		An IP address (eg www.mulesoft.com, localhost, 127.0.0.1).
port	port number	no		A port number.
pollingFrequency	long	no		Required: no Default: 1000 ms The frequency in milliseconds that the read directory should be checked. Note that the read directory is specified by the endpoint of the listening component.
autoDelete	boolean	no		Required: no Default: true Whether to delete the file after successfully reading it.
fileAge	long	no		Required: no Default: disabled Minimum age (in ms) for a file to be processed. This can be useful when consuming large files. It tells Mule to wait for a period of time before consuming the file, allowing the file to be completely written before the file is processed. WARNING: The fileAge attribute will only work properly if the servers where Mule and the sftp-server runs have synchronized time. NOTE: See attribute sizeCheckWaitTime for an alternate method of determining if a incoming file is ready for processing.
sizeCheckWaitTime	long	no		Required: no Default: disabled Wait time (in ms) between size-checks to determine if a file is ready to be processed. Disabled if not set or set to a negative value. This feature can be useful to avoid processing not yet completely written files (e.g. consuming large files). It tells Mule to do two size checks waiting the specified time between the two size calls. If the two size calls return the same value Mule consider the file ready for processing. NOTE: See attribute fileAge for an alternate method of determining if a incoming file is ready for processing.
archiveDir	string	no		Required: no Default: disabled Archives a copy of the file in the specified directory on the file system where mule is running. The archive folder must have been created before Mule is started and the user Mule runs under must have privileges to read and write to the folder.
archiveTempReceivingDir	string	no		Required: no Default: disabled If specified then the file to be archived is received in this folder and then moved to the archiveTempSendingDir while sent further on to the outbound endpoint. This folder is created as a subfolder to the archiveDir. NOTE: Must be specified together with the archiveTempSendingDir and archiveDir attributes.

archiveTempSendingDir	string	no	Required: no Default: disabled If specified then the file to be archived is sent to the outbound endpoint from this folder. This folder is created as a subfolder to the archiveDir. After the file is consumed by the outbound endpoint or the component itself (i.e. when the underlying InputStream is closed) it will be moved to the archive folder. NOTE: Must be specified together with the archiveTempReceivingDir and archiveDir attributes.
identityFile	string	no	Required: no Default: disabled An identityFile location for a PKI private key.
passphrase	string	no	Required: no Default: disabled The passphrase (password) for the identityFile if required.
tempDir	string	no	Required: No Default: disabled If specified then Mule tries to create the temp-directory in the endpoint folder if it doesn't already exist. Ensure that the user Mule is configured to use to access the sftp server has privileges to create a temp folder if required! For inbound endpoints: A temporary directory on the ftp-server from where the download takes place. The file will be moved (locally on the sftp-server) to the tempDir, to mark that a download is taking place, before the download starts. NOTE: A file in the tempDir of an inbound endpoint is always correct (has only been moved locally on the sftp-server) and can therefore be used to restart a failing file transfer. For outbound endpoints: A temporary directory on the sftp-server to first upload the file to. When the file is fully uploaded the file is moved to its final destination. The tempDir will be created as a sub directory to the endpoint. NOTE: A file in the tempDir of an outbound endpoint might not be correct (since the upload takes place to this folder) and can therefore NOT be used to restart a failing file transfer.
useTempFileTimestampSuffix	boolean	no	Required: No Default: disabled Used together with the tempDir - attribute to give the files in the tempDir a guaranteed unique name based on the local time when the file was moved to the tempDir.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	

property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

cache: Unexpected program error: java.lang.NullPointerException

## Outbound endpoint

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.

responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
path	string	no		A file location.
user	string	no		A username.
password	string	no		A password.
host	string	no		An IP address (eg www.mulesoft.com, localhost, 127.0.0.1).

port	port number	no		A port number.
outputPattern	string	no		Required: no Default: the message id, e.g. ee241e68-c619-11de-986b-adeb3d6db038 The pattern to use when writing a file to disk. This can use the patterns supported by the filename-parser configured for this connector, by default the <a href="#">Legacy Filename Parser</a> is used. See section <a href="#">Child Elements to File Connector</a> for information on how to override the default parser.
keepFileOnError	boolean	no		Required: no Default: true If true the file on the inbound-endpoint will not be deleted if an error occurs when writing to the outbound-endpoint. NOTE: This assumes that both the inbound and outbound endpoints are using the SFTP-Transport.
duplicateHandling	duplicateHandlingType	no		Required: no Default: throwException Determines what to do if a file already exist on the outbound endpoint with the specified name. throwException: Will throw an exception if a file already exists overwrite: Will overwrite an existing file addSeqNo: Will add a sequence number to the target filename making the filename unique, starting with 1 and incrementing the number until a unique filename is found The default behavior is to throw an exception.
identityFile	string	no		Required: no Default: disabled An identityFile location for a PKI private key.
passphrase	string	no		Required: no Default: disabled The passphrase (password) for the identityFile if required.
tempDir	string	no		Required: No Default: disabled If specified then Mule tries to create the temp-directory in the endpoint folder if it doesn't already exist. Ensure that the user Mule is configured to use to access the sftp server has privileges to create a temp folder if required! For inbound endpoints: A temporary directory on the ftp-server from where the download takes place. The file will be moved (locally on the sftp-server) to the tempDir, to mark that a download is taking place, before the download starts. NOTE: A file in the tempDir of an inbound endpoint is always correct (has only been moved locally on the sftp-server) and can therefore be used to restart a failing file transfer. For outbound endpoints: A temporary directory on the sftp-server to first upload the file to. When the file is fully uploaded the file is moved to its final destination. The tempDir will be created as a sub directory to the endpoint. NOTE: A file in the tempDir of an outbound endpoint might not be correct (since the upload takes place to this folder) and can therefore NOT be used to restart a failing file transfer.
useTempFileTimestampSuffix	boolean	no		Required: No Default: disabled Used together with the tempDir - attribute to give the files in the tempDir a guaranteed unique name based on the local time when the file was moved to the tempDir.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.

abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

#### – Transformers

There are no additional transformers defined for the SFTP transport.

#### – Filters

There are no additional filters defined for the SFTP transport.

## Schema

You can view the full schema for the SFTP transport [here](#).

## Javadoc API Reference

The Javadoc for this transport can be found [here](#).

## Maven

This transport is part of the following maven module (for version 3.1.1+):

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-module-sftp</artifactId>
</dependency>
```

## Best Practices

Place your SFTP login credentials in a file and reference them in the Mule configuration.

## Notes

To read about the differences between FTP, SFTP, FTPS and SCP, look [here](#)

Your Rating:  Results:  0 rates

## STDIO Transport Reference

### STDIO Transport Reference

The STDIO Transport allows the reading and writing of streaming data to Java's System.out and System.in objects for debugging.

cache: Unexpected program error: java.lang.NullPointerException

## Connector

### Attributes of <connector...>

Name	Type	Required	Default	Description
messageDelayTime	long	no		Delay in milliseconds before printing the prompt to stdout.
outputMessage	string	no		Text printed to stdout when a message is sent.
promptMessage	string	no		Text printed to stdout when waiting for input.
promptMessageCode	string	no		Code used to retrieve prompt message from resource bundle.
outputMessageCode	string	no		Code used to retrieve output message from resource bundle.
resourceBundle	string	no		Resource bundle to provide prompt with promptMessageCode.

To configure the STDIO connector:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/stdio/3.0/mule-stdio.xsd">

    <stdio:connector name="stdioConnector" messageDelayTime="1234" outputMessage="abc" promptMessage="bcd" promptMessageCode="456" resourceBundle="dummy-messages" />
    <model name="model">
        <service name="service">
            <inbound>
                <stdio:inbound-endpoint name="in" system="IN" connector-ref="stdioConnector" />
            </inbound>
            <outbound>
                <multicasting-router>
                    <stdio:outbound-endpoint name="out" system="OUT" connector-ref="stdioConnector" />
                    <stdio:outbound-endpoint name="err" system="ERR" connector-ref="stdioConnector" />
                </multicasting-router>
            </outbound>
        </service>
    </model>
</mule>

```

## Transformers

There are no built-in transformers for the STDIO transport.

## Internationalizing Messages

If you are [internationalizing](#) your application, you can also internationalize the promptMessages and outputMessages for the STDIO connector. (This assumes that you have already created a resource bundle that contains your messages as described on that page.)

To internationalize, you must specify both the resourceBundle parameter and the promptMessageCode and/or outputMessageCode parameters. The resourceBundle parameter will contain the key to the resource bundle itself. The promptMessageCode provides the key to the message in the bundle for the prompt message. In the snippet above, the "dummy-messages" resource bundle means that the prompt message "456" will be expected in the bundle META-INF/services/org/mule/i18n/dummy-messages<langCode>.properties.

Your Rating: 

Results:  0 rates

## TCP Transport Reference

### TCP Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Example Configurations](#) ] [ [Configuration Options](#) ] [ [Configuration Reference](#) ] [ [TCP Transport](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ] [ [Extending this Transport](#) ] [ [Notes](#) ]

### Introduction

The TCP transport allow sending or receiving messages over TCP connections. TCP is a layer over IP and used to implement many other reliable protocols such as HTTP and FTP. However, you may want to use the TCP transport directly if you require a specific protocol for reading the message payload that is not supported by one of these higher level protocols. This is often the case when communicating with legacy or native system applications that don't support web services.

### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	M
-----------	-----	---------	----------	---------	--------------	-----------	---------	------	-------------	---

TCP	JavaDoc		SchemaDoc							one-way, request-response	request-response	or
-----	---------	--	-----------	--	--	--	--	--	--	------------------------------	------------------	----

## Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

## Namespace and Syntax

XML namespace:

```
xmlns:tcp="http://www.mulesoft.org/schema/mule/tcp"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/tcp http://www.mulesoft.org/schema/mule/tcp/3.1/mule-tcp.xsd
```

Connector syntax:

```
<tcp:connector name="tcpConnector" receiveBufferSize="1024" receiveBacklog="50" sendTcpNoDelay="false"
    reuseAddress="true" clientSoTimeout="0" serverSoTimeout="0" socketSoLinger="0"
    keepSendSocketOpen="false" keepAlive="true" dispatcherFactory-ref="dispatcherBean">
    <tcp:PROTOCOL-TYPE/>
</tcp:connector>
<tcp:polling-connector name="tcpConnector" receiveBufferSize="1024" receiveBacklog="50"
    sendTcpNoDelay="false"
    reuseAddress="true" clientSoTimeout="0" serverSoTimeout="0" socketSoLinger="0"
    keepSendSocketOpen="false" keepAlive="true" timeout="10000" pollingFrequency="30000"
    dispatcherFactory-ref="dispatcherBean">
    <tcp:PROTOCOL-TYPE/>
</tcp:polling-connector>
```

PROTOCOL-TYPE defines how messages in Mule are reconstituted from the data packets. The protocol types are:

```

<tcp:direct-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:eof-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:length-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:xml-protocol rethrowExceptionOnRead="true" />

<tcp:xml-eof-protocol rethrowExceptionOnRead="true" />

<tcp:streaming-protocol rethrowExceptionOnRead="true" />

<tcp:safe-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-class-loading-protocol classLoader-ref="classLoaderBean" payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-protocol class="com.mycompany.MyProtocol" rethrowExceptionOnRead="true" />

```

If no protocol is specified, safe-protocol is used.

Your Rating: 

Results:  1 rates

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```
<tcp:inbound-endpoint host="localhost" port="65433"/>
```

2. Non-prefixed URI:

```
<inbound-endpoint address="tcp://localhost:65433"/>
```

See the sections below for more information.

## Considerations

TCP is one of the standard communication protocols used on the Internet, and supports communication both across the internet and within a local area network. The Mule TCP transport uses native Java socket support, adding no communication overhead to the classes in java.net, while allowing many of the advanced features of TCP programming to be specified in the Mule configuration rather than coded in Java.

Use this transport when communicating using low-level TCP connections. To determine when this is appropriate, you can use the following decision tree:

- Communicating with an external service that uses low-level unsecured TCP connections? If so, use the TCP protocol.
- Are you communicating with a flow or service always located in the same Mule application instance? If so, consider use the VM transport.
- Is it important that messages be persisted until they can be processed? If so, consider using a persistent transport like JMS or File.
- Are there advantages to a higher-level protocol built on top of TCP, for instance, the request-response features of HTTP, or the store-and-forward features of Email? If so, use the transport for that protocol instead.
- Is performance the primary concern and it is not important that messages be delivered in the proper order or that the sender is notified if any are lost? If so, use the lighter-weight UDP transport instead.
- Should messages be secured? If so, use the SSL transport.

Your Rating: 

Results:  3 rates

- If you get this far, TCP is a good candidate.

As shown in the examples below, the TCP transport can be used to

- [Create a TCP server](#)
- [Send messages to a TCP server](#)
- [Poll from a TCP server](#)

## Features

The TCP module allows a Mule application both to send and receive messages over TCP connections, and to declaratively customize the following features of TCP (with the standard name for each feature, where applicable):

- The timeout for blocking socket operations. This can be declared separately for client and server operations. (SO\_TIMEOUT)
- How long to keep the socket open to allow pending sends to complete. (SO\_LINGER)
- Whether to send available data immediately rather than buffering it. (TCP\_NODELAY)
- Whether to reuse a socket address immediately (SO\_REUSEADDR)
- Whether to use keep-alive to detect when a remote system is no longer reachable (SO\_KEEPALIVE).
- The size in bytes of the network buffer (SO\_SNDBUF).
- The number of pending connection requests to allow.
- Whether to close a client socket after sending a message.

In addition, since TCP and SSL are stream-oriented and Mule is message-oriented, some application protocol is needed to define where each message begins and ends within the stream. The table below lists the built-in protocols, describing:

- The XML tag used to specify them
- Any XML attributes
- How it defines a message when reading
- Any processing it does while writing a message

XML tag	Options	Read	Write	Notes
<tcp:custom-class-loading-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength, classLoader-ref	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	Like the length protocol, but specifies a classloader used to deserialize objects
<tcp:custom-protocol>	rethrowExceptionOnRead, class, ref	varies	varies	Allows user-written protocols, for instance, to match existing TCP services.
<tcp:direct-protocol>	rethrowExceptionOnRead, payloadOnly	All currently available bytes	none	There are no explicit message boundaries.
<tcp:eof-protocol>	rethrowExceptionOnRead, payloadOnly	All bytes sent until the socket is closed	none	
<tcp:length-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	
<tcp:safe-protocol	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to be preceded by the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Somewhat safer than the length protocol because of the extra check. This is the default if no protocol is specified.
<tcp:streaming-protocol>	rethrowExceptionOnRead	All bytes sent until the socket is closed	none	
<tcp:xml-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration	none	The XML declaration must occur in all messages

<tcp:xml-eof-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration, or whatever remains at EOF	none	The XML declaration must occur in all messages
------------------------	------------------------	--	------	--

Protocol attributes:

name	values	default value	notes
class	The name of the class that implements the custom protocol		See below for an example of writing a custom protocol
classLoader-ref	A reference to a Spring bean that contains the custom classloader		
maxMessageLength	the maximum message length allowed	0 (no maximum )	A message longer than the maximum causes an exception to be thrown.
payloadOnly	true	If true, only the Mule message payload is sent or received. If false, the entire Mule message is sent or received.	Protocols that don't support this attribute always process payloads
ref	A reference to a Spring bean that implements the custom protocol		
rethrowExceptionOnRead	Whether to rethrow exception that occur trying to read from the socket	false	Setting this to "false" avoids logging stack traces when the remote socket is closed unexpectedly

Your Rating: 

Results:  0 rates

TCP endpoints can be used in one of three ways:

- To create a TCP server that accepts incoming connections, declare an inbound tcp endpoint with a tcp:connector. This creates a TCP server socket that will read requests from and optionally write responses to client sockets.
- To poll from a TCP server, declare an inbound tcp endpoint with a tcp:polling-connector. This creates a TCP client socket that will read requests from and optionally write responses to the server socket.
- To write to a TCP server, create an outbound endpoint with a tcp:connector. This creates a TCP client socket that will write requests to and optionally read responses from a server socket.

## Usage

To use TCP endpoints, follow the following steps:

1. Add the MULE TCP namespace to your configuration:
  - Define the tcp prefix using xmlns:tcp="http://www.mulesoft.org/schema/mule/tcp"
  - Define the schema location with [http://www.mulesoft.org/schema/mule/tcp/3.1/mule-tcp.xsd](http://www.mulesoft.org/schema/mule/tcp/http://www.mulesoft.org/schema/mule/tcp/3.1/mule-tcp.xsd)
2. Define one or more connectors for TCP endpoints.

### Create a TCP server

To act as a server that listens for and accepts TCP connections from clients, create a simple TCP connector that inbound endpoints will use:

```
<tcp:connector name="tcpConnector" />
```

### Poll from a TCP server

To act as a client that repeatedly opens connections to a TCP server and reads data from it, create a polling connector that inbound endpoints will use:

```
<tcp:polling-connector name="tcpConnector" />
```

### Send messages to a TCP server

To send messages on a TCP connection, create a simple TCP connector that outbound endpoints will use:

```
<tcp:connector name="tcpConnector" />
```

1. Configure the features of each connector that was created.
  - Begin by choosing the protocol to be used for each message that will be sent or received.
  - For each polling connector, choose how often it will poll and how long it will wait for the connection to complete.
  - Consider the other connector options as well. For instance, if it is important to detect when the remote system becomes unreachable, set `keepAlive` to `true`.
2. Create TCP endpoints.
  - Messages will be received on inbound endpoints.
  - Messages will be sent to outbound endpoints.
  - Both kinds of endpoints are identified by a host name and a port.

By default, TCP endpoints use the request-response exchange pattern, but they can be explicitly configured as one-way. The decision should be straightforward:

Message flow	Connector type	Endpoint type	Exchange Pattern
Mule receives messages from clients but sends no response	tcp:connector	inbound	one-way
Mule receives messages from clients and sends response	tcp:connector	inbound	request-response
Mule reads messages from a server but sends no responses	tcp:polling-connector	inbound	request-response
Mule reads messages from a server and sends responses	tcp:polling-connector	inbound	request-response
Mule sends messages to a server but receives no response	tcp:connector	outbound	one-way
Mule sends messages to a server and receives responses	tcp:connector	outbound	request-response

### Example Configurations

#### Standard TCP connector in flow

```
<tcp:connector name="connector" payloadOnly="false">
    <tcp:eof-protocol />
</tcp:connector>

<flow name="echo">
    <tcp:inbound-endpoint host="localhost" port="4444" >
        <tcp:outbound-endpoint host="remote" port="5555" />
    </tcp:inbound-endpoint>
</flow>
```

### Standard TCP connector in service

```
<tcp:connector name="connector" payloadOnly="false">
    <tcp:safe-protocol />
</tcp:connector>

<model name="echoModel">
    <service name="echo">
        <inbound>
            <tcp:inbound-endpoint host="localhost" port="4444" />
        </inbound>
        <outbound>
            <pass-through-router>
                <tcp:outbound-endpoint host="remote" port="5555" />
            </pass-through-router>
        </outbound>
    </service>
</model>
```

This shows how to create a TCP server in Mule. The connector at defines that a server socket will be created that accepts connections from clients. Complete mule messages are read from the connection (direct protocol) will become the payload of a Mule message (since payload only is false). The endpoint at applies these definitions to create a server at port 4444 on the local host. The messages read from there are then sent to a remote tcp endpoint at .

The flow version uses the eof protocol (), so that every byte sent on the connection is part of the same Mule message. The service version uses the safe protocol (), so that multiple messages can be sent on the TCP connection, with each being preceded by a header than=t specifies its length.

### Polling TCP connector in flow

```
<tcp:polling-connector name="pollingConnector"
    clientSoTimeout="3000" pollingFrequency="1000">
    <tcp:direct-protocol payloadOnly="true" />
</tcp:polling-connector>

<flow name="echo">
    <tcp:inbound-endpoint host="localhost" port="4444" />
    <vm:outbound-endpoint path="out" connector-ref="queue" />
</flow>
```

### Polling TCP connector in service

```
<tcp:polling-connector name="pollingConnector"
    clientSoTimeout="3000" pollingFrequency="1000">
    <tcp:direct-protocol payloadOnly="true" />
</tcp:polling-connector>

<model name="echoModel">
    <service name="echo">
        <inbound>
            <tcp:inbound-endpoint host="localhost" port="4444" />
        </inbound>
        <outbound>
            <pass-through-router>
                <vm:outbound-endpoint path="out" connector-ref="queue" />
            </pass-through-router>
        </outbound>
    </service>
</model>
```

This shows how to create a TCP endpoint that repeatedly reads from an TCP server. The connector at defines that a connection will be attempted every second which will wait up to three seconds to complete. Everything read from the connection (direct protocol) will become the payload of a Mule message (payload only). The endpoint at applies these definitions to port 4444 on the local host. The messages read from there are then sent to a VM endpoint at .

## Configuration Options

TCP Connector attributes

Name	Description	Default
clientSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a TCP server socket	system default
keepAlive	Whether to send keep-alive messages to detect when the remote socket becomes unreachable	false
keepSendSocketOpen	Whether to keep the the socket open after sending a message	false
receiveBacklog	The number of connection attempts that can be outstanding	system default
receiveBufferSize	This is the size of the network buffer used to receive messages. In most cases, there is no need to set this, since the system default will be sufficient	system default
reuseAddress	Whether to reuse a socket address that's currently in a TIMED_WAIT state. This avoids triggering the error that the socket is unavailable	true
sendBufferSize	The size of the network send buffer	system default
sendTcpNoDelay	Whether to send data as soon as its available, rather than waiting for more to arrive to economize on the number of packets sent	false
socketSoLinger	How long (in milliseconds) to wait for the socket to close so that all pending data is flushed	system default
serverSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a client socket	system default

Your Rating: 

Results:  1 rates

Polling TCP Connector-specific attributes

Name	Description	Default
pollingFrequency	How often (in milliseconds) to connect to the TCP sever	1000 milliseconds
timeout	How long (in milliseconds) to wait for the connection to complete	system default

## Configuration Reference

### *Element Listing*

cache: Unexpected program error: java.lang.NullPointerException

## TCP Transport

The TCP transport enables events to be sent and received over TCP sockets.

### **Connector**

Connects Mule to a TCP socket to send or receive data via the network.

#### Attributes of <connector...>

Name	Type	Required	Default	Description
sendBufferSize	integer	no		The size of the buffer (in bytes) used when sending data, set on the socket itself.
receiveBufferSize	integer	no		The size of the buffer (in bytes) used when receiving data, set on the socket itself.
receiveBacklog	integer	no		The maximum queue length for incoming connections.
sendTcpNoDelay	boolean	no		If set, transmitted data is not collected together for greater efficiency but sent immediately.

reuseAddress	boolean	no		If set (the default), SO_REUSEADDRESS is set on server sockets before binding. This helps reduce "address already in use" errors when a socket is re-used.
clientSoTimeout	integer	no		This sets the SO_TIMEOUT value when the socket is used as a client. Reading from the socket will block for up to this long (in milliseconds) before the read fails. A value of 0 (the default) causes the read to wait indefinitely (if no data arrives).
serverSoTimeout	integer	no		This sets the SO_TIMEOUT value when the socket is used as a server. Reading from the socket will block for up to this long (in milliseconds) before the read fails. A value of 0 (the default) causes the read to wait indefinitely (if no data arrives).
socketSoLinger	integer	no		This sets the SO_LINGER value. This is related to how long (in milliseconds) the socket will take to close so that any remaining data is transmitted correctly.
keepSendSocketOpen	boolean	no		If set, the socket is not closed after sending a message. This attribute only applies when sending data over a socket (Client).
keepAlive	boolean	no		Enables SO_KEEPALIVE behavior on open sockets. This automatically checks socket connections that are open but unused for long periods and closes them if the connection becomes unavailable. This is a property on the socket itself and is used by a server socket to control whether connections to the server are kept alive before they are recycled.
socketMaxWait	integer	no		Sets the maximum amount of time (in milliseconds) the socket pool should block waiting for a socket before throwing an exception. When less than or equal to 0 it may block indefinitely (the default).
dispatcherFactory-ref	string	no		Allows to define a custom message dispatcher factory

#### Child Elements of <connector...>

Name	Cardinality	Description
abstract-protocol	0..1	The class name for the protocol handler. This controls how the raw data stream is converted into messages. By default, messages are constructed as data is received, with no correction for multiple packets or fragmentation. Typically, change this value, or use a transport that includes a protocol like HTTP.

#### Polling connector

Connects Mule to a TCP socket to send or receive data via the network.

#### Attributes of <polling-connector...>

Name	Type	Required	Default	Description
sendBufferSize	integer	no		The size of the buffer (in bytes) used when sending data, set on the socket itself.
receiveBufferSize	integer	no		The size of the buffer (in bytes) used when receiving data, set on the socket itself.
receiveBacklog	integer	no		The maximum queue length for incoming connections.
sendTcpNoDelay	boolean	no		If set, transmitted data is not collected together for greater efficiency but sent immediately.
reuseAddress	boolean	no		If set (the default), SO_REUSEADDRESS is set on server sockets before binding. This helps reduce "address already in use" errors when a socket is re-used.
clientSoTimeout	integer	no		This sets the SO_TIMEOUT value when the socket is used as a client. Reading from the socket will block for up to this long (in milliseconds) before the read fails. A value of 0 (the default) causes the read to wait indefinitely (if no data arrives).
serverSoTimeout	integer	no		This sets the SO_TIMEOUT value when the socket is used as a server. Reading from the socket will block for up to this long (in milliseconds) before the read fails. A value of 0 (the default) causes the read to wait indefinitely (if no data arrives).
socketSoLinger	integer	no		This sets the SO_LINGER value. This is related to how long (in milliseconds) the socket will take to close so that any remaining data is transmitted correctly.
keepSendSocketOpen	boolean	no		If set, the socket is not closed after sending a message. This attribute only applies when sending data over a socket (Client).

keepAlive	boolean	no		Enables SO_KEEPALIVE behavior on open sockets. This automatically checks socket connections that are open but unused for long periods and closes them if the connection becomes unavailable. This is a property on the socket itself and is used by a server socket to control whether connections to the server are kept alive before they are recycled.
socketMaxWait	integer	no		Sets the maximum amount of time (in milliseconds) the socket pool should block waiting for a socket before throwing an exception. When less than or equal to 0 it may block indefinitely (the default).
dispatcherFactory-ref	string	no		Allows to define a custom message dispatcher factory
timeout	long	no		The timeout to wait in milliseconds for data to come from the server
pollingFrequency	long	no		The time in milliseconds to wait between each request to the TCP server.

#### Child Elements of <polling-connector...>

Name	Cardinality	Description
abstract-protocol	0..1	The class name for the protocol handler. This controls how the raw data stream is converted into messages. By default, messages are constructed as data is received, with no correction for multiple packets or fragmentation. Typically, change this value, or use a transport that includes a protocol like HTTP.

#### Abstract protocol

##### Attributes of <abstract-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails

#### Child Elements of <abstract-protocol...>

Name	Cardinality	Description

#### Streaming protocol

TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol. However, this is not an issue with streaming, so the streaming-protocol element is an alias for the "direct" (null) protocol.

##### Attributes of <streaming-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails

#### Child Elements of <streaming-protocol...>

Name	Cardinality	Description

#### Xml protocol

TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol. The xml-protocol element configures the XML protocol, which uses XML syntax to isolate messages from the stream of bytes received, so it will only work with well-formed XML.

##### Attributes of <xml-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails

#### Child Elements of <xml-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Xml eof protocol***

Similar to xml-protocol, the xml-eof-protocol element configures the XML protocol, but it will also use socket closure to terminate a message (even if the XML is not well-formed).

#### Attributes of <xml-eof-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails

#### Child Elements of <xml-eof-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Eof protocol***

TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol. The eof-protocol element configures a protocol that simply accumulates all data until the socket closes and places it in a single message.

#### Attributes of <eof-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
payloadOnly	boolean	yes		Sends only the payload, not the entire Mule message object or its properties. This defaults to true when the protocol is not specified explicitly (when the safe protocol is used).

#### Child Elements of <eof-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Direct protocol***

TCP does not guarantee that data written to a socket is transmitted in a single packet. Using the direct-protocol element to configure the "null" protocol does not change the normal TCP behavior, so message fragmentation may occur. For example, a single sent message may be received in several pieces, each as a separate received message. Typically, it is not a good choice for messaging within Mule, but it may be necessary to interface with external TCP-based protocols.

#### Attributes of <direct-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
payloadOnly	boolean	yes		Sends only the payload, not the entire Mule message object or its properties. This defaults to true when the protocol is not specified explicitly (when the safe protocol is used).

#### Child Elements of <direct-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Safe protocol***

Similar to length-protocol, safe-protocol also includes a prefix. Verification of the prefix allows mis-matched protocols to be detected and avoids interpreting "random" data as a message length (which may give out-of-memory errors). This is the default protocol in Mule 2.x.

#### Attributes of <safe-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
payloadOnly	boolean	yes		Sends only the payload, not the entire Mule message object or its properties. This defaults to true when the protocol is not specified explicitly (when the safe protocol is used).
maxMessageLength	integer	no		An optional maximum length for the number of bytes in a single message. Messages larger than this will trigger an error in the receiver, but it give an assurance that no out-of-memory error will occur.

Child Elements of <safe-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Custom class loading protocol***

A length protocol that uses a specific class loader to load objects from streams

Attributes of <custom-class-loading-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
payloadOnly	boolean	yes		Sends only the payload, not the entire Mule message object or its properties. This defaults to true when the protocol is not specified explicitly (when the safe protocol is used).
maxMessageLength	integer	no		An optional maximum length for the number of bytes in a single message. Messages larger than this will trigger an error in the receiver, but it give an assurance that no out-of-memory error will occur.
classLoader-ref	string	no		Allows Spring beans to be defined for class loading

Child Elements of <custom-class-loading-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Length protocol***

The length-protocol element configures the length protocol, which precedes each message with the number of bytes sent so that an entire message can be constructed on the received.

Attributes of <length-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
payloadOnly	boolean	yes		Sends only the payload, not the entire Mule message object or its properties. This defaults to true when the protocol is not specified explicitly (when the safe protocol is used).
maxMessageLength	integer	no		An optional maximum length for the number of bytes in a single message. Messages larger than this will trigger an error in the receiver, but it give an assurance that no out-of-memory error will occur.

Child Elements of <length-protocol...>

Name	Cardinality	Description
------	-------------	-------------

### ***Custom protocol***

The custom-protocol element allows you to configure your own protocol implementation.

#### Attributes of <custom-protocol...>

Name	Type	Required	Default	Description
rethrowExceptionOnRead	boolean	no		Rethrow the exception if read fails
class	class name	no		A class that implements the TcpProtocol interface.
ref	name (no spaces)	no		Reference to a spring bean that implements the TcpProtocol interface.

#### Child Elements of <custom-protocol...>

Name	Cardinality	Description
------	-------------	-------------

#### *Inbound endpoint*

The inbound-endpoint element configures the endpoint on which the messages are received.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		The host of the TCP socket.
port	port number	no		The port of the TCP socket.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

#### *Outbound endpoint*

The outbound-endpoint element configures the endpoint where the messages are sent.

#### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		The host of the TCP socket.
port	port number	no		The port of the TCP socket.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

#### *Endpoint*

The endpoint element configures a global TCP endpoint definition.

#### Attributes of <endpoint...>

Name	Type	Required	Default	Description
host	string	no		The host of the TCP socket.
port	port number	no		The port of the TCP socket.

#### Child Elements of <endpoint...>

Name	Cardinality	Description
------	-------------	-------------

#### **Schema**

The schema for the TCP module appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/tcp"

Targeting Schemas (1):

[mule-tcp.xsd](#)

Targeting Components:

15 global elements, 11 complexTypes, 1 attribute group

Schema Summary	
<a href="#">mule-tcp.xsd</a>	<p>The TCP transport enables events to be sent and received over TCP sockets.</p> <p>Target Namespace: <a href="http://www.mulesoft.org/schema/mule/tcp">http://www.mulesoft.org/schema/mule/tcp</a></p> <p>Defined Components: 15 global elements, 11 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form: Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location: <a href="http://www.mulesoft.org/schema/mule/tcp/3.1/mule-tcp.xsd">http://www.mulesoft.org/schema/mule/tcp/3.1/mule-tcp.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (3): <a href="#">mule-schemadoc.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p> <p>Imported by Schemas (3): <a href="#">mule-http.xsd</a>, <a href="#">mule-ssl.xsd</a>, <a href="#">mule-tls.xsd</a></p>
All Element Summary	
<a href="#">abstract-protocol</a>	Type: <a href="#">abstractProtocolType</a> Content: empty, 1 attribute Subst.Gr:may be substituted with 9 elements Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: at 10 locations
<a href="#">connector</a>	Connects Mule to a TCP socket to send or receive data via the network. Type: <a href="#">tcpConnectorType</a> Content: complex, 17 attributes, attr. wildcard, 6 elements Subst.Gr:may substitute for element <a href="#">mule:abstract-connector</a> Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: never
<a href="#">custom-class-loading-protocol</a>	A length protocol that uses a specific class loader to load objects from streams Type: <a href="#">customClassLoadingProtocolType</a> Content: empty, 4 attributes Subst.Gr:may substitute for element <a href="#">abstract-protocol</a> Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: never
<a href="#">custom-protocol</a>	The custom-protocol element allows you to configure your own protocol implementation. Type: <a href="#">customProtocolType</a> Content: empty, 3 attributes Subst.Gr:may substitute for element <a href="#">abstract-protocol</a> Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: never
<a href="#">direct-protocol</a>	TCP does not guarantee that data written to a socket is transmitted in a single packet. Type: <a href="#">byteOrMessageProtocolType</a> Content: empty, 2 attributes Subst.Gr:may substitute for element <a href="#">abstract-protocol</a> Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: never
<a href="#">endpoint</a>	The endpoint element configures a global TCP endpoint definition. Type: <a href="#">globalEndpointType</a>

	<p>Content: complex, 13 attributes, attr. wildcard, 12 elements  Subst.Gr:may substitute for element <code>mule:abstract-global-endpoint</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>eof-protocol</code>	<p>TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol.  Type: <code>byteOrMessageProtocolType</code>  Content: empty, 2 attributes  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>inbound-endpoint</code>	<p>The inbound-endpoint element configures the endpoint on which the messages are received.  Type: <code>inboundEndpointType</code>  Content: complex, 13 attributes, attr. wildcard, 12 elements  Subst.Gr:may substitute for element <code>mule:abstract-inbound-endpoint</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>length-protocol</code>	<p>The length-protocol element configures the length protocol, which precedes each message with the number of bytes sent so that an entire message can be constructed on the received.  Type: <code>lengthProtocolType</code>  Content: empty, 3 attributes  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>outbound-endpoint</code>	<p>The outbound-endpoint element configures the endpoint where the messages are sent.  Type: <code>outboundEndpointType</code>  Content: complex, 13 attributes, attr. wildcard, 12 elements  Subst.Gr:may substitute for element <code>mule:abstract-outbound-endpoint</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>polling-connector</code>	<p>Connects Mule to a TCP socket to send or receive data via the network.  Type: <code>pollingTcpConnectorType</code>  Content: complex, 19 attributes, attr. wildcard, 6 elements  Subst.Gr:may substitute for element <code>mule:abstract-connector</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>safe-protocol</code>	<p>Similar to length-protocol, safe-protocol also includes a prefix.  Type: <code>lengthProtocolType</code>  Content: empty, 3 attributes  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>streaming-protocol</code>	<p>TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol.  Type: <code>abstractProtocolType</code>  Content: empty, 1 attribute  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>xml-eof-protocol</code>	<p>Similar to xml-protocol, the xml-eof-protocol element configures the XML protocol, but it will also use socket closure to terminate a message (even if the XML is not well-formed).  Type: <code>abstractProtocolType</code>  Content: empty, 1 attribute  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<code>xml-protocol</code>	<p>TCP does not guarantee that data written to a socket is transmitted in a single packet, so if you want to transmit entire Mule messages reliably, you must specify an additional protocol.  Type: <code>abstractProtocolType</code>  Content: empty, 1 attribute  Subst.Gr:may substitute for element <code>abstract-protocol</code>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Used: never</p>
<b>Complex Type Summary</b>	
<code>abstractProtocolType</code>	<p>Content: empty, 1 <a href="#">attribute</a>  Defined: globally in <a href="#">mule-tcp.xsd</a>; see <a href="#">XML source</a>  Includes:definition of 1 attribute</p>

	Used: at 6 locations
byteOrMessageProtocolType	Content: empty, 2 attributes Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 3 locations
customClassLoadingProtocolType	Content: empty, 4 attributes Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 1 location
customProtocolType	Content: empty, 3 attributes Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 attributes Used: at 1 location
globalEndpointType	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
inboundEndpointType	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
lengthProtocolType	Content: empty, 3 attributes Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 3 locations
noProtocolTcpConnectorType	Content: complex, 16 attributes, attr. wildcard, 5 elements Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 10 attributes Used: at 2 locations
outboundEndpointType	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Used: at 1 location
pollingTcpConnectorType	Content: complex, 19 attributes, attr. wildcard, 6 elements Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 attributes Used: at 1 location
tcpConnectorType	Content: complex, 17 attributes, attr. wildcard, 6 elements Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 1 attribute, 1 element Used: at 4 locations
Attribute Group Summary	
addressAttributes	Content: 2 attributes Defined: globally in <a href="#">mule-tcp.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 attributes Used: at 3 locations

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this module can be found here: [TCP](#)

## Maven

The TCP Module can be included with the following dependency:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-tcp</artifactId>
  <version>3.1.0</version>
</dependency>
```

## Extending this Transport

When using TCP to communicate with an external program, it may be necessary to write a custom Mule protocol. The first step is to get a complete description of how the external program delimits messages within the TCP stream. The next is to implement the protocol as a Java class.

- All protocols must implement the interface `org.mule.transport.tcp.TcpProtocol`, which contains three methods:
  - `Object read(InputStream is)` reads a message from the TCP socket
  - `write(OutputStream os, Object data)` writes a message to the TCP socket
  - `ResponseOutputStream createResponse(Socket socket)` creates a stream to which a response can be written.
- Protocols which process byte-streams rather than serialized Mule messages can inherit much useful infrastructure by subclassing `org.mule.transport.tcp.protocols.AbstractByteProtocol`. This class
  - implements `createResponse`
  - handles converting messages to byte arrays, allowing subclasses to implement only the simpler method `writeByteArray(OutputStream os, byte[] data)`
  - provides methods `safeRead(InputStream is, byte[] buffer)` and `safeRead(InputStream is, byte[] buffer, int size)` that handle the situation where data is not currently available when doing non-blocking reads from the TCP socket

Suppose we want to communicate with a server that has a simple protocol: all messages are terminated by `>>>`. The protocol class would look like this:

```
package org.mule.transport.tcp.integration;

import org.mule.transport.tcp.protocols.AbstractByteProtocol;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class CustomByteProtocol extends AbstractByteProtocol
{

    /**
     * Create a CustomByteProtocol object.
     */
    public CustomByteProtocol()
    {
        super(false); // This protocol does not support streaming.
    }

    /**
     * Write the message's bytes to the socket,
     * then terminate each message with '>>>'.
     */
    @Override
    protected void writeByteArray(OutputStream os, byte[] data) throws IOException
    {
        super.writeByteArray(os, data);
        os.write('>');
        os.write('>');
        os.write('>');
    }

    /**
     * Read bytes until we see '>>>', which ends the message
     */
    public Object read(InputStream is) throws IOException
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        int count = 0;
        byte read[] = new byte[1];

        while (true)
        {
            // if no bytes are currently available, safeRead()
            if (count < 1)
                count = safeRead(is, read, 0, 1);
            if (read[0] == '>' && read[1] == '>' && read[2] == '>')
                break;
            baos.write(read[0]);
        }
        return baos.toString("UTF-8");
    }
}
```

```
// will wait until some arrive
if (safeRead(is, read) < 0)
{
    // We've reached EOF.  Return null, so that our
// caller will know there are no
// remaining messages
return null;
}
byte b = read[0];
if (b == '>')
{
    count++;
    if (count == 3)
    {
        return baos.toByteArray();
    }
}
else
{
    for (int i = 0; i < count; i++)
    {
        baos.write('>');
    }
    count = 0;
    baos.write(b);
}
}
```

```
}
```

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## Notes

TCP and SSL are very low-level transports, so the usual tools for debugging their use, for instance, logging messages as they arrive, might not be sufficient. Once messages are being sent and received successfully, things are largely working. It may be necessary to use software (or hardware) than can track messages at the packet level, particularly when a custom protocol is being used. Alternatively, you can debug by temporarily using the direct protocol on all inbound endpoints, since it will accept (and you can then log) bytes as they are received.

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## UDP Transport Reference

### UDP Module Reference

[ Introduction ] [ Transport Info ] [ Namespace and Syntax ] [ Considerations ] [ Features ] [ Usage ] [ Example Configurations ] [ Configuration Options ] [ Configuration Reference ] [ UDP Transport ] [ Schema ] [ Javadoc API Reference ] [ Maven ] [ Notes ]

#### Introduction

Universal Datagram Protocol, or UDP, is a stateless protocol for sending or receiving large numbers of messages (datagrams) quickly. The UDP transport in Mule allow sending messages to or receiving messages with Mule using UDP sockets.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
UDP	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	org.mule.transport.udp

#### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name and artifact name for this transport in Maven

## Namespace and Syntax

XML namespace:

```
xmlns:udp="http://www.mulesoft.org/schema/mule/udp"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/udp http://www.mulesoft.org/schema/mule/udp/3.1/mule-udp.xsd
```

Connector syntax:

```
<udp:connector name="udpConnector" receiveBufferSize="1024" sendBufferSize="1024"  
timeout="0" keepSendSocketOpen="false" broadcast="false"/>
```

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```
<udp:inbound-endpoint host="localhost" port="65433"/>
```

2. Non-prefixed URI:

```
<inbound-endpoint address="udp://localhost:65433"/>
```

See the sections below for more information.

## Considerations

UDP is one of the standard communication protocols used on the Internet, and supports communication both across the internet and within a local area network. The Mule UDP module uses native Java socket support, adding no communication overhead to the classes in java.net, while allowing many of the advanced features of UDP programming to be specified in the Mule configuration rather than coded in Java.

Use this transport when communicating using low-level UDP datagrams. UDP is designed to maximize speed and scale over reliability, ordering or data integrity. UDP datagrams are not guaranteed to arrive with any particular speed, or at all, and they may arrive in a different order than they are sent in. If any of these guarantees are important to your application, use a different transport, such as TCP.

Note, UDP provides no error checking, so you may want to perform additional validation or error handling in your application, if it is important.

As shown in the examples below, the UDP transport can be used to

- Send messages to a UDP socket
- Read messages from a UDP socket

## Features

The UDP module allows a Mule application both to send and receive UDP datagrams, and to declaratively customize the following features of UDP (with the standard name for each feature, where applicable):

- The timeout for sending or receiving messages (SO\_TIMEOUT).
- Whether to allow sending broadcast messages (SO\_BROADCAST).
- Whether to close a socket after sending a message.
- The maximum size of messages that can be received.

UDP endpoints can be used in one of two ways:

- To receive a UDP datagram, create an inbound UDP endpoint.
- To send a UDP datagram, create an outbound UDP endpoint.

## Usage

To use UDP endpoints

1. Add the MULE UDP namespace to your configuration:
  - Define the udp prefix using xmlns:udp="http://www.mulesoft.org/schema/mule/udp"
  - Define the schema location with <http://www.mulesoft.org/schema/mule/udp>  
<http://www.mulesoft.org/schema/mule/udp/3.1/mule-udp.xsd>
2. Define one or more connectors for UDP endpoints.

- Create a UDP connector:

```
<udp:connector name="udpConnector"/>
```

### 3. Create UDP endpoints.

- Datagrams will be received on inbound endpoints. The bytes in the datagram will become the message payload.
- Datagrams will be sent to outbound endpoints. The bytes in the message payload will become the datagram.
- Both kinds of endpoints are identified by a host name and a port.

NOTE: UDP endpoints are always one-way.

## Example Configurations

### Copy datagrams from one port to another

```
<udp:connector name="connector"/>

<flow name="copy">
    <udp:inbound-endpoint host="localhost" port="4444" exchange-pattern="one-way"/>
    <udp:outbound-endpoint host="remote" port="5555" exchange-pattern="one-way" />
</flow>
```

The connector at uses all default properties. The inbound endpoint at receives datagrams and copies them to the outbound endpoint at .

## Configuration Options

UDP Connector attributes

Name	Description	Default
broadcast	set this to true to allow sending to broadcast ports	false
keepSendSocketOpen	Whether to keep the the socket open after sending a message	false
receiveBufferSize	This is the size of the largest (in bytes) datagram that can be received.	16 Kbytes
sendBufferSize	The size of the network send buffer	16 Kbytes
timeout	the timeout used for both sending and receiving	system default

## Configuration Reference

### Element Listing

cache: Unexpected program error: java.lang.NullPointerException

## UDP Transport

The UDP transport enables events to be sent and received as Datagram packets.

### Connector

#### Attributes of <connector...>

Name	Type	Required	Default	Description
receiveBufferSize	integer	no		The size of the receiving buffer for the socket.
timeout	long	no		The amount of time after which a Send or Receive call will time out.
receiveTimeout	long	no		Deprecated – use timeout.
sendTimeout	long	no		Deprecated – use timeout.

sendBufferSize	integer	no		The size of the sending buffer for the socket.
broadcast	boolean	no		Whether to enable the socket to send broadcast data.
keepSendSocketOpen	boolean	no		Whether to keep the Sending socket open.

Child Elements of <connector...>

Name	Cardinality	Description
------	-------------	-------------

### *Inbound endpoint*

Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

### *Outbound endpoint*

Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

### *Endpoint*

Attributes of <endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

Child Elements of <endpoint...>

Name	Cardinality	Description
------	-------------	-------------

### **Schema**

The schema for the UDP module appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/udp"

Targeting Schemas (1):

mule-udp.xsd

Targeting Components:

4 global elements, 4 complexTypes, 1 attribute group

Schema Summary	
mule-udp.xsd	<p>The UDP transport enables events to be sent and received as Datagram packets.</p> <p>Target Namespace:</p> <p style="padding-left: 2em;"><a href="http://www.mulesoft.org/schema/mule/udp">http://www.mulesoft.org/schema/mule/udp</a></p> <p>Defined Components:</p> <p style="padding-left: 2em;">4 global elements, 4 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form:</p> <p style="padding-left: 2em;">Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:</p> <p style="padding-left: 2em;"><a href="http://www.mulesoft.org/schema/mule/udp/3.1/mule-udp.xsd">http://www.mulesoft.org/schema/mule/udp/3.1/mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (3):</p> <p style="padding-left: 2em;"><a href="#">mule-schemadoc.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p> <p>Imported by Schema:</p> <p style="padding-left: 2em;"><a href="#">mule-multicast.xsd</a></p>
All Element Summary	
connector	<p>Type: <a href="#">udpConnectorType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 5 elements</p> <p>Subst.Gr:may substitute for element mule:abstract-connector</p> <p>Defined: globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
endpoint	<p>Type: <a href="#">globalEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element mule:abstract-global-endpoint</p> <p>Defined: globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
inbound-endpoint	<p>Type: <a href="#">inboundEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element mule:abstract-inbound-endpoint</p> <p>Defined: globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
outbound-endpoint	<p>Type: <a href="#">outboundEndpointType</a></p> <p>Content: complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Subst.Gr:may substitute for element mule:abstract-outbound-endpoint</p> <p>Defined: globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
Complex Type Summary	
globalEndpointType	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: at 1 <a href="#">location</a></p>
inboundEndpointType	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: at 1 <a href="#">location</a></p>
outboundEndpointType	<p>Content:complex, 13 attributes, attr. wildcard, 12 elements</p> <p>Defined:globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Used: at 1 <a href="#">location</a></p>
udpConnectorType	<p>Content: complex, 13 attributes, attr. wildcard, 5 elements</p> <p>Defined:globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Includes:definitions of 7 attributes</p> <p>Used: at 2 <a href="#">locations</a></p>
Attribute Group Summary	
addressAttributes	<p>Content: 2 <a href="#">attributes</a></p> <p>Defined: globally in <a href="#">mule-udp.xsd</a>; see <a href="#">XML source</a></p> <p>Includes:definitions of 2 <a href="#">attributes</a></p>

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this module can be found here: [UDP](#)

## Maven

The UDP Module can be included with the following dependency:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-udp</artifactId>
  <version>3.1.0</version>
</dependency>
```

## Notes

Before Mule 3.1.1, there were two different attributes for setting timeout on UDP connectors, `sendTimeout` and `receiveTimeout`. It was necessary to set them to the same value.

Your Rating:  Results:  1 rates

## VM Transport Reference

### In Memory (VM) Transport Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Transformers](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Example Configurations](#) ] [ [Configuration Reference](#) ] [ [VM Transport](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ] [ [Best Practices](#) ]

#### Introduction

The In Memory transport can be used for intra-JVM communication between Mule Flows and/or Services. This transport by default uses in-memory queues but can optionally be configured to use persistent queues.

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Art
VM	JavaDoc SchemaDoc	✓	✓	✓	✓ (XA)	✓	✗	one-way, request-response	one-way	org.mule.tr:

#### Legend

▶ Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see [Streaming](#).

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP  
**Maven Artifact** - The group name a artifact name for this transport in [Maven](#)

## Namespace and Syntax

XML namespace:

```
xmlns:vm "http://www.mulesoft.org/schema/mule/vm"
```

XML Schema Location:

```
http://www.mulesoft.org/schema/mule/vm
http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd
```

Connector syntax:

```
<vm:connector name="persistent">
  <vm:queue-profile persistent="true" maxOutstandingMessages="500" />
</vm:connector>
```

VM with in-memory queues (or no queue, for request-response endpoints):

```
<vm:connector name="memory" />
```

Endpoint Syntax:

1. Prefixed endpoint:

```
<vm:outbound-endpoint path="out" />
```

1. Non-prefixed URI:

```
<outbound-endpoint address="vm://out" />
```

## Transformers

There are no specific transformers for the VM transport.

## Considerations

The VM transport has often been used to implement complex integrations made up of multiple services. Improvements in Mule 3 obviate the need for VM in many cases.

Because of the introduction of MessageProcessor in most Mule elements, you can accomplish more complex integrations without tying together services. You can use Flow References to directly reference one flow from another without a transport in the middle.

VM is still useful in certain situations. Suppose, for instance, that most of the parts of your solution are local, but some will need to be decoupled for testing, or because some will need to be made remote.



### WARNING

Each application in a Mule instance has its own, unique set of VM endpoints. Thus the VM transport cannot be used to communicate between different Mule applications.

## Features

The in memory (VM) transport has two modes of operation: One for use with *request-response* and another for use with *one-way* endpoints. (See MEPs for more information)

### **request-response:**

When using *request-response* endpoints, messages are delivered directly from an outbound vm endpoint to the inbound vm endpoint that is listening on the same path. This delivery is blocking and occurs in the same thread. If there is no inbound *request-response* vm endpoint in the same Mule application listening on this path, then dispatching of the message from the outbound endpoint will fail.

### **one-way:**

When using *one-way* endpoints, messages are delivered to the corresponding inbound endpoint via a queue. This delivery is non-blocking. If there is no inbound *one-way* endpoint in the same Mule application listening on this path, then, although dispatching of the message will succeed, the message will remain in the queue. By default, this queue is in memory, but it is also possible to configure a persistent queue that will use the file system as its persistence mechanism.



#### **NOTE**

You cannot send a message from a request-response endpoint to a one-way endpoint or the other way around. This will either cause the send to fail or the message will stay in the queue and never arrive at the expected destination.

## Usage

To use VM endpoints

1. Add the MULE VM namespace to your configuration:
  - Define the vm prefix using xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
  - Define the schema location with <http://www.mulesoft.org/schema/mule/vm>  
<http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd>
2. Optionally, define one or more connectors for VM endpoints.
  - Create a VM connector:

```
<vm:connector name="vmConnector" />
```

If none is created, all VM endpoints will use a default connector.

3. Create VM endpoints.
  - Messages will be received on inbound endpoints.
  - Messages will be sent to outbound endpoints.
  - Both kinds of endpoints are identified by a path name.

### **Namespace Declaration**

To use the VM transport, you must declare the VM namespace in the header of the Mule configuration file. For example:

#### **VM Transport Namespace Declaration**

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/vm
          http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">
```

### **Connector Configuration**

The configuration of the VM connector is optional. Configuring a connector allows you to configure a Queue Profile which will allow persistence to be used. You can also set the maximum queue size.

### **Endpoints**

Endpoints are configured as with all transports.

The VM transport specific endpoints are configured using the vm namespace and use a *path* attribute. For example:

```
<vm:outbound-endpoint path="out" exchange-pattern="one-way" />
```

If you need to invoke a VM endpoint from Mule Client, use an Endpoint URI. The format of an endpoint uri for VM is as follows:

```
vm://path
```

### Using Transactions

one-way VM queues can take part in distributed XA Transactions. To make a VM endpoint transactional, use a configuration like the following:

```
<flow>
  <vm:inbound-endpoint address="vm://dispatchInQueue">
    <vm:transaction action="BEGIN_OR_JOIN" />
  </vm:inbound-endpoint>
</flow>
```

Using XA requires that you add a transaction manager to your configuration. For more information, see [Transaction Management](#).

### Example Configurations

#### Example usage of VM endpoints

```
<vm:connector name="vmConnector">

<vm:connector name="persistentVmConnector" queueTimeout="1000">
  <queue-profile maxOutstandingMessages="100" persistent="true" />
</vm:connector>

<flow>
  <vm:inbound-endpoint path="in" exchange-pattern="request-response" connector-ref="vmConnector" />
  <component class="org.mule.CompoenntClass"/>
  <vm:inbound-endpoint path="in" connector-ref="persistentVmConnector" />
</flow>
```

The first connector uses default connector configuration.

The second connector configures a queue profile and queueTimeout.

The flow uses two VM endpoints, the inbound endpoint uses a *request-response* exchange pattern. The outbound endpoint use a *one-way* endpoint as well as an alternative connector with persistence configured.

### Configuration Reference

#### Element Listing

cache: Unexpected program error: java.lang.NullPointerException

### VM Transport

The VM transport is used for intra-VM communication between components managed by Mule. The transport provides options for configuring VM transient or persistent queues.

#### Connector

##### Attributes of <connector...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
createMultipleTransactedReceivers	boolean	no		Whether to create multiple concurrent receivers for this connector. This property is used by transports that support transactions, specifically receivers that extend the TransactedPollingMessageReceiver, and provides better throughput.
numberOfConcurrentTransactedReceivers	integer	no		If createMultipleTransactedReceivers is set to true, the number of concurrent receivers that will be launched.
queueTimeout	positiveInteger	no		The timeout setting for the queue used for asynchronous endpoints

#### Child Elements of <connector...>

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
spring:property	0..*	

receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.
queueProfile	0..1	DEPRECATED. USE "<queue-profile>" instead.
queue-profile	0..1	Configures the properties of this connector's queue (see <a href="#">Configuring Queues</a> ).

### Inbound endpoint

The endpoint on which this connector receives messages from the transport.

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
path	string	no		The queue path, such as dispatchInQueue to create the address vm://dispatchInQueue.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-xa-transaction	0..1	A placeholder for XA transaction elements. XA transactions allow a series of operations to be grouped together spanning different transports, such as JMS and JDBC.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

### ***Outbound endpoint***

The endpoint to which this connector sends messages.

#### **Attributes of <outbound-endpoint...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
exchange-pattern	one-way/request-response	no		
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.

transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
path	string	no		The queue path, such as dispatchInQueue to create the address vm://dispatchInQueue.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-xa-transaction	0..1	A placeholder for XA transaction elements. XA transactions allow a series of operations to be grouped together spanning different transports, such as JMS and JDBC.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

#### Endpoint

An endpoint "template" that can be used to construct an inbound or outbound endpoint elsewhere in the configuration by referencing the endpoint name.

#### Attributes of <endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).

address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
path	string	no		The queue path, such as dispatchInQueue to create the address vm://dispatchInQueue.

#### Child Elements of <endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-xa-transaction	0..1	A placeholder for XA transaction elements. XA transactions allow a series of operations to be grouped together spanning different transports, such as JMS and JDBC.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

#### Transaction

The transaction element configures a transaction. Transactions allow a series of operations to be grouped together so that they can be rolled back if a failure occurs. For more information, see [Transaction Management](#).

## Child Elements of <transaction...>

Name	Cardinality	Description
------	-------------	-------------

## Schema

The schema for the VM module appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/vm"

Targeting Schemas (1):

mule-vm.xsd

Targeting Components:

5 global elements, 2 local elements, 4 complexTypes, 1 attribute group

Schema Summary	
mule-vm.xsd	<p>The VM transport is used for intra-VM communication between components managed by Mule.          Target Namespace:  <a href="http://www.mulesoft.org/schema/mule/vm">http://www.mulesoft.org/schema/mule/vm</a></p> <p>Defined Components:          5 global elements, 2 local elements, 4 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form:          Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:  <a href="http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd">http://www.mulesoft.org/schema/mule/vm/3.1/mule-vm.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (3):  <a href="#">mule-schemadoc.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p>

## All Element Summary

connector	<p>Type: <a href="#">vmConnectorType</a>          Content: complex, 7 attributes, attr. wildcard, 7 elements          Subst.Gr:may substitute for element <a href="#">mule:abstract-connector</a>          Defined: globally in <a href="#">mule-vm.xsd</a>; see <a href="#">XML source</a>          Used: never</p>
endpoint	<p>An endpoint "template" that can be used to construct an inbound or outbound endpoint elsewhere in the configuration by referencing the endpoint name.          Type: <a href="#">globalEndpointType</a>          Content: complex, 12 attributes, attr. wildcard, 11 elements          Subst.Gr:may substitute for element <a href="#">mule:abstract-global-endpoint</a>          Defined: globally in <a href="#">mule-vm.xsd</a>; see <a href="#">XML source</a>          Used: never</p>
inbound-endpoint	<p>The endpoint on which this connector receives messages from the transport.          Type: <a href="#">inboundEndpointType</a>          Content: complex, 12 attributes, attr. wildcard, 11 elements          Subst.Gr:may substitute for element <a href="#">mule:abstract-inbound-endpoint</a>          Defined: globally in <a href="#">mule-vm.xsd</a>; see <a href="#">XML source</a>          Used: never</p>
outbound-endpoint	<p>The endpoint to which this connector sends messages.          Type: <a href="#">outboundEndpointType</a>          Content: complex, 12 attributes, attr. wildcard, 11 elements          Subst.Gr:may substitute for element <a href="#">mule:abstract-outbound-endpoint</a>          Defined: globally in <a href="#">mule-vm.xsd</a>; see <a href="#">XML source</a>          Used: never</p>
queue-profile	<p>Configures the properties of this connector's queue (see [Configuring Queues]).          Type: <a href="#">mule:queueProfileType</a>          Content:empty, 2 attributes          Defined:locally within <a href="#">vmConnectorType</a> complexType in <a href="#">mule-vm.xsd</a>; see <a href="#">XML source</a></p>

queueProfile	DEPRECATED. Type: <a href="#">mule:queueProfileType</a> Content:empty, 2 attributes Defined:locally within <a href="#">vmConnectorType</a> complexType in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a>
transaction	The transaction element configures a transaction. Type: <a href="#">mule:baseTransactionType</a> Content: empty, 2 attributes Subst.Gr:may substitute for element <a href="#">mule:abstract-transaction</a> Defined: globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Used: never
Complex Type Summary	
globalEndpointType	Content:complex, 12 attributes, attr. wildcard, 11 elements Defined:globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Used: at 1 <a href="#">location</a>
inboundEndpointType	Content:complex, 12 attributes, attr. wildcard, 11 elements Defined:globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Used: at 1 <a href="#">location</a>
outboundEndpointType	Content:complex, 12 attributes, attr. wildcard, 11 elements Defined:globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Used: at 1 <a href="#">location</a>
vmConnectorType	Content: complex, 7 attributes, attr. wildcard, 7 elements Defined: globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 1 attribute, 2 elements Used: at 1 <a href="#">location</a>
Attribute Group Summary	
addressAttributes	Content: 1 attribute Defined: globally in <a href="#">mule-vm.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 3 locations

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this module can be found here: [VM](#)

## Maven

The In Memory Transport can be included with the following dependency:

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-vm</artifactId>
</dependency>
```

## Best Practices

Be certain that inbound request-response endpoints are paired with outbound request-response endpoints and inbound one-way endpoints are paired with outbound one-way endpoints.

Your Rating:  Results:  0 rates

## WSDL Connectors

### WSDL Connectors

[ [Generic WSDL Endpoints](#) ] [ [Specifying an Alternate WSDL Location](#) ] [ [Example of the CXF WSDL Endpoint](#) ]

The [Axis](#) and [CXF](#) transports provide WSDL connectors that can be used for invoking remote web services by obtaining the service WSDL. Mule

creates a dynamic proxy for the service and then invokes it.

The Javadoc for the Axis WSDL connector is [here](#), and for the CXF WSDL connector it's [here](#).

## Generic WSDL Endpoints

A WSDL endpoint enables you to easily invoke web services without generating a client. At startup, it reads the WSDL to determine how to invoke the remote web service during runtime. When a message is sent through the WSDL endpoint, it is able to construct a SOAP message using the message payload and its knowledge of the expected payload format from the WSDL.

You must provide the full URL to the WSDL of the service to invoke, and you must supply a `method` parameter that tells Mule which operation to invoke on the service:

```
wsdl:http://www.webservicex.net/stockquote.asmx?WSDL&method=GetQuote
```

The WSDL URL is prepended with the `wsdl:` prefix. Mule checks your class path to see if there is a WSDL provider that it can use to create a client proxy from the WSDL. Mule supports both Axis and CXF as WSDL providers. If you want to use a specific one, you can specify it on the URL as follows:

```
wsdl-cxf:http://www.webservicex.net/stockquote.asmx?WSDL&method=GetQuote
```

or

```
wsdl-axis:http://www.webservicex.net/stockquote.asmx?WSDL&method=GetQuote
```

In general, you should use the CXF WSDL endpoint. The one limitation of the CXF WSDL provider is that it does not allow you to use non-Java primitives (objects that are not a String, int, double, and so on). Sometimes the Axis WSDL generation will not work (incorrect namespaces are used), so you can experiment with each one to see which works best.

Note that there are no specific transformers to set on WSDL endpoints.

## Specifying an Alternate WSDL Location

By default, the WSDL provider will look for your WSDL by taking the endpoint address and appending `?wsdl` to it. With the CXF transport, you have the option of specifying a location for the WSDL that is different from that specified with the `?wsdl` parameter. This may be useful in cases where the WSDL isn't available the normal way, either because the SOAP engine doesn't provide it or the provider does not want to expose the WSDL publicly.

In these cases, you can specify the `wsdlLocation` property of the CXF endpoint as follows:

```
<endpoint  
    address="wsdl-cxf:http://localhost:8080/book/services/BookService?method=getBooks">  
    <properties>  
        <property name="wsdlLocation" value="file:///c:/BookService.wsdl"/>  
    </properties>  
</endpoint>
```

In this case, the WSDL CXF endpoint works as it normally does, except that it reads the WSDL from the local drive.

## Example of the CXF WSDL Endpoint

This example demonstrates how to take multiple arguments from the console, invoke a web service, and print the output to the screen. It uses the [Currency Convertor](#) web service on [www.webservicex.net](#).

This service requires two arguments: the "from" currency code and the "to" currency code. When the CXF dispatcher prepares arguments for the invocation of the service, it expects to find a message payload of `Object[]` - that is, an Object array. In the case of the Currency Convertor, this should be an array of two Objects - the "from" currency and the "to" currency.

There are several ways to construct this object array, but the easiest way is to use the custom transformer `StringToObjectArrayTransformer`, which translates a delimited String into an Object array. In the example below, you simply type in a String in the format of `<fromCurrency>, <toCurrency>`.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core/2.2"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-2.5.xsd
          http://www.mulesoft.org/schema/mule/core/2.2
          http://www.mulesoft.org/schema/mule/core/2.2/mule.xsd">

    <model name="sample">
        <service name="inputService">
            <inbound>
                <inbound-endpoint address="stdio://System.in?promptMessage=Enter from and to currency
symbols, separated by a comma:">
                    synchronous="true">
                <transformers>
                    <!-- Convert the input to an object array -->
                    <custom-transformer class="org.mule.transformer.simple.StringToObjectArray">
                        <spring:property name="delimiter" value=","/>
                    </custom-transformer>
                </transformers>
            </inbound-endpoint>
            </inbound>
            <outbound>
                <chaining-router>
                    <outbound-endpoint address=
"wsdl-cxf:http://www.webservicex.net/CurrencyConvertor.asmx?WSDL&method=ConversionRate"
                        synchronous="true"/>
                    <outbound-endpoint address="stdio://System.out"/>
                </chaining-router>
            </outbound>
        </service>
    </model>
</mule>

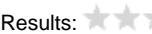
```

For example, type "EUR,USD" to get the conversion rate for Euros to US Dollars, and you'll get something like this:

```

Enter from and to currency symbols, separated by a comma:
EUR,USD
1.3606

```

Your Rating:  Results:  0 rates

## XMPP Transport Reference

### XMPP Transport Reference

[ Introduction ] [ Transport Info ] [ Namespace and Syntax ] [ Filters ] [ Considerations ] [ Features ] [ Usage ] [ Example Configurations ] [ Configuration Options ] [ Configuration Reference ] [ XMPP Transport ] [ Schema ] [ Javadoc API Reference ] [ Maven ] [ Extending this Module or Transport ] [ Best Practices ] [ Notes ]

#### Introduction

The XMPP transport enables receiving and sending Mule messages over the Extensible Messaging and Presence Protocol (aka Jabber).

#### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Art
XMPP	JavaDoc SchemaDoc	✓	✓	✓	✗	✗	✗	one-way, request-response	one-way	org.mule.tr...

### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name and artifact name for this transport in Maven

## Namespace and Syntax

### XML Namespace

```
http://www.mulesoft.org/schema/mule/xmpp
```

### XML schema location

```
http://www.mulesoft.org/schema/mule/xmpp/3.1/mule-xmpp.xsd
```

### Endpoint syntax

#### 1. raw URLs

```
xmpp://MESSAGE/theUser@jabber.server.com
xmpp://CHAT/theUser@jabber.server.com
xmpp://GROUPCHAT/theGroupChat
```

#### 2. transport specific endpoints in XML

```
<xmpp:inbound-endpoint type="MESSAGE" from="theUser@jabber.server.com"/>
<xmpp:outbound-endpoint type="MESSAGE" recipient="theUser@jabber.server.com"/>

<xmpp:inbound-endpoint type="CHAT" from="theUser@jabber.server.com"/>
<xmpp:outbound-endpoint type="CHAT" recipient="theUser@jabber.server.com"/>

<xmpp:inbound-endpoint type="GROUPCHAT" from="theGroupChat" nickname="muley"/>
<xmpp:outbound-endpoint type="GROUPCHAT" recipient="theGroupChat" nickname="muley"/>
```

### Connector and endpoint syntax

```
<xmpp:connector name="xmpp" host="localhost" user="theUser" password="secret"/>
```

## Filters

There are several filters in the [org.mule.transport.xmpp.filters](#) package that filter XMPP messages. `AbstractXmppFilter` is an abstract filter adapter that allows `Smack` filters to be configured as Mule filters. The following filter classes extend the abstract filter:

- `XmppAndFilter`
- `XmppFromContainsFilter`
- `XmppMessageTypeFilter`
- `XmppNotFilter`
- `XmppOrFilter`
- `XmppPacketIDFilter`
- `XmppPacketTypeFilter`
- `XmppThreadFilter`
- `XmppToContainsFilter`

## Considerations

### Features

- Send and receive messages to/from an individual Jabber user
- Send and receive messages to/from a Jabber one on one chat
- Send and receive messages to/from a Jabber multi user chat

### Usage

The configuration of a connector is mandatory as it represents the Jabber server that will be connected. Thus, all information to establish the connection such as the host to contact and the user credentials to use are configured here.

Configuration options on an endpoint include the type (whether it's a chat, a message or a groupchat) and the Jabber ID the endpoint is talking to.

The XMPP transport does not support transactions as the XMPP protocol is not transactional.

### Example Configurations

#### Receive messages in a flow

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xmpp="http://www.mulesoft.org/schema/mule/xmpp"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/xmpp
          http://www.mulesoft.org/schema/mule/xmpp/3.1/mule-xmpp.xsd">

    <xmpp:connector name="xmppConnector" host="localhost" user="theUser" password="secret"/>

    <flow name="receiveMessage">
        <xmpp:inbound-endpoint type="MESSAGE" from="other@jabber.server.com"/>
        <component class="com.mycompany.mule.JabberMessageHandler"/>
    </flow>
</mule>
```

### Receive messages in a service

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xmpp="http://www.mulesoft.org/schema/mule/xmpp"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/xmpp
          http://www.mulesoft.org/schema/mule/xmpp/3.1/mule-xmpp.xsd">

    <xmpp:connector name="xmppConnector" host="localhost" user="theUser" password="secret"/>

    <model>
        <service name="receiveFromJabber">
            <inbound>
                <xmpp:inbound-endpoint type="MESSAGE" from="otherUser@jabber.server.com"/>
            </inbound>
            <component class="com.mycompany.mule.JabberMessageHandler" />
        </service>
    </mule>
```

### Simple Jabber chat client

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:xmpp="http://www.mulesoft.org/schema/mule/xmpp"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/xmpp
          http://www.mulesoft.org/schema/mule/xmpp/3.1/mule-xmpp.xsd
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/stdio/3.1/mule-stdio.xsd">

    <xmpp:connector name="xmppConnector" host="localhost" user="theUser" password="secret"/>

    <flow name="stdio2xmpp">
        <stdio:inbound-endpoint system="IN"/>
        <xmpp:outbound-endpoint type="CHAT" recipient="otheruser@localhost"/>
    </flow>

    <flow name="xmpp2stdio">
        <xmpp:inbound-endpoint type="CHAT" from="otheruser@localhost"/>
        <xmpp:xmpp-to-object-transformer/>
        <stdio:outbound-endpoint system="OUT"/>
    </flow>
</mule>
```

## Configuration Options

### Configuration Reference

#### *Element Listing*

cache: Unexpected program error: java.lang.NullPointerException

### XMPP Transport

The XMPP transport connects Mule to an XMPP (Jabber) server.

## **Connector**

Connect Mule to an XMPP (Jabber) server to send or receive data via the network.

### **Attributes of <connector...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
name	name (no spaces)	yes		Identifies the connector so that other elements can reference it.
dynamicNotification	boolean	no	false	Enables dynamic notifications for notifications fired by this connector. This allows listeners to be registered dynamically at runtime via the MuleContext, and the configured notification can be changed. This overrides the default value defined in the 'configuration' element.
validateConnections	boolean	no	true	Causes Mule to validate connections before use. Note that this is only a configuration hint, transport implementations may or may not make an extra effort to validate the connection. Default is true.
dispatcherPoolFactory-ref	string	no		Allows Spring beans to be defined as a dispatcher pool factory
host	string	no		Host name or IP address of the Jabber server.
port	port number	no		The port number to connect on. Default port is 5222.
serviceName	string	no		The service name to use when connecting the Jabber server.
user	string	no		The username used for authentication.
password	string	no		The password for the user being authenticated.
resource	string	no		The resource portion of the address, such as user@host/resource or domain/resource.
createAccount	boolean	no		If true, an attempt is made to create an account using the user and password while connecting. Default is false.

### **Child Elements of <connector...>**

Name	Cardinality	Description
spring:property	0..*	
receiver-threading-profile	0..1	The threading profile to use when a connector receives messages.
dispatcher-threading-profile	0..1	The threading profile to use when a connector dispatches messages.
abstract-reconnection-strategy	0..1	Reconnection strategy that defines how Mule should handle a connection failure. A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
service-overrides	0..1	Service overrides allow the connector to be further configured/customized by allowing parts of the transport implementation to be overridden, for example, the message receiver or dispatcher implementation, or the message adaptor that is used.

## **Inbound endpoint**

The endpoint on which this connector receives messages from the xmpp connection.

### **Attributes of <inbound-endpoint...>**

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.

connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
recipient	string	no		The Jabber ID of the intended recipient of the messages, such as ross@myco.com. For GROUPCHAT type endpoints, this is the address of the chat to join.
from	string	no		The user who sent the message. Ignored in GROUPCHAT type endpoints.
type	MESSAGE/CHAT/GROUPCHAT	no	CHAT	The type of the Jabber message to send: MESSAGE, CHAT or GROUPCHAT.
subject	string	no		The subject of the message (applies to type=MESSAGE endpoints only).
thread	string	no		The thread to which the message belongs.
nickname	string	no		The user's nickname in a groupchat.

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	

property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

## Outbound endpoint

The endpoint to which this connector sends messages.

### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is not need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).

address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
name	name (no spaces)	no		Identifies the endpoint in the registry. There is no need to set the 'name' attribute on inbound or outbound endpoints, only on global endpoints.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
contentType	string	no		The mime type, e.g. text/plain or application/json

exchange-pattern	one-way/request-response	no		
recipient	string	no		The Jabber ID of the intended recipient of the messages, such as ross@myco.com. For GROUPCHAT type endpoints, this is the address of the chat to join.
from	string	no		The user who sent the message. Ignored in GROUPCHAT type endpoints.
type	MESSAGE/CHAT/GROUPCHAT	no	CHAT	The type of the Jabber message to send: MESSAGE, CHAT or GROUPCHAT.
subject	string	no		The subject of the message (applies to type=MESSAGE endpoints only).
thread	string	no		The thread to which the message belongs.
nickname	string	no		The user's nickname in a groupchat.

#### Child Elements of <outbound-endpoint...>

Name	Cardinality	Description
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.

abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

## Endpoint

An endpoint "template" that can be used to construct an inbound or outbound endpoint elsewhere in the configuration by referencing the endpoint name.

### Attributes of <endpoint...>

Name	Type	Required	Default	Description
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.

disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
name	name (no spaces)	yes		Identifies the endpoint so that other elements can reference it. This name can also be referenced in the MuleClient.
ref	string	no		A reference to a global endpoint, which is used as a template to construct this endpoint. A template fixes the address (protocol, path, host, etc.), and may specify initial values for various properties, but further properties can be defined locally (as long as they do not change the address in any way).
address	string	no		The generic address for this endpoint. If this attribute is used, the protocol must be specified as part of the URI. Alternatively, most transports provide their own attributes for specifying the address (path, host, etc.). Note that the address attribute cannot be combined with 'ref' or with the transport-provided alternative attributes.
responseTimeout	integer	no		The timeout for a response if making a synchronous endpoint call
encoding	string	no		String encoding used for messages.
connector-ref	string	no		The name of the connector associated with this endpoint. This must be specified if more than one connector is defined for this transport.
transformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the message before it is delivered to the component.
responseTransformer-refs	list of names	no		A list of the transformers that will be applied (in order) to the synchronous response before it is returned via the transport.
disableTransportTransformer	boolean	no		Don't use the default inbound/outbound/response transformer which corresponds to this endpoint's transport, if any.
mimeType	string	no		The mime type, e.g. text/plain or application/json
exchange-pattern	one-way/request-response	no		
recipient	string	no		The Jabber ID of the intended recipient of the messages, such as ross@myco.com. For GROUPCHAT type endpoints, this is the address of the chat to join.
from	string	no		The user who sent the message. Ignored in GROUPCHAT type endpoints.
type	MESSAGE/CHAT/GROUPCHAT	no	CHAT	The type of the Jabber message to send: MESSAGE, CHAT or GROUPCHAT.
subject	string	no		The subject of the message (applies to type=MESSAGE endpoints only).
thread	string	no		The thread to which the message belongs.
nickname	string	no		The user's nickname in a groupchat.

#### Child Elements of <endpoint...>

Name	Cardinality	Description
------	-------------	-------------

response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.
response	0..1	
abstract-transaction	0..1	A placeholder for transaction elements. Transactions allow a series of operations to be grouped together.
abstract-reconnection-strategy	0..1	A placeholder for a reconnection strategy element. Reconnection strategies define how Mule should attempt to handle a connection failure.
abstract-multi-transaction	0..1	A placeholder for multi-transaction elements. Multi-transactions allow a series of operations to be grouped together spanning different transports, e.g. JMS and JDBC, but without the overhead of XA. The trade-off is that XA reliability guarantees aren't available, and services must be ready to handle duplicates. This is very similar to a 1.5 PC concept. EE-only feature.
abstract-transformer	0..1	A placeholder for transformer elements. Transformers convert message payloads.
abstract-filter	0..1	A placeholder for filter elements, which control which messages are handled.
abstract-security-filter	0..1	A placeholder for security filter elements, which control access to the system.
abstract-intercepting-message-processor	0..1	A placeholder for intercepting router elements.
abstract-observer-message-processor	0..1	A placeholder for message processors that observe the message but do not mutate it used for example for logging.
processor	0..1	A reference to a message processor defined elsewhere.
custom-processor	0..1	
property	0..*	Sets a Mule property. This is a name/value pair that can be set on components, services, etc., and which provide a generic way of configuring the system. Typically, you shouldn't need to use a generic property like this, since almost all functionality is exposed via dedicated elements. However, it can be useful in configuring obscure or overlooked options and in configuring transports from the generic endpoint elements.
properties	0..1	A map of Mule properties.

#### Xmpp to object transformer

The xmpp-to-object-transformer element configures a transformer that converts an XMPP message into an object by extracting the message payload.

#### Child Elements of <xmpp-to-object-transformer...>

Name	Cardinality	Description
------	-------------	-------------

#### **Object to xmpp transformer**

The object-to-xmpp-transformer element configures a transformer that converts an object into an XMPP message.

#### Child Elements of <object-to-xmpp-transformer...>

Name	Cardinality	Description
------	-------------	-------------

#### Schema

Complete schema reference documentation.

#### Javadoc API Reference

The Javadoc for this module can be found [here](#).

#### Maven

This transport is part of the following maven module (for version 3.1.0):

```
<dependency>
  <groupId>org.mule.transports</groupId>
  <artifactId>mule-transport-xmpp</artifactId>
  <version>3.1.0</version>
</dependency>
```

#### Extending this Module or Transport

#### Best Practices

Put your login credentials in a properties file, not hard-coded in the configuration. This also allows you to use different settings between development, test and production systems.

#### Notes

The current implementation of the transport is limited to one-way endpoints only. The logic that supports request-response endpoints is currently not implemented.

Your Rating:  Results:  1 rates

## TCP, SSL, and TLS Transports Reference

Description:

#### TCP, SSL, and TLS Transports Reference

Your Rating:  Results:  1 rates

There are two Mule transports that provide access to TCP connections:

The [TCP Transport](#), which uses the basic TCP transport.

The [SSL and TLS Transports](#), which use TCP with socket-level security. (SSL and TLS are actually two names for the same transport.)

Other than the type of socket used, these transports all behave quite similarly.

## Choosing a Transport

- Communicating with an external service that uses low-level unsecured TCP connections? If so, use the TCP protocol.
- Are you communicating with a flow or service always located in the same Mule application instance? If so, consider use the VM transport.
- Is it important that messages be persisted until they can be processed? If so, consider using a persistent transport like JMS or File.
- Are there advantages to a higher-level protocol built on top of TCP, for instance, the request-response features of HTTP, or the store-and-forward features of Email? If so, use the transport for that protocol instead.
- Is performance the primary concern and it is not important that messages be delivered in the proper order or that the sender is notified if any are lost? If so, use the lighter-weight UDP transport instead.
- Should messages be secured? If so, use the SSL transport.

Your Rating: 

Results:  3 rates

## Custom TCP Protocol

When using TCP to communicate with an external program, it may be necessary to write a custom Mule protocol. The first step is to get a complete description of how the external program delimits messages within the TCP stream. The next is to implement the protocol as a Java class.

- All protocols must implement the interface `org.mule.transport.tcp.TcpProtocol`, which contains three methods:
  - `Object read(InputStream is)` reads a message from the TCP socket
  - `write(OutputStream os, Object data)` writes a message to the TCP socket
  - `ResponseOutputStream createResponse(Socket socket)` creates a stream to which a response can be written.
- Protocols which process byte-streams rather than serialized Mule messages can inherit much useful infrastructure by subclassing `org.mule.transport.tcp.protocols.AbstractByteProtocol`. This class
  - implements `createResponse`
  - handles converting messages to byte arrays, allowing subclasses to implement only the simpler method `writeByteArray(OutputStream os, byte[] data)`
  - provides methods `safeRead(InputStream is, byte[] buffer)` and `safeRead(InputStream is, byte[] buffer, int size)` that handle the situation where data is not currently available when doing non-blocking reads from the TCP socket

Suppose we want to communicate with a server that has a simple protocol: all messages are terminated by `>>>`. The protocol class would look like this:

```
package org.mule.transport.tcp.integration;

import org.mule.transport.tcp.protocols.AbstractByteProtocol;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class CustomByteProtocol extends AbstractByteProtocol
{

    /**
     * Create a CustomByteProtocol object.
     */
    public CustomByteProtocol()
    {
        super(false); // This protocol does not support streaming.
    }

    /**
     * Write the message's bytes to the socket,
     * then terminate each message with '>>>'.
     */
    @Override
    protected void writeByteArray(OutputStream os, byte[] data) throws IOException
```

```

{
    super.writeByteArray(os, data);
    os.write('>');
    os.write('>');
    os.write('>');
}

/**
 * Read bytes until we see '>>>', which ends the message
 */
public Object read(InputStream is) throws IOException
{
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    int count = 0;
    byte read[] = new byte[1];

    while (true)
    {
        // if no bytes are currently available, safeRead()
        // will wait until some arrive
        if (safeRead(is, read) < 0)
        {
            // We've reached EOF.  Return null, so that our
            // caller will know there are no
            // remaining messages
            return null;
        }
        byte b = read[0];
        if (b == '>')
        {
            count++;
            if (count == 3)
            {
                return baos.toByteArray();
            }
        }
        else
        {
            for (int i = 0; i < count; i++)
            {
                baos.write('>');
            }
            count = 0;
            baos.write(b);
        }
    }
}

```

```
}
```

Your Rating:  Results:  0 rates

## Protocol Tables

In addition, since TCP and SSL are stream-oriented and Mule is message-oriented, some application protocol is needed to define where each message begins and ends within the stream. The table below lists the built-in protocols, describing:

- The XML tag used to specify them
- Any XML attributes
- How it defines a message when reading
- Any processing it does while writing a message

XML tag	Options	Read	Write	Notes
<tcp:custom-class-loading-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength, classLoader-ref	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	Like the length protocol, but specifies a classloader used to deserialize objects
<tcp:custom-protocol>	rethrowExceptionOnRead, class, ref	varies	varies	Allows user-written protocols, for instance, to match existing TCP services.
<tcp:direct-protocol>	rethrowExceptionOnRead, payloadOnly	All currently available bytes	none	There are no explicit message boundaries.
<tcp:eof-protocol>	rethrowExceptionOnRead, payloadOnly	All bytes sent until the socket is closed	none	
<tcp:length-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	
<tcp:safe-protocol	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to begin with the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Somewhat safer than the length protocol because of the extra check. This is the default if no protocol is specified.
<tcp:streaming-protocol	rethrowExceptionOnRead	All bytes sent until the socket is closed	none	
<tcp:xml-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration	none	The XML declaration must occur in all messages
<tcp:xml-eof-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration, or whatever remains at EOF	none	The XML declaration must occur in all messages

Protocol attributes:

<b>name</b>	<b>values</b>	<b>default value</b>	<b>notes</b>
class	The name of the class that implements the custom protocol		See below for an example of writing a custom protocol
classLoader-ref	A reference to a Spring bean that contains the custom classloader		
maxMessageLength	the maximum message length allowed	0 (no maximum )	A message longer than the maximum causes an exception to be thrown.
payloadOnly	true	If true, only the Mule message payload is sent or received. If false, the entire Mule message is sent or received.	Protocols that don't support this attribute always process payloads
ref	A reference to a Spring bean that implements the custom protocol		
rethrowExceptionOnRead	Whether to rethrow exception that occur trying to read from the socket	false	Setting this to "false" avoids logging stack traces when the remote socket is closed unexpectedly

Your Rating: 

Results:  0 rates

## Protocol Types

PROTOCOL-TYPE defines how messages in Mule are reconstituted from the data packets. The protocol types are:

```
<tcp:direct-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:eof-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:length-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:xml-protocol rethrowExceptionOnRead="true" />

<tcp:xml-eof-protocol rethrowExceptionOnRead="true" />

<tcp:streaming-protocol rethrowExceptionOnRead="true" />

<tcp:safe-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-class-loading-protocol classLoader-ref="classLoaderBean" payloadOnly="true"
maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-protocol class="com.mycompany.MyProtocol" rethrowExceptionOnRead="true" />
```

If no protocol is specified, safe-protocol is used.

Your Rating: 

Results:  1 rates

## SSL and TLS Transports Reference

### SSL and TLS Transports Reference

[ [Introduction](#) ] [ [Transport Info](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Example Configurations](#) ] [ [Configuration Options](#) ] [ [Configuration Reference](#) ] [ [SSL Transport](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ] [ [Extending this Transport](#) ] [ [Notes](#) ]

#### Introduction

SSL and TLS are alternative names for the same transport. (For simplicity, this page will refer to it only as SSL, but everything here applies to TLS as well.) This transport allows sending or receiving messages over SSL connections. SSL is a layer over IP and used to implement many other reliable protocols such as HTTPS and SMTPS. However, you may want to use the SSL transport directly if you require a specific protocol for

reading the message payload that is not supported by one of these higher level protocols. This is often the case when communicating with legacy or native system applications that don't support web services.

### Transport Info

cache: Unexpected program error: java.lang.NullPointerException

Transport	Doc	Inbound	Outbound	Request	Transactions	Streaming	Retries	MEPs	Default MEP	Maven Artifact
SSL	JavaDoc SchemaDoc	✓	✓	✓	✗	✓	✗	one-way, request-response	request-response	org.mule.transport.ssl.SSLTransport

#### Legend

► Click here to expand...

**Transport** - The name/protocol of the transport

**Docs** - Links to the JavaDoc and SchemaDoc for the transport

**Inbound** - Whether the transport can receive inbound events and can be used for an inbound endpoint

**Outbound** - Whether the transport can produce outbound events and be used with an outbound endpoint

**Request** - Whether this endpoint can be queried directly with a request call (via MuleClinet or the EventContext)

**Transactions** - Whether transactions are supported by the transport. Transports that support transactions can be configured in either local or distributed two-phase commit (XA) transaction.

**Streaming** - Whether this transport can process messages that come in on an input stream. This allows for very efficient processing of large data. For more information, see Streaming.

**Retry** - Whether this transport supports retry policies. Note that all transports can be configured with Retry policies, but only the ones marked here are officially supported by MuleSoft

**MEPs** - Message Exchange Patterns supported by this transport

**Default MEP** - The default MEP for endpoints that use this transport that do not explicitly configure a MEP

**Maven Artifact** - The group name and artifact name for this transport in Maven

### Namespace and Syntax

XML namespace:

```
xmlns:ssl="http://www.mulesoft.org/schema/mule/ssl"
xmlns:tls="http://www.mulesoft.org/schema/mule/tls"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/ssl http://www.mulesoft.org/schema/mule/ssl/3.1/mule-ssl.xsd
http://www.mulesoft.org/schema/mule/tls http://www.mulesoft.org/schema/mule/tls/3.1/mule-tls.xsd
```

Connector syntax:

```
<ssl:connector name="tcpConnector" receiveBufferSize="1024" receiveBacklog="50" sendTcpNoDelay="false"
    reuseAddress="true" clientSoTimeout="0" serverSoTimeout="0" socketSoLinger="0"
    keepSendSocketOpen="false" keepAlive="true" dispatcherFactory-ref="dispatcherBean">
    <tcp:PROTOCOL-TYPE/>
    <ssl:client path="clientKeystore" storePassword="swordfish" class="JKS"/>
    <ssl:key-store path="keystore" class="JKS" keyPassword="swordfish" storePassword="sturgeon"
    algorithm="SHA"/>
    <ssl:server class="JKS" algorithm="SHA" explicitOnly="false" requireClientAuthentication = "false"
/>
    <ssl:protocol-handler type="com.mycompany.protocols"/>
</ssl:connector>
```

PROTOCOL-TYPE defines how messages in Mule are reconstituted from the data packets. The protocol types are:

```

<tcp:direct-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:eof-protocol payloadOnly="true" rethrowExceptionOnRead="true" />

<tcp:length-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:xml-protocol rethrowExceptionOnRead="true" />

<tcp:xml-eof-protocol rethrowExceptionOnRead="true" />

<tcp:streaming-protocol rethrowExceptionOnRead="true" />

<tcp:safe-protocol payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-class-loading-protocol classLoader-ref="classLoaderBean" payloadOnly="true" maxMessageLength="1024" rethrowExceptionOnRead="true" />

<tcp:custom-protocol class="com.mycompany.MyProtocol" rethrowExceptionOnRead="true" />

```

If no protocol is specified, safe-protocol is used.

Your Rating: 

Results:  1 rates

Endpoint syntax:

You can define your endpoints 2 different ways:

1. Prefixed endpoint:

```
<ssl:inbound-endpoint host="localhost" port="65433"/>
```

2. Non-prefixed URI:

```
<inbound-endpoint address="ssl://localhost:65433"/>
```

See the sections below for more information.

## Considerations

SSL is one of the standard communication protocols used on the Internet, and supports secure communication both across the internet and within a local area network. The Mule SSL transport uses native Java socket support, adding no communication overhead to the classes in java.net, while allowing many of the advanced features of SSL programming to be specified in the Mule configuration rather than coded in Java.

Use this transport when communicating using low-level SSL connections. To determine when this is appropriate, you can use the following decision tree:

- Communicating with an external service that uses low-level unsecured TCP connections? If so, use the TCP protocol.
- Are you communicating with a flow or service always located in the same Mule application instance? If so, consider use the VM transport.
- Is it important that messages be persisted until they can be processed? If so, consider using a persistent transport like JMS or File.
- Are there advantages to a higher-level protocol built on top of TCP, for instance, the request-response features of HTTP, or the store-and-forward features of Email? If so, use the transport for that protocol instead.
- Is performance the primary concern and it is not important that messages be delivered in the proper order or that the sender is notified if any are lost? If so, use the lighter-weight UDP transport instead.
- Should messages be secured? If so, use the SSL transport.

Your Rating: 

Results:  3 rates

As shown in the examples below, the SSL transport can be used to

- [Create a SSL server](#)
- [Send messages to a SSL server](#)

The use of SSL with Java is described in detail in the [JSSE Reference Guide](#). In particular, it describes the keystores used by SSL, how the certificates they contain are used, and how to create and maintain them.

## Features

The SSL module allows a Mule application both to send and receive messages over SSL connections, and to declaratively customize the following features of SSL (with the standard name for each feature, where applicable):

- The timeout for blocking socket operations. This can be declared separately for client and server operations. (SO\_TIMEOUT)
- How long to keep the socket open to allow pending sends to complete. (SO\_LINGER)
- Whether to send available data immediately rather than buffering it. (TCP\_NODELAY)
- Whether to reuse a socket address immediately (SO\_REUSEADDR)
- Whether to use keep-alive to detect when a remote system is no longer reachable (SO\_KEEPALIVE).
- The size in bytes of the network buffer (SO\_SNDBUF).
- The number of pending connection requests to allow.
- Whether to close a client socket after sending a message.

In addition, since TCP and SSL are stream-oriented and Mule is message-oriented, some application protocol is needed to define where each message begins and ends within the stream. The table below lists the built-in protocols, describing:

- The XML tag used to specify them
- Any XML attributes
- How it defines a message when reading
- Any processing it does while writing a message

XML tag	Options	Read	Write	Notes
<tcp:custom-class-loading-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength, classLoader-ref	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	Like the length protocol, but specifies a classloader used to deserialize objects
<tcp:custom-protocol>	rethrowExceptionOnRead, class, ref	varies	varies	Allows user-written protocols, for instance, to match existing TCP services.
<tcp:direct-protocol>	rethrowExceptionOnRead, payloadOnly	All currently available bytes	none	There are no explicit message boundaries.
<tcp:eof-protocol>	rethrowExceptionOnRead, payloadOnly	All bytes sent until the socket is closed	none	
<tcp:length-protocol>	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to begin with a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with a 4-byte length (in DataOutput.writeInt() format)	
<tcp:safe-protocol	rethrowExceptionOnRead, payloadOnly , maxMessageLength	Expects the message to be preceded by the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Precedes the message with the string "You are using SafeProtocol" followed by a 4-byte length (in DataOutput.writeInt() format)	Somewhat safer than the length protocol because of the extra check. This is the default if no protocol is specified.
<tcp:streaming-protocol>	rethrowExceptionOnRead	All bytes sent until the socket is closed	none	
<tcp:xml-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration	none	The XML declaration must occur in all messages

<tcp:xml-eof-protocol>	rethrowExceptionOnRead	A message is an XML document that begins with an XML declaration, or whatever remains at EOF	none	The XML declaration must occur in all messages
------------------------	------------------------	--	------	--

Protocol attributes:

name	values	default value	notes
class	The name of the class that implements the custom protocol		See below for an example of writing a custom protocol
classLoader-ref	A reference to a Spring bean that contains the custom classloader		
maxMessageLength	the maximum message length allowed	0 (no maximum )	A message longer than the maximum causes an exception to be thrown.
payloadOnly	true	If true, only the Mule message payload is sent or received. If false, the entire Mule message is sent or received.	Protocols that don't support this attribute always process payloads
ref	A reference to a Spring bean that implements the custom protocol		
rethrowExceptionOnRead	Whether to rethrow exception that occur trying to read from the socket	false	Setting this to "false" avoids logging stack traces when the remote socket is closed unexpectedly

Your Rating: 

Results:  0 rates

SSL endpoints can be used in one of two ways:

- To create an SSL server that accepts incoming connections, declare an inbound ssl endpoint with an ssl:connector. This creates an SSL server socket that will read requests from and optionally write responses to client sockets..
- To write to an SSL server, create an outbound endpoint with an ssl:connector. This creates an SSL client socket that will write requests to and optionally read responses from a server socket.

## Usage

To use SSL endpoints, follow the following steps:

1. Add the MULE SSL namespace to your configuration:
  - Define the ssl prefix using xmlns:ssl="http://www.mulesoft.org/schema/mule/ssl"
  - Define the schema location with [http://www.mulesoft.org/schema/mule/ssl/3.1/mule-ssl.xsd](http://www.mulesoft.org/schema/mule/ssl/http://www.mulesoft.org/schema/mule/ssl/3.1/mule-ssl.xsd)
2. Define one or more connectors for SSL endpoints.

### Create an SSL server

To act as a server that listens for and accepts SSL connections from clients, create an SSL connector that inbound endpoints will use:

```
<ssl:connector name="sslConnector" />
```

### Send messages to an SSL server

To send messages on an SSL connection, create a simple TCP connector that outbound endpoints will use:

```
<tcp:connector name="sslConnector" />
```

1. Configure the features of each connector that was created.

- Begin by choosing the protocol to be used for each message that will be sent or received.
  - For each polling connector, choose how often it will poll and how long it will wait for the connection to complete.
  - Consider the other connector options as well. For instance, if it is important to detect when the remote system becomes unreachable, set `keepAlive` to `true`.
2. Create SSL endpoints.
- Messages will be received on inbound endpoints.
  - Messages will be sent to outbound endpoints.
  - Both kinds of endpoints are identified by a host name and a port.

By default, SSL endpoints use the request-response exchange pattern, but they can be explicitly configured as one-way. The decision should be straightforward:

Message flow	Connector type	Endpoint type	Exchange Pattern
Mule receives messages from clients but sends no response	ssl:connector	inbound	one-way
Mule receives messages from clients and sends response	ssl:connector	inbound	request-response
Mule sends messages to a server but receives no response	ssl:connector	outbound	one-way
Mule sends messages to a server and receives responses	ssl:connector	outbound	request-response

### Example Configurations

SSL connector in flow
<pre> &lt;ssl:connector name="serverConnector" payloadOnly="false"&gt;     &lt;tcp:eof-protocol /&gt;     &lt;ssl:client path="clientKeystore" /&gt;     &lt;ssl:key-store path="serverKeystore" /&gt; &lt;/tcp:connector&gt;  &lt;flow name="echo"&gt;     &lt;ssl:inbound-endpoint host="localhost" port="4444" &gt;         &lt;ssl:outbound-endpoint host="remote" port="5555" /&gt;     &lt;/flow&gt; </pre>
SSL connector in service
<pre> &lt;ssl:connector name="connector" payloadOnly="false"&gt;     &lt;tcp:safe-protocol /&gt;     &lt;ssl:client path="clientKeystore" /&gt;     &lt;ssl:key-store path="serverKeystore" /&gt; &lt;/ssl:connector&gt;  &lt;model name="echoModel"&gt;     &lt;service name="echo"&gt;         &lt;inbound&gt;             &lt;ssl:inbound-endpoint host="localhost" port="4444" /&gt;         &lt;/inbound&gt;         &lt;outbound&gt;             &lt;pass-through-router&gt;                 &lt;ssl:outbound-endpoint host="remote" port="5555" /&gt;             &lt;/pass-through-router&gt;         &lt;/outbound&gt;     &lt;/service&gt; &lt;/model&gt; </pre>

This shows how to create an SSL server in Mule. The connector at `defines that a server socket will be created that accepts connections from clients. Complete mule messages are read from the connection (direct protocol) will become the payload of a Mule message (since payload only is false). The endpoint at applies these definitions to create a server at port 4444 on the local host. The messages read from there are then sent to a remote ssl endpoint at .`

The flow version uses the `eof protocol ()`, so that every byte sent on the connection is part of the same Mule message. The service version uses the `safe protocol ()`, so that multiple messages can be sent on the SSL connection, with each being preceded by a header that specifies its length. Note that both connectors specify separate keystores to be used by the client (outbound) and server (inbound) endpoints.

## Configuration Options

SSL Connector attributes

Name	Description	Default
clientSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a TCP server socket	system default
keepAlive	Whether to send keep-alive messages to detect when the remote socket becomes unreachable	false
keepSendSocketOpen	Whether to keep the the socket open after sending a message	false
receiveBacklog	The number of connection attempts that can be outstanding	system default
receiveBufferSize	This is the size of the network buffer used to receive messages. In most cases, there is no need to set this, since the system default will be sufficient	system default
reuseAddress	Whether to reuse a socket address that's currently in a TIMED_WAIT state. This avoids triggering the error that the socket is unavailable	true
sendBufferSize	The size of the network send buffer	system default
sendTcpNoDelay	Whether to send data as soon as its available, rather than waiting for more to arrive to economize on the number of packets sent	false
socketSoLinger	How long (in milliseconds) to wait for the socket to close so that all pending data is flushed	system default
serverSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a client socket	system default

Your Rating: 

Results:  1 rates

SSL Connector child elements and their attributes

Name	Description
client	Configures the client keystore

Client's attributes:

Name	Description
path	location of the client keystore
storePassword	Password for the client keystore
class	the type of keystore used

Name	Description
key-store	Configures the server keystore

key-store's attributes:

Name	Description
path	location of the server keystore
storePassword	Password for the server keystore
class	the type of server keystore used
keyPassword	Password for the private key
algorithm	algorithm used by the server keystore

Name	Description

server	Configures the server trust store
--------	-----------------------------------

server's attributes:

Name	Description
class	the type of keystore used for the trust store
algorithm	algorithm used by the trust stor
factory-ref	A TrustManagerFactory configured as a Spring bean
explicitOnly	If true, do not use the server keystore when a trust store is unavailable. Defaults to false.
requireClientAuthentication	If true, all clients must authenticate themselves when communicating with a Mule SSL server endpoint. Defaults to false.

Name	Description
protocol-handler	Defines a list of Java packages in which protocol handlers are found

protocol-handler's attributes:

Name	Description
property	The list of packages.

For more details about creating protocol handlers in Java, see <http://java.sun.com/developer/onlineTraining/protocolhandlers>.

## Configuration Reference

### Element Listing

cache: Unexpected program error: java.lang.NullPointerException

## SSL Transport

The SSL transport can be used for secure socket communication using SSL or TLS. The Javadoc for this transport can be found [here](#).

### Connector

Connects Mule to an SSL socket to send or receive data via the network.

### Inbound endpoint

#### Attributes of <inbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

#### Child Elements of <inbound-endpoint...>

Name	Cardinality	Description
------	-------------	-------------

### Outbound endpoint

#### Attributes of <outbound-endpoint...>

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

**Child Elements of <outbound-endpoint...>**

Name	Cardinality	Description
------	-------------	-------------

**Endpoint**

**Attributes of <endpoint...>**

Name	Type	Required	Default	Description
host	string	no		
port	port number	no		

**Child Elements of <endpoint...>**

Name	Cardinality	Description
------	-------------	-------------

**Schema**

The schema for the SSL module appears [here](#). Its structure is shown below.

Namespace "http://www.mulesoft.org/schema/mule/ssl"

Targeting Schemas (1):

[mule-ssl.xsd](#)

Targeting Components:

4 global elements, 4 local elements, 3 complexTypes, 1 attribute group

Schema Summary	
<a href="#">mule-ssl.xsd</a>	<p>The SSL transport can be used for secure socket communication using SSL or TLS.</p> <p>Target Namespace: <a href="http://www.mulesoft.org/schema/mule/ssl">http://www.mulesoft.org/schema/mule/ssl</a></p> <p>Defined Components: 4 global elements, 4 local elements, 3 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form: Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location: <a href="http://www.mulesoft.org/schema/mule/ssl/3.1/mule-ssl.xsd">http://www.mulesoft.org/schema/mule/ssl/3.1/mule-ssl.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (4): <a href="#">mule-schemadoc.xsd</a>, <a href="#">mule-tcp.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p>
All Element Summary	
<a href="#">client</a>	<p>The client key store.</p> <p>Type: <a href="#">mule:tlsClientKeyStoreType</a></p> <p>Content:empty, 3 attributes</p> <p>Defined: locally within <a href="#">connector</a> element in <a href="#">mule-ssl.xsd</a>; see <a href="#">XML source</a></p>
<a href="#">connector</a>	<p>Connects Mule to an SSL socket to send or receive data via the network.</p> <p>Type: <a href="#">anonymous</a> (extension of <a href="#">tcp:tcpConnectorType</a>)</p> <p>Content: complex, 17 attributes, attr. wildcard, 10 elements</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-connector</a></p> <p>Defined: globally in <a href="#">mule-ssl.xsd</a>; see <a href="#">XML source</a></p> <p>Includes: definitions of 4 elements</p> <p>Used: never</p>
<a href="#">endpoint</a>	<p>Type: <a href="#">globalEndpointType</a></p>

	Content: complex, 13 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element mule:abstract-global-endpoint Defined: globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: never
inbound-endpoint	Type: <a href="#">inboundEndpointType</a> Content: complex, 13 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element mule:abstract-inbound-endpoint Defined: globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: never
key-store	The key store information, including location, key store type, and algorithm. Type: <a href="#">mule:tlsKeyStoreType</a> Content:empty, 5 attributes Defined:locally within connector element in <a href="#">mule-ssl.xsd</a> ; see XML source
outbound-endpoint	Type: <a href="#">outboundEndpointType</a> Content: complex, 13 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element mule:abstract-outbound-endpoint Defined: globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: never
protocol-handler	Configures the global Java protocol handler by setting the java.protocol.handler.pkgs system property. Type: <a href="#">mule:tlsProtocolHandler</a> Content:empty, 1 attribute Defined:locally within connector element in <a href="#">mule-ssl.xsd</a> ; see XML source
server	The server trust store. Type: <a href="#">mule:tlsServerTrustStoreType</a> Content:empty, 7 attributes Defined:locally within connector element in <a href="#">mule-ssl.xsd</a> ; see XML source

#### Complex Type Summary

<a href="#">globalEndpointType</a>	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: at 1 location
<a href="#">inboundEndpointType</a>	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: at 1 location
<a href="#">outboundEndpointType</a>	Content:complex, 13 attributes, attr. wildcard, 12 elements Defined:globally in <a href="#">mule-ssl.xsd</a> ; see XML source Used: at 1 location

#### Attribute Group Summary

<a href="#">addressAttributes</a>	Content: 2 attributes Defined: globally in <a href="#">mule-ssl.xsd</a> ; see XML source Includes:definitions of 2 attributes Used: at 3 locations
-----------------------------------	---

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by Altova XMLSpy via DocFlex/XML XMLSpy Integration.

#### Javadoc API Reference

The Javadoc for this module can be found here: [SSL](#)

#### Maven

The SSLModule can be included with the following dependency:

```
<dependency>
<groupId>org.mule.transports</groupId>
<artifactId>mule-transport-ssl</artifactId>
<version>3.1.0</version>
</dependency>
```

#### Extending this Transport

When using TCP to communicate with an external program, it may be necessary to write a custom Mule protocol. The first step is to get a complete description of how the external program delimits messages within the TCP stream. The next is to implement the protocol as a Java class.

- All protocols must implement the interface `org.mule.transport.tcp.TcpProtocol`, which contains three methods:
  - `Object read(InputStream is)` reads a message from the TCP socket
  - `write(OutputStream os, Object data)` writes a message to the TCP socket
  - `ResponseOutputStream createResponse(Socket socket)` creates a stream to which a response can be written.
- Protocols which process byte-streams rather than serialized Mule messages can inherit much useful infrastructure by subclassing `org.mule.transport.tcp.protocols.AbstractByteProtocol`. This class
  - implements `createResponse`
  - handles converting messages to byte arrays, allowing subclasses to implement only the simpler method `writeByteArray(OutputStream os, byte[] data)`
  - provides methods `safeRead(InputStream is, byte[] buffer)` and `safeRead(InputStream is, byte[] buffer, int size)` that handle the situation where data is not currently available when doing non-blocking reads from the TCP socket

Suppose we want to communicate with a server that has a simple protocol: all messages are terminated by `>>>`. The protocol class would look like this:

```
package org.mule.transport.tcp.integration;

import org.mule.transport.tcp.protocols.AbstractByteProtocol;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class CustomByteProtocol extends AbstractByteProtocol
{

    /**
     * Create a CustomByteProtocol object.
     */
    public CustomByteProtocol()
    {
        super(false); // This protocol does not support streaming.
    }

    /**
     * Write the message's bytes to the socket,
     * then terminate each message with '>>>'.
     */
    @Override
    protected void writeByteArray(OutputStream os, byte[] data) throws IOException
    {
        super.writeByteArray(os, data);
        os.write('>');
        os.write('>');
        os.write('>');
    }

    /**
     * Read bytes until we see '>>>', which ends the message
     */
    public Object read(InputStream is) throws IOException
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        int count = 0;
        byte read[] = new byte[1];

        while (true)
        {
            // if no bytes are currently available, safeRead()
            // will wait until some arrive
            if (safeRead(is, read) < 0)

```

```
{  
    // We've reached EOF.  Return null, so that our  
    // caller will know there are no  
    // remaining messages  
    return null;  
}  
  
byte b = read[0];  
if (b == '>')  
{  
    count++;  
    if (count == 3)  
    {  
        return baos.toByteArray();  
    }  
}  
else  
{  
    for (int i = 0; i < count; i++)  
    {  
        baos.write('>');  
    }  
    count = 0;  
    baos.write(b);  
}  
}
```

```
}
```

Your Rating:  Results:  0 rates

## Notes

TCP and SSL are very low-level transports, so the usual tools for debugging their use, for instance, logging messages as they arrive, might not be sufficient. Once messages are being sent and received successfully, things are largely working. It may be necessary to use software (or hardware) than can track messages at the packet level, particularly when a custom protocol is being used. Alternatively, you can debug by temporarily using the direct protocol on all inbound endpoints, since it will accept (and you can then log) bytes as they are received.

Your Rating:  Results:  0 rates

Your Rating:  Results:  1 rates

## TCP and SSL Debugging Notes

TCP and SSL are very low-level transports, so the usual tools for debugging their use, for instance, logging messages as they arrive, might not be sufficient. Once messages are being sent and received successfully, things are largely working. It may be necessary to use software (or hardware) than can track messages at the packet level, particularly when a custom protocol is being used. Alternatively, you can debug by temporarily using the direct protocol on all inbound endpoints, since it will accept (and you can then log) bytes as they are received.

Your Rating:  Results:  0 rates

## TCP Connector Attributes

Name	Description	Default
clientSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a TCP server socket	system default
keepAlive	Whether to send keep-alive messages to detect when the remote socket becomes unreachable	false
keepSendSocketOpen	Whether to keep the the socket open after sending a message	false
receiveBacklog	The number of connection attempts that can be outstanding	system default
receiveBufferSize	This is the size of the network buffer used to receive messages. In most cases, there is no need to set this, since the system default will be sufficient	system default
reuseAddress	Whether to reuse a socket address that's currently in a TIMED_WAIT state. This avoids triggering the error that the socket is unavailable	true
sendBufferSize	The size of the network send buffer	system default
sendTcpNoDelay	Whether to send data as soon as its available, rather than waiting for more to arrive to economize on the number of packets sent	false
socketSoLinger	How long (in milliseconds) to wait for the socket to close so that all pending data is flushed	system default
serverSoTimeout	the amount of time (in milliseconds) to wait for data to be available when reading from a client socket	system default

Your Rating:  Results:  1 rates

## BPM Transport Reference

### BPM Transport Reference



As of Mule 3.0.1, the recommended way to interact with a BPM system is via the `<bpm:process>` component / message processor. This approach is documented here. Usage of the legacy BPM transport is still supported for 3.0.x but will be removed for 3.1.

[ Features ] [ Usage ] [ Integration with BPM Engines ] [ JBoss jBPM ]

The BPM transport allows Mule events to initiate and/or advance processes in a Business Process Management System (BPMS), also known as a process engine. It also allows executing processes to generate Mule events. For a high-level introduction to Business Process Management with Mule and the concepts involved, read this [blog entry](#) or [presentation](#).

To see the BPM Connector in action (with JBoss jBPM), take a look at the [Loan Broker BPM Example](#) (available in the full Mule distribution).

Javadocs for the BPM transport are available [here](#).

## Features

- Incoming Mule events can launch new processes, advance or terminate running processes.
- A running process can send synchronous or asynchronous messages to any Mule endpoint.
- Synchronous responses from Mule are automatically fed back into the running process and can be stored into process variables.
- Endpoints of type "bpm://MyProcess" are used to intelligently route process-generated events within Mule.
- Mule can interact with different running processes in parallel.



### BPEL

The connector can only integrate with BPM engines that provide a Java API. If you need to integrate with a BPEL engine that only exposes SOAP endpoints, you will need to [use standard web services](#).

## Usage

```
<model>
    <service name="MessagesToProcess">
        <inbound>
            <jms:inbound-endpoint queue="queueA" />
        </inbound>
        <outbound>
            <pass-through-router>
                <bpm:outbound-endpoint process="myProcess" />
            </pass-through-router>
        </outbound>
    </service>

    <service name="MessagesFromProcess">
        <inbound>
            <bpm:inbound-endpoint process="myProcess" />
        </inbound>
        <outbound>
            <bpm:outbound-router>
                <jms:outbound-endpoint queue="queueC" />
                <jms:outbound-endpoint queue="queueD" />
            </bpm:outbound-router>
        </outbound>
    </service>
</model>
```

- Incoming Mule messages and properties can be stored as process variables, and process variables can be sent as Mule messages. How to configure this depends on the BPMS and its process definition language.
- Incoming messages from Mule to the BPMS are correlated based on the message property `MULE_BPM_PROCESS_ID`. If a process already exists with this ID, then the message will advance the existing process one step. Otherwise, a new process will be created and started. Any outgoing messages generated by the process will have `MULE_BPM_PROCESS_ID` set on them. In the case of an asynchronous response, it is important that your application maintain this property in the response message so that it gets correlated with the correct process instance.
- The `process` attribute on the endpoint is used to differentiate among multiple process definitions. For example, a message sent to the endpoint `<bpm:outbound-endpoint process="foo" />` will start or advance the process called "foo".

## Integration with BPM Engines

One of the basic design principles of Mule is to promote maximum flexibility for the user. Based on this, the user should ideally be able to "plug in" any BPM engine to use with Mule. Unfortunately, there is no standard JEE specification to enable this. Therefore, the Mule BPM Transport simply defines its own simple interface.

```
public interface BPMS
{
    public Object startProcess(Object processType, Object transition, Map processVariables) throws
Exception;

    public Object advanceProcess(Object processId, Object transition, Map processVariables) throws
Exception;

    // MessageService contains a callback method used to generate Mule messages from your process.
    public void setMessageService(MessageService msgService);
}
```

Any BPM engine that implements the interface (`org.mule.transport.bpm.BPMS`) can "plug in" to Mule via the BPM transport. Currently, the **jBPM** engine is the only one included in the Mule distribution. Other implementations may be found on the [MuleForge](#).

### jBPM

jBoss jBPM is the best-of-breed open source BPMS and is well-integrated with Mule. For general information on jBPM and how to configure it, refer to the [jBPM User Guide](#)

#### **jBPM Configuration**

In order to use jBPM with Mule, you need to have two important files on your classpath:

##### **jbpm.cfg.xml**

This file contains the jBPM configuration. If defaults are ok for you, then it could be as simple as the following. Note that you need to define the `MuleMessageService` within `<process-engine-context>` otherwise jBPM will not be able to "see" Mule.

```
<jbpm-configuration>
    <import resource="jbpm.default.cfg.xml" />
    <import resource="jbpm.jpd1.cfg.xml" />
    <import resource="jbpm.tx.hibernate.cfg.xml" />

    <process-engine-context>
        <object class="org.mule.transport.jbpm.MuleMessageService" />
    </process-engine-context>
</jbpm-configuration>
```

##### **jbpm.hibernate.cfg.xml**

This file contains the Hibernate configuration. jBPM uses Hibernate to persist its objects to an RDBMS, so you will need to configure your database settings here. For example, a simple in-memory Derby database might use these settings:

```
<property name="hibernate.dialect">org.hibernate.dialect.DerbyDialect</property>
<property name="hibernate.connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</property>
<property name="hibernate.connection.url">jdbc:derby:muleEmbeddedDB</property>
<property name="hibernate.hbm2ddl.auto">create-drop</property>
```

while an Oracle database might use these settings:

```
<property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:user/pass@server:1521:dbname</property>
```

## Mule configuration

Configuring the BPM connector with jBPM is then as simple as using the `<bpm:j bpm-connector>` element and giving it the location of your process definition(s). These processes will be loaded into jBPM when the connector starts up.

```
<bpm:j bpm-connector name="bpmConnector">
  <bpm:process name="processA" resource="processADef.jpd1.xml" />
  <bpm:process name="processB" resource="processBDef.jpd1.xml" />
</bpm:j bpm-connector>
```

## Process definition (jPDL)

Mule provides two custom elements for jBPM's process definition language (jPDL). You can combine these in your process definition with other standard jPDL elements such as `<state>`, `<java>`, `<script>`, `<decision>`.

### <mule-send>

*Usage:* `<mule-send expr="" endpoint="" synchronous="" var="" type="">`

Activity which sends a message with the payload `expr` to the Mule `endpoint`. If `synchronous` = true (the default value), the send will block and the response message will be stored into `var`. If the message is not of `type`, an exception will be thrown. `expr` can be a literal value or an expression which references process variables.

The only mandatory attributes are `expr` and `endpoint`, the rest are optional.

### <mule-send> example

```
<mule-send name="sendToMediumBank" expr="#{loanRequest}" endpoint="MediumBank" var="loanQuote" type="org.mule.example.loanbroker.messages.LoanQuote">
```

### <mule-receive>

*Usage:* `<mule-receive var="" endpoint="" type="">`

Wait state which expects a message to arrive from the Mule `endpoint` and stores it into `var`. If the message is not of `type`, an exception will be thrown.

The attributes are all optional.

`<mule-receive>` can replace `<start>` as the first state of a process and this way you can store the message which initiated the process into a variable.

### <mule-receive> example

```
<mule-receive name="waitForCreditAgency" endpoint="CreditProfiles" var="creditProfile">
```

Your Rating: 

Results:  0 rates

## Modules Reference

### Modules Reference

This topic relates to the most recent version of Mule ESB

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

Modules are similar to transports in that they provide pluggable functionality, configured via dedicated schema, but they differ in that there is no underlying transport to send or receive data. Following is a list of the default Mule ESB modules.

Module	Description
Acegi Module	Security via Acegi.
Atom Module	Support for consuming and creating Atom feeds.
BPM	Mule's support for BPM allows you to send/receive messages to/from a running process. A message from Mule can start or advance a process, the message can be stored as a process variable, and a running process can send messages to any endpoint in your Mule application.
CXF Module	Mule 3.0 architectural changes bring much better support for CXF, meaning it can be used just like another pipe/filter element.
Client Module	MuleClient and the remote dispatcher, giving simple access to the Mule server.
JAAS Module	Security via JAAS.
JBoss Transaction Manager	JBoss transaction support.
jBPM Module	JBoss jBPM is a best-of-breed open source BPMS and is well-integrated with Mule. One advantage of jBPM is that it is embedded directly in the Mule runtime, allowing for faster performance.
Jersey Module	Support for RESTful web services built using Jersey.
JSON Module	JSON data and binding support.
Management Module	Mule agents for server management using JMX.
OGNL Module	Provides a filter using OGNL expressions. For details, see <a href="#">Using OGNL Expressions</a> .
PGP Module	Security via PGP.
 SAML Module	Provides authentication and authorization capabilities to Mule based on the SAML standard. (As of Mule enterprise edition 2.2.3)
RSS Module	Support for consuming RSS feeds
Scripting Module	Interface between Mule and scripting languages (currently Groovy).
Spring Extras Module	Extensions for using the Spring framework with Mule.
SXC Module	A very fast streaming XPath router and filter.
XML Module	XML based utilities (mainly filters and routers).

## CXF Module Reference

### CXF Module Reference

[ [Introduction](#) ] [ [Using the CXF Module](#) ] [ [Troubleshooting](#) ]

#### Introduction

The Mule CXF module provides support for web service integration via Apache CXF. Apache CXF is an open source services framework that helps you build and develop services using front-end programming APIs, like JAX-WS.

#### Using the CXF Module

Use the following links for information on configuring and using the CXF transport with your implementation of Mule.

## Building and Consuming Services

- Overview
- Building Web Services
- Consuming Web Services
- Proxying Web Services
- Using HTTP GET Requests
- Bookstore Example
- Configuration Reference
- Using the WSDL Connector
- Upgrading CXF from Mule 2

## Web Service Standards

- Supported Web Service Standards
- Using MTOM
- Enabling WS-Addressing
- Enabling WS-ReliabilityMessaging
- Enabling WS-Security
- WS-Trust, WS-SecureConversation, and WS-SecurityPolicy on the CXF website

## Troubleshooting

This section includes common problems and solutions you might encounter when using the CXF transport.

### I've received a "java.lang.IllegalArgumentException: wrong number of arguments" when using the CXF outbound endpoint

The CXF outbound endpoint uses the CXF generated to client to send messages. Because of this, your message payload must match the signature of the operation being invoked. This error results when the payload does not match the signature.

Here's an example. If you have an operation on the client like this:

```
public void sendData(String data1, String data2, String data3);
```

Your message payload must be like this:

```
Object payload = new Object[] { "data1", "data2", "data3" };
```

### My WSDL does not have the correct service address (i.e. its localhost instead of foo.com)

If you are doing WSDL-first development, ensure that your @WebService annotation on your service implementation has the correct name, serviceName, targetNamespace, and portName attributes. If these are not correct, CXF cannot navigate your WSDL, find the address, and replace it with the address currently being used.

Your WebService annotation should look like this:

```
@WebService(name = "YourWSDLPortType",
            serviceName = "YourServiceName",
            portName = "YourPortName",
            targetNamespace = "http://your-namespace",
            endpointInterface = "your.endpoint.Interface",
            wsdlLocation = "your.wsdl")
```

Your Rating:  Results:  0 rates

## CXF Module Configuration Reference

### CXF Module Configuration Reference

[ Configuration ] [ Jaxws service ] [ Jaxws client ] [ Common CXF Elements ] [ Wrapper component ]

This page provides reference information about the elements and attributes you can configure for the [CXF Module Reference](#). For an example of using CXF, see the [Echo Example](#).

- As of Mule enterprise edition 2.2.3, to use HTTP with a CXF 1.0 client endpoint, add the following to your transformers:

```
<message-properties-transformer>
  <add-message-property key="http.version" value="HTTP/1.0"/>
</message-properties-transformer>
```

## Configuration

### Attributes of <configuration...>

Name	Type	Required	Default	Description
configurationLocation	string	no		The location of a CXF configuration file, if any needs to be supplied.
enableMuleSoapHeaders	boolean	no	true	Whether or not CXF should write Mule SOAP headers which pass along the correlation and ReplyTo information. This is true by default, but the Mule SOAP headers are only triggered in situations where there is an existing correlation ID and the ReplyTo header is set. (As of 2.2.1)
initializeStaticBusInstance	boolean	no	true	Initialize the static CXF Bus with a Bus configured to use Mule for all transports. This will affect any CXF generated clients that you run standalone. Defaults to true.
name	string	no	_cxfrConfiguration	

### Child Elements of <configuration...>

Name	Cardinality	Description
------	-------------	-------------

## Jaxws service

### Attributes of <jaxws-service...>

Name	Type	Required	Default	Description
bindingId	string	no		The binding that should be used for this service. It defaults to the SOAP binding by default.
port	string	no		The WSDL port name of your service.
namespace	string	no		The service namespace. (As of 2.2.1)
service	string	no		The WSDL service name of your service.
serviceClass	string	no		The class CXF should use to construct its service model. This is optional, and by default it will use the implementation class of your component, on inbound cxf endpoint. But it is mandatory for outbound endpoint when using "aegis" frontend.
validationEnabled	boolean	no		Whether or not validation should be enabled on this service. Validation only occurs on inbound server messages.
mtomEnabled	boolean	no		Whether or not MTOM (attachment support) is enabled for this endpoint.
wsdlLocation	string	no		The location of the WSDL for your service. If this is a server side endpoint it will be served to your users.
enableMuleSoapHeaders	boolean	no	true	Whether or not this endpoint should write Mule SOAP headers which pass along the correlation and ReplyTo information. This is true by default, but the Mule SOAP headers are only triggered in situations where there is an existing correlation ID and the ReplyTo header is set. (As of 2.2.1)
configuration-ref	string	no		The CXF configuration that should be used.

### Child Elements of <jaxws-service...>

Name	Cardinality	Description
schemaLocations	0..1	
configuration-ref	0..1	
databinding	0..1	The databinding implementation that should be used. By default, this is JAXB for the JAX-WS frontend and Aegis for the simple frontend. This should be specified in the form of a Spring bean.
features	0..1	Any CXF features you want to apply to the client/server. See the CXF documentation for more information on features.
inInterceptors	0..1	Additional incoming interceptors for this service.
inFaultInterceptors	0..1	Additional incoming fault interceptors.
outInterceptors	0..1	Additional outgoing interceptors.
outFaultInterceptors	0..1	Additional outgoing fault interceptors.

### Jaxws client

#### Attributes of <jaxws-client...>

Name	Type	Required	Default	Description
mtomEnabled	boolean	no		Whether or not MTOM (attachment support) is enabled for this endpoint.
wsdlLocation	string	no		The location of the WSDL for your service. If this is a server side endpoint it will be served to your users.
enableMuleSoapHeaders	boolean	no	true	Whether or not this endpoint should write Mule SOAP headers which pass along the correlation and ReplyTo information. This is true by default, but the Mule SOAP headers are only triggered in situations where there is an existing correlation ID and the ReplyTo header is set. (As of 2.2.1)
configuration-ref	string	no		The CXF configuration that should be used.
serviceClass	string	no		The class CXF should use to construct its service model for the client.
decoupledEndpoint	string	no		The reply to endpoint for clients which have WS-Addressing enabled.
operation	string	no		The operation you want to invoke on the outbound endpoint.
clientClass	string	no		The name of the client class that CXF generated using CXF's wsdl2java tool. You must use wsdl2java if you do not have both the client and the server in the same JVM. Otherwise, this can be optional if the endpoint address is the same in both cases.
port	string	no		The WSDL port you want to use to communicate with the service.

#### Child Elements of <jaxws-client...>

Name	Cardinality	Description
configuration-ref	0..1	
databinding	0..1	The databinding implementation that should be used. By default, this is JAXB for the JAX-WS frontend and Aegis for the simple frontend. This should be specified in the form of a Spring bean.
features	0..1	Any CXF features you want to apply to the client/server. See the CXF documentation for more information on features.
inInterceptors	0..1	Additional incoming interceptors for this service.
inFaultInterceptors	0..1	Additional incoming fault interceptors.
outInterceptors	0..1	Additional outgoing interceptors.
outFaultInterceptors	0..1	Additional outgoing fault interceptors.

### Common CXF Elements

Following are the sub-elements you can set on CXF service and client. For further information on CXF interceptors, see the [CXF documentation](#).

Name	Description
databinding	The databinding implementation that should be used. By default, this is JAXB for the JAX-WS frontend and Aegis for the simple frontend. This should be specified in the form of a Spring bean.
features	Any CXF features you want to apply to the client/server. See the CXF documentation for more information on features.
inInterceptors	Additional incoming interceptors for this service.
inFaultInterceptors	Additional incoming fault interceptors.
outInterceptors	Additional outgoing interceptors.
outFaultInterceptors	Additional outgoing fault interceptors.

### Interceptors example

```
<cxf:jaxws-client serviceClass="com.mulesoft.example.HelloWorld"
                   operation="sayHello" port="HelloWorldPort">
    <cxf:inInterceptors>
        <spring:bean class="org.apache.cxf.interceptor.LoggingInInterceptor"/>
    </cxf:inInterceptors>
    <cxf:outInterceptors>
        <spring:bean class="org.apache.cxf.interceptor.LoggingOutInterceptor"/>
    </cxf:outInterceptors>
</cxf:jaxws-client>
```

### Databinding example

```
<cxf:simple-service>
    <cxf:databinding>
        <spring:bean class="org.apache.cxf.aegis.databinding.AegisDatabinding">
            <spring:property name="configuration">
                <spring:bean class="org.apache.cxf.aegis.type.TypeCreationOptions" />
            </spring:property>
        </spring:bean>
    </cxf:databinding>
</cxf:simple-service>
```

### Features example

```
<cxf:jaxws-service serviceClass="com.mulesoft.mule.example.security.Greeter">
    <cxf:features>
        <spring:bean class="org.mule.module.cxf.feature.PrettyLoggingFeature" />
    </cxf:features>
</cxf:jaxws-service>
```

## Wrapper component

The WebServiceWrapperComponent class allows you to send the result of a web service call to another endpoint.

### Attributes of <wrapper-component...>

Name	Type	Required	Default	Description
address	string	no		The URL of the web service.
addressFromMessage	boolean	no		Specifies that the URL of the web service will be obtained from the message itself.
wsdlPort	string	no		The WSDL port you want to use to communicate to the service.

operation	string	no		The operation you want to invoke on the outbound endpoint.	
-----------	--------	----	--	--	--

#### Child Elements of <wrapper-component...>

Name	Cardinality	Description
------	-------------	-------------

Your Rating:  Results:  0 rates

## CXF Module Overview

### CXF Module Overview

The CXF web services support inside Mule allows you to build sophisticated web service applications and do things like:

- Implement JAX-WS code first or WSDL first services
- Consume WSDL based web services
- Create a proxies/gateways which mediate SOAP messages to:
  - Create a WS-Security gateway
  - Do content based routing
  - Validate incoming requests against the WSDL
- Build reliable web services using WS-Addressing and WS-ReliableMessaging and a JDBC store.
- Build secure services which are transport agnostic using WS-Security and WS-SecureConversation.
- Configure services using WS-Policy

For information about what standards are supported, see [Supported Web Service Standards](#).

When using CXF inside of Mule, there are several different ways to build and consume service:

Frontend Mode	Server	Client
Simple	< <b>simple-service</b> > builds services based on simple POJOs - no annotations are needed. CXF will introspect your POJOs and generated a WSDL for them	< <b>simple-client</b> > allows you to interact with a service which was built with the simple frontend if you have a copy of the service interface.
JAX-WS	< <b>jaxws-service</b> > builds a web service message processor which using the JAX-WS and JAXB standard annotations or from a set of classes generated from a WSDL. These annotations give you complete control over how your schemas and WSDL are generated.	< <b>jaxws-client</b> > builds a message processor which can operate in two modes: 1) it can use a JAX-WS client generated from WSDL. 2) it can use a JAX-WS service interface which matches your server interface.
Proxy	< <b>proxy-service</b> > provides raw SOAP and WS-* processing for incoming XML messages, allowing you to apply things like WS-Security to incoming messages and work with the raw XML.	< <b>proxy-client</b> > provides raw SOAP and WS-* processing for outgoing XML messages, allowing you to send outgoing messages in raw XML form and apply things like WS-Security to them.

Your Rating:  Results:  3 rates

## Building Web Services with CXF

### Building Web Services with CXF

[ [Creating a JAX-WS Service](#) ] [ [Creating a WSDL First JAX-WS Service](#) ] [ [Creating a simple frontend web service](#) ] [ [Advanced Configuration](#) ]

This page describes how to build a CXF web service and use it in Mule.

There are three ways to build a web service.

1. Use the JAX-WS frontend to build a code first web service using the standard JAX-WS annotations with the JAXB databinding
2. Use the JAX-WS frontend to build a WSDL first web service
3. Use the "simple" frontend in CXF to create a web service from simple POJOs

### Creating a JAX-WS Service

The JAX-WS specification is a series of APIs and annotations to help you build web services. This section describes how to create a very simple JAX-WS web service.

First, you write the service interface. For example, you could write an operation called "sayHello" that says "Hello" to whoever submits their name.

```
package org.example;

import javax.jws.WebService;

@WebService
public interface HelloWorld {
    String sayHi(String text);
}
```

Your implementation would then look like this:

```
package org.example;

import javax.jws.WebService;

@WebService(endpointInterface = "org.example.HelloWorld",
            serviceName = "HelloWorld")
public class HelloWorldImpl implements HelloWorld {

    public String sayHi(String text) {
        return "Hello " + text;
    }
}
```

## Configuring the service

To configure Mule to use the service, simply declare your service and use a CXF endpoint as shown in the following example:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:cxf="http://www.mulesoft.org/schema/mule/cxf"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
          http://www.mulesoft.org/schema/mule/cxf
          http://www.mulesoft.org/schema/mule/cxf/3.1/mule-cxf.xsd">

    <flow name="helloService">
        <http:inbound-endpoint address="http://localhost:63081/hello" exchange-pattern="request-response">
            <cxf:jaxws-service serviceClass="org.example.HelloWorld"/>
        </http:inbound-endpoint>
        <component class="org.example.HelloWorldImpl" />
    </flow>
    ...

```

If you go to "http://localhost:63081/hello?wsdl", you will see the WSDL that CXF generates for you.

If you want to use a POJO instead of creating an annotated JAX-WS service, you could host the POJO as a service component in Mule and use the simple front-end client with its CXF inbound endpoint.

## Injecting resources into JAX-WS services

If you need access to JAX-WS resources you can have the injected into your service implementation class. Simply add an annotated field like this:

```

@WebService
public class HelloWorldImpl implements HelloWorld {

    @Resource
    private WebServiceContext context;
}

```

To enable injection into your service class you need to declare it as a Spring bean so that CXF's processor can perform the injection on it.

```

<spring:bean class="org.apache.cxf.bus.spring.Jsr250BeanPostProcessor" />
<spring:bean id="hello" class="org.example.HelloWorldImpl" />

<flow name="helloService">
    <http:inbound-endpoint address="http://localhost:63081/hello" exchange-pattern="request-response">
        <cxfrs:jaxws-service serviceClass="org.example.HelloWorld" />
    </http:inbound-endpoint>
    <component>
        <spring-object bean="hello" />
    </component>
</flow>

```

NOTE: if your component gets executed in a different thread, injection will not work correctly because of limitations in CXF. This means that you cannot use this feature together with endpoints that do not use the request-response exchange-pattern or with the <async/> flow construct.

### **Creating a WSDL First JAX-WS Service**

In addition to the code-first approach outlined above, you can also use CXF to do WSDL-first services. While the details are out of the scope of this guide, the CXF distribution includes many examples of how to do this.

First, you generate your web service interface from your WSDL. You can do this using the [WSDL to Java tool](#) from CXF or the [Maven plugin](#).

Next, you write a service implementation class that implements your service interface. You can then use this implementation class in the Mule configuration exactly as in the previous example.

### **Supplying the Original WSDL to CXF**

You can supply your original WSDL to CXF by using the `@WebService` attribute:

```

@WebService(endpointInterface = "demo.hw.server.HelloWorld",
           serviceName = "HelloWorld",
           wsdlLocation="foo/bar/hello.wsdl")
public class HelloWorldImpl implements HelloWorld

```

Another way is to specify the `wsdlLocation` property on the endpoint:

```

<cxfrs:jaxws-service address="http://localhost:63081/hello" wsdlLocation="foo/bar/hello.wsdl" />

```

CXF is able to locate this WSDL inside your webapp, on the classpath, or on the file system.

### **Creating a simple frontend web service**

The simple frontend allows you to create web services which don't require any annotations. First, you write the service interface. As in the example above, you could write an operation called "sayHello" that says "Hello" to whoever submits their name.

Note: You don't have to use a service interface, you can just use an implementation class. However, the service interface makes it possible to consume the service very easily. See [Consuming Web Services](#) for more information.

```

package org.example;

public interface HelloWorld {
    String sayHi(String text);
}

```

Your implementation would then look like this:

```

package org.example;

public class HelloWorldImpl implements HelloWorld {

    public String sayHi(String text) {
        return "Hello " + text;
    }
}

```

### Configuring the service

To configure Mule to use the service, simply declare your service and use a CXF message processor as shown in the following example:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:cxf="http://www.mulesoft.org/schema/mule/cxf"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core
        http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/cxf
        http://www.mulesoft.org/schema/mule/cxf/3.1/mule-cxf.xsd">

    <flow name="helloService">
        <http:inbound-endpoint address="http://localhost:63081/hello" exchange-pattern="request-response">
            <cxf:simple-service serviceClass="org.example.HelloWorld"/>
        </http:inbound-endpoint>
        <component class="org.example.HelloWorldImpl" />
    </flow>
    ...

```

If you go to "<http://localhost:63081/hello?wsdl>", you will see the WSDL that CXF generates for you.

### Advanced Configuration

#### Validation of Messages

To enable schema validation for incoming messages add a validationEnabled attribute to your service declaration. For example:

```

<simple-service validationEnabled="true" />
<jaxws-service validationEnabled="true" />
<proxy-service validationEnabled="true" />

```

### Changing the Data Binding

You can use the databinding property on an endpoint to configure the data binding to use with that service. Following are the types of data bindings available with CXF:

1. AegisDatabinding

2. JAXBDataBinding (Default)
3. StaxDataBinding

You specify the databinding class to use as follows:

```
<cxn:simple-service serviceClass="com.acme.MyService">
  <cxn:databinding>
    <spring:bean class="org.apache.cxf.aegis.databinding.AegisDataBinding" />
  </cxn:databinding>
</cxn:simple-service>
```

The `<cxn:databinding>` element can be used with any CXF frontend.

#### Setting the Binding URI

The `bindingUri` attribute specifies how your service operations are mapped to resources. You configure this attribute as follows:

```
<cxn:jaxws-service serviceClass="com.acme.MyService" bindingUri=
  "http://www.w3.org/2003/05/soap/bindings/HTTP/" />
```

#### Changing the Default Message Style

By default, CXF uses the Document/Literal message style. However, you can change the service to be exposed as RPC instead of document or to send complex types as wrapped instead of literal. To change the message style, you set the `@SOAPBinding` annotation on the service's interface, specifying the style, use, and optionally the parameterStyle.

In the following example, the parameter style is set to BARE. This means that each parameter is placed into the message body as a child element of the message root. This is WRAPPED by default.

```
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
             use=SOAPBinding.Use.LITERAL,
             parameterStyle=SOAPBinding.ParameterStyle.BARE)
@WebService
public interface Echo
{
    String echo(String src);
}
```

For more information on the supported message styles, go to [Optional Annotations](#).

Your Rating:  Results:  4 rates

## Consuming Web Services with CXF

### Consuming Web Services with CXF

This page describes how to consume web services using the CXF client message processors.

There are 4 ways to consume web services. The first 3 correspond to the 3 ways of [building web services](#):

1. Generate and use a client from a WSDL
2. Use a client based on the interface of a JAX-WS service
3. Use a client based on the interface of a "simple" frontend web service
4. Use the JAX-WS Java client API

The last two options are only usable if you have the Java interface for your service, meaning that their use is normally limited to use within an organization.

#### **WSDL First JAX-WS Client**

You can use a CXF-generated client as an outbound endpoint. First, you generate a CXF client using the [WSDL to Java](#) tool from CXF or the [Maven plugin](#). (Note: the CXF transport doesn't support wrapper-style web service method calls. You may need to create a binding file or change

the WSDL directly. See the [WSDL to Java tool page](#) for more details.)

Next, you configure the client as an outbound endpoint using the following properties:

- `clientClass`: The client class generated by CXF, which extends `javax.xml.ws.Service`.
- `port`: The WSDL port to use for communicating with the service
- `wsdlLocation`: The location of the WSDL for the service. CXF uses this to configure the client.
- `operation`: The operation name to invoke on the web service. The objects that you pass to the outbound router must match the signature of the method for this operation. If your method takes multiple parameters, they must be put in an `Object[]` array.

Here is a simple example:

```
<flow name="csvPublisher">
    ...
    <jaxws-client
        clientClass="org.apache.hello_world_soap_http.SOAPService"
        wsdlPort="SoapPort"
        wsdlLocation="classpath:/wsdl/hello_world.wsdl"
        operation="greetMe"/>
    <outbound-endpoint address="http://localhost:63081/services/greeter"/>
</flow>
```

### JAX-WS Code First Client

You can also build a client for your JAX-WS services without the need to generate a client from WSDL. To do this, you need a copy of your service interface and all your data objects locally to use. This can simplify consuming web services if you already have access to the code used to build the service.

```
<flow name="csvPublisher">
    ...
    <jaxws-client serviceClass="org.example.HelloService" operation="sayHi"/>
    <outbound-endpoint address="http://localhost:63081/services/greeter"/>
</flow>
```

### Using the JAX-WS Client API

This section describes how to use the JAX-WS client APIs to talk to web services. This allows you to talk to web services outside of Mule configurations.

There are two ways to use CXF clients. First, you can generate a client from WSDL using the CXF WSDL to Java tool. Second, if you've built your service via "code-first" methodologies, you can use the service interface to build a client proxy object.

When using a CXF client, it automatically discovers the Mule instance (provided you're in the same JVM/ClassLoader) and uses it for your transport. Therefore, if you've generated a client from WSDL, invoking a service over Mule can be as simple as the following:

```
HelloWorldService service = new HelloWorldService();
HelloWorld hello = service.getSoapPort();

// Possibly set an alternate request URL:
// ((BindingProvider) greeter).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
// "http://localhost:63081/greeter");
String sayHi = hello.sayHi();
```

### Building a Client Proxy

Following is an example of how to construct a client using the service that was developed in [Building Web Services with CXF](#):

```

QName SERVICE_NAME = new QName("http://server.hw.demo/", "HelloWorld");
QName PORT_NAME = new QName("http://server.hw.demo/", "HelloWorldPort");

Service service = Service.create(SERVICE_NAME);

// Endpoint Address
String endpointAddress = http://localhost:63081/hello;

// Add a port to the Service
service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING, endpointAddress);

HelloWorld hw = service.getPort(HelloWorld.class);

System.out.println(hw.sayHi("World"));

```

### **Simple Frontend Clients**

You can build a client for your simple frontend based services without the need to generate a client from WSDL. To do this, you need a copy of your service interface and all your data objects locally to use. This can simplify consuming web services if you already have access to the code used to build the service.

```

<flow name="csvPublisher">
  ...
  <cxft:simple-client serviceClass="org.example.HelloService" operation="sayHi"/>
  <outbound-endpoint address="http://localhost:63081/services/greeter"/>
</flow>

```

### **Additional Resources**

- Developing a JAX-WS consumer
- WSDL to Java
- CXF Maven plugin

Your Rating:  Results:  2 rates

## **Enabling WS-Addressing**

### **Enabling WS-Addressing**

CXF supports the 2004-08 and 1.0 versions of WS-Addressing. To enable WS-Addressing on your services, you must configure the CXF service to use the WS-Addressing feature:

```

<cxf:jaxws-service ... >
  <cxf:features>
    <wsa:addressing />
  </cxf:features>
</cxf:jaxws-service/>

```

This works with clients as well:

```

<cxf:jaxws-client ... >
  <cxf:features>
    <wsa:addressing />
  </cxf:features>
</cxf:jaxws-client/>

```

### **Enabling WS-Addressing Globally**

To enable WS-Addressing globally, there are two steps. First, create an external configuration file with the following:

```
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://cxf.apache.org/core"
    xmlns:wsa="http://cxf.apache.org/ws/addressing"
    xmlns:wsrm-mgr="http://cxf.apache.org/ws/rm/manager"
    xsi:schemaLocation="
        http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
        http://schemas.xmlsoap.org/ws/2005/02/rm/policy
        http://schemas.xmlsoap.org/ws/2005/02/rm/wsrm-policy.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <cxf:bus>
        <cxf:features>
            <wsa:addressing />
        </cxf:features>
    </cxf:bus>

</beans>
```

Second, you'll need to tell the CXF module to use this configuration file:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:quartz="http://www.mulesoft.org/schema/mule/quartz"
    xmlns:cxf="http://www.mulesoft.org/schema/mule/cxf"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/cxf
        http://www.mulesoft.org/schema/mule/quartz
        http://www.mulesoft.org/schema/mule/quartz.xsd">

    <cxf:configuration configurationLocation="myConfig.xml"/>
    ...
</mule>
```

## Asynchronous Reply To Destinations

**NOTE: this requires Mule 3.1.**

To set up an asynchronous reply to destination for your client, you can set the "decoupledEndpoint" attribute property on your client message processor.

```
<cxf:jaxws-client ... decoupledEndpoint="http://localhost:8888/myReplyDestination">
    <cxf:features>
        <wsa:addressing />
    </cxf:features>
</cxf:jaxws-client/>
```

The URL in the decoupledEndpoint property will be used as your reply to destination for all messages.

## Manipulating WS-Addressing Headers

To manipulate the WS-Addressing headers inside the messages, you can write a JAX-WS handler or a CXF interceptor. For a quick start, see the WS-Addressing sample inside the CXF distribution.

Your Rating:  5 stars

Results:  0 rates

## Enabling WS-ReliableMessaging

### Enabling WS-ReliableMessaging

As of 3.1, Mule includes support for ReliableMessaging.

CXF supports the February 2005 version of the Web Services Reliable Messaging Protocol (WS-ReliableMessaging) specification. To enable WS-RM globally, there are two steps. First, create an external configuration file with the following:

```
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://cxf.apache.org/core"
    xmlns:wsa="http://cxf.apache.org/ws/addressing"
    xmlns:wsrm-policy="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
    xmlns:wsrm-mgr="http://cxf.apache.org/ws/rm/manager"
    xsi:schemaLocation="
        http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
        http://schemas.xmlsoap.org/ws/2005/02/rm/policy
        http://schemas.xmlsoap.org/ws/2005/02/rm/wsrmpolicy.xsd
        http://cxf.apache.org/ws/rm/manager http://cxf.apache.org/schemas/configuration/wsrmp-manager.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <cxf:bus>
        <cxf:features>
            <wsa:addressing/>
            <wsrm-mgr:reliableMessaging>
                <wsrm-policy:RMAssertion>
                    <wsrm-policy:BaseRetransmissionInterval Milliseconds="4000" />
                    <wsrm-policy:AcknowledgementInterval Milliseconds="2000" />
                </wsrm-policy:RMAssertion>
                <wsrm-mgr:destinationPolicy>
                    <wsrm-mgr:acksPolicy intraMessageThreshold="0" />
                </wsrm-mgr:destinationPolicy>
            </wsrm-mgr:reliableMessaging>
        </cxf:features>
    </cxf:bus>

</beans>
```

Second, you'll need to tell the CXF module to use this configuration file:

```
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:quartz="http://www.mulesoft.org/schema/mule/quartz"
    xmlns:cxf="http://www.mulesoft.org/schema/mule/cxf"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd
        http://www.mulesoft.org/schema/mule/cxf
        http://www.mulesoft.org/schema/mule/quartz
        http://www.mulesoft.org/schema/mule/quartz
        http://www.mulesoft.org/schema/mule/quartz/3.1/mule-quartz.xsd">

    <cxf:configuration configurationLocation="myConfig.xml" />
    ...
</mule>
```

To configure a JDBC transaction based store for persisting messaging, please see the [CXF instructions](#) for configuring the JDBC store.

Your Rating:  0 rates

Results:  0 rates

## Enabling WS-Security

### Enabling WS-Security

[ UsernameToken Scenario ] [ Server Configuration ] [ Client Configuration ] [ Configure the CXF Client to Use WS-Security ] [ Client Password Callback ] [ UsernameToken verification with the Mule SecurityManager ]

This page describes how to configure a client and service to use WS-Security. You should already have a basic client and server running. For a good primer on WS-Security, see [Understand WS-Security](#) on the Microsoft site. For further information on how to configure WS-Security with CXF, you should consult the [CXF documentation](#) which goes into more detail on how to configure the CXF service interceptors.

If you are using the enterprise edition of Mule, you can use the [SAML Module](#) to add SAML support to your configuration. See the [WS-Security Example](#) for an example of working with WS-Security with Mule Enterprise Edition.

#### *UsernameToken Scenario*

The UsernameToken feature in WS-Security is an interoperable way to exchange security tokens inside a SOAP message. The following section describes how to configure the client and server to exchange a username/password security token.

#### Server Configuration

On the server side, you do the following:

- Create a Mule service for your component implementation
- Configure the WSS4JInInterceptor and the SAAJInInterceptor. The former is responsible for checking the security of your message.
- Write a server PasswordCallback that verifies the password.

You configure the server in the Mule configuration file. Following is an example:

```
<flow name="greeterFlow">
    <http:inbound-endpoint address="http://localhost:63081/services/greeter" exchange-pattern="request-response"/>
    <cxft:jaxws-service serviceClass="org.apache.hello_world_soap_http.GreeterImpl">
        <cxft:inInterceptors>
            <spring:bean class="org.apache.cxf.binding.soap.saaj.SAAJInInterceptor" />
            <spring:bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
                <spring:constructor-arg>
                    <spring:map>
                        <spring:entry key="action" value="UsernameToken" />
                        <spring:entry key="passwordCallbackRef" value-ref="serverCallback" />
                    </spring:map>
                </spring:constructor-arg>
            </spring:bean>
        </cxft:inInterceptors>
    </cxft:jaxws-service>
    <http:inbound-endpoint>
        <component>
            <singleton-object class="org.apache.hello_world_soap_http.GreeterImpl" />
        </component>
    </http:inbound-endpoint>
</flow>
```

The `<cxft:inInterceptors>` element configures the incoming interceptors on the service. The WSS4JInInterceptor performs the security operations on the incoming SOAP message. The "action" parameter controls which actions it performs on the incoming message - in this case the "UsernameToken" action specifies that it will verify the username token via a specified password callback. The password callback is specified by the "passwordCallbackRef" property, which is detailed in the next section. The SAAJInInterceptor is also installed here. It enables the use of SAAJ, an in-memory DOM document format, which is required by WSS4J.

Server callbacks verify passwords by supplying the password with which the incoming password will be compared.

```

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class ServerPasswordCallback implements CallbackHandler
{

    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {

        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];

        if (pc.getIdentifier().equals("joe")) {
            // set the password on the callback. This will be compared to the
            // password which was sent from the client.
            pc.setPassword("password");
        }
    }
}

```

This allows you to write custom code which can load and compare passwords from custom backends, such as databases or LDAP.

### Client Configuration

On the client side, you do the following:

- Set up the CXF outbound endpoint
- Configure the CXF client so that it uses ws-security
- Set up a ClientPasswordCallback that supplies the password for the invocation

Following is a simple example that configures a CXF JAX-WS client message processor:

```

<jaxws-client
    name="clientEndpoint"
    clientClass="org.apache.hello_world_soap_http.SOAPService"
    port="SoapPort"
    wsdlLocation="classpath:/wsdl/hello_world.wsdl"
    operation="greetMe">
    <outInterceptors>
        <bean class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor" />
        <bean class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
            <constructor-arg>
                <map>
                    <entry key="action" value="UsernameToken" />
                    <entry key="user" value="joe" />
                    <entry key="passwordType" value="PasswordDigest" />
                    <!-- The callback supplies the password so its not stored in our config file -->
                    <entry key="passwordCallbackRef" value-ref="clientCallback" />
                </map>
            </constructor-arg>
        </bean>
    </outInterceptors>
</jaxws-client>

```

### Configure the CXF Client to Use WS-Security

To use WS-Security, you add a configuration section to your "my-cxf-config.xml" file.

**Note:** if your client and your server are on separate machines, you create two separate files and then a CXF connector configuration on each one.

```

<jaxws:client name="http://apache.org/hello\_world\_soap\_httpSoapPort" createdFromAPI="true">
    <jaxws:outInterceptors>
        <bean class="org.apache.cxf.binding.soap.saaj.SAAJOutInterceptor" />
        <bean class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
            <constructor-arg>
                <map>
                    <entry key="action" value="UsernameToken" />
                    <entry key="user" value="joe" />
                    <entry key="passwordType" value="PasswordDigest" />
                    <!-- The callback supplies the password so its not stored in our config file -->
                    <entry key="passwordCallbackRef" value-ref="clientCallback" />
                </map>
            </constructor-arg>
        </bean>
    </jaxws:outInterceptors>
</jaxws:client>

<bean id="clientCallback" class="org.mule.providers.soap.cxf.ClientPasswordCallback"/>

```

The above configuration specifies the following:

- CXF should invoke the UsernameToken action.
- The user name is "joe"
- Send the password in digest form.
- Use the "clientCallback" bean to supply the password. (see below)

#### **Client Password Callback**

Following is a simple example client password callback that sets the password to use for the outgoing invocation:

```

import java.io.IOException;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import org.apache.ws.security.WSPasswordCallback;

public class ClientPasswordCallback implements CallbackHandler
{
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];

        // set the password for our message.
        pc.setPassword("yourpassword");
    }
}

```

#### ***UsernameToken verification with the Mule SecurityManager***

If you're using the [Mule security manager](#), you can set up WSS4J to verify passwords with it. This allows you to easily integrate your own authentication mechanisms or use Mule's support for Spring Security.

First, you'll want to set up your security manager:

```

<mule-ss:security-manager>
    <mule-ss:delegate-security-provider name="memory-dao" delegate-ref="authenticationManager" />
</mule-ss:security-manager>

<spring:beans>
    <ss:authentication-manager alias="authenticationManager" />

    <ss:authentication-provider>
        <ss:user-service id="userService">
            <ss:user name="joe" password="password" authorities="ROLE_ADMIN" />
            <ss:user name="anon" password="anon" authorities="ROLE_ANON" />
        </ss:user-service>
    </ss:authentication-provider>

</spring:beans>

```

Next, you'll want to create a `<cxfrs:security-manager-callback>` element. This callback is responsible for bridging together the Mule security manager and WSS4J.

```

<spring:beans>
    ...
    <cxfrs:security-manager-callback id="serverCallback" />
</spring:beans>

```

Finally, you'll want to set up your server side WSS4J handlers to use this callback:

```

<cxf:jaxws-service>
    <cxf:inInterceptors>
        <spring:bean class="org.apache.cxf.binding.soap.saaj.SAAJInInterceptor" />
        <spring:bean class="org.apache.cxf.ws.security.wss4j.WSS4JInInterceptor">
            <spring:constructor-arg>
                <spring:map>
                    <spring:entry key="action" value="UsernameToken" />
                    <spring:entry key="passwordCallbackRef" value-ref="serverCallback" />
                </spring:map>
            </spring:constructor-arg>
        </spring:bean>
    </cxf:inInterceptors>
</cxf:jaxws-service>

```

In this example, the `CXF jaxws-service}` creates a `WSS4JInInterceptor` which performs `UsernameToken` verification of the message. Once it reads in the `username/password`, it will perform a callback to the Mule security manager using the `<cxfrs:security-manager-callback>`.



On the client side, you'll want to use plaintext passwords for this to work. To do this, set the "passwordType" property on the `WSS4JOutInterceptor` to "PasswordText".

Your Rating: Results: 0 rates

## Proxying Web Services with CXF

### Proxying Web Services with CXF

Normally when building CXF web services, you'll databind the XML to POJOs. A CXF component might receive an `OrderRequest` object, or you might send an `OrderRequest` object via a CXF outbound router. However, it is often useful to work with the XML directly when building web services or consuming other web services. The CXF module provides the ability to do this.

## **Deciding How to Proxy Your Web Service**

While many times you can proxy web services without using CXF (see [Proxying Web Services](#)), there are several cases where you might want to use CXF proxies:

- To work directly with the SOAP body, such as adding XML directly to it
- To take advantage of the CXF web service standards support to use WS-Security or WS-Addressing
- To enforce WS-Policy assertions
- To easily service a WSDL associated with your service
- To transform a SOAP body

You would use wsdl-cxf instead of proxies when you want to invoke a service and one or more of the following applies:

- You don't want to generate a client from WSDL
- You don't have the raw XML
- The service takes simple arguments such as string, int, long, or date

Note that currently CXF proxies only support working with the SOAP body. They do not send the whole SOAP message along.

### **Server-side Proxying**

To proxy a web service so you can work with the raw XML, you can create a CXF proxy message processor:

```
<cxn:proxy-service />
```

This will make the SOAP body available in the Mule message payload as an XMLStreamReader. To service a WSDL using a CXF proxy, you must specify the WSDL namespace as a property:

```
<cxn:proxy-service wsdlLocation="foo.wsdl"
    serviceName="YOUR_WSDL_SERVICE"
    namespace="YOUR_WSDL_NAMESPACE" />
```

### **Client-side Proxying**

Similarly, you can create an outbound endpoint to send raw XML payloads:

```
<cxn:proxy-client/>
```

Your Rating:  Results:  5 rates

## **Supported Web Service Standards**

### **Web Service Standards**

The CXF transport supports a variety of web service standards.

- SOAP 1.1 and 1.2
- WSDL 1.1
- WS Addressing 2004 and 1.0
- WS Security 1.1
- WS Policy
- WS Reliable Messaging (Feb 2005)
- WS SecureConversation
- WS SecurityPolicy
- WS Trust (client side only)
- WS-I BasicProfile 1.1
- WS-I SecurityProfile
- SAML 1.0 (with WS-Security)
- MTOM
- SOAP with attachments

Your Rating: 

Results:  1 rates

## Upgrading CXF from Mule 2

### Upgrading CXF from Mule 2

Mule 3 completely revamps the CXF support. CXF no longer exists as a transport, but rather a module which is a series of message processors.

This has the following benefits:

- Flexibility - it is now much easier to intermix CXF with transformers, filters and other message processors. You simply insert a CXF message processor in the flow where you wish it to be executed.
- Easier configuration - there are a number of configuration elements, e.g. jaxws-service, simple-service, proxy-service, etc, which each have their own syntax targeted to their specific use cases.
- Increased robustness - with the improved property scoping and message processor model, the CXF module property is more robust and rigorous in its behavior.

To learn more about the new syntax, see:

- [Building Web Services with CXF](#) - Information on how to build JAX-WS code first, JAX-WS WSDL first, and simple frontend services.
- [Consuming Web Services with CXF](#) - Information on how to consume JAX-WS services and simple frontend services.
- [Proxying Web Services with CXF](#) - Information on how to create web service proxies, e.g. if your CXF endpoint had proxy="true" before.

Your Rating: 

Results:  1 rates

## Using a Web Service Client Directly

### Using a Web Service Client Directly

This page describes how to use a standalone CXF-generated client with Mule. This approach is different from the [outbound router approach](#), which is typically a more appropriate fit for most applications.

There are two ways to use CXF clients. First, you can generate a client from WSDL using the CXF WSDL to Java tool. Second, if you've built your service via "code-first" methodologies, you can use the service interface to build a client proxy object.

When using a CXF client, it automatically discovers the Mule instance (provided you're in the same JVM/Classloader) and uses it for your transport. Therefore, if you've generated a client from WSDL, invoking a service over Mule can be as simple as the following:

```
HelloWorldService service = new HelloWorldService();
HelloWorld hello = service.getSoapPort();

// Possibly set an alternate request URL:
// ((BindingProvider) greeter).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
"http://localhost:63081/greeter");
String sayHi = hello.sayHi();
```

### Building a Client Proxy

Following is an example of how to construct a client using the service that was developed in [Building Web Services with CXF](#):

```

QName SERVICE_NAME = new QName("http://server.hw.demo/", "HelloWorld");
QName PORT_NAME = new QName("http://server.hw.demo/", "HelloWorldPort");

Service service = Service.create(SERVICE_NAME);

// Endpoint Address
String endpointAddress = http://localhost:63081/hello;

// Add a port to the Service
service.addPort(PORT_NAME, SOAPBinding.SOAP11HTTP_BINDING, endpointAddress);

HelloWorld hw = service.getPort(HelloWorld.class);

System.out.println(hw.sayHi("World"));

```

## Additional Resources

- Developing a JAX-WS consumer
- WSDL to Java
- CXF Maven plugin

Your Rating:  Results:  0 rates

## Using HTTP GET Requests

### Using HTTP GET Requests

CXF has built-in support for understanding GET requests, which use the following format:

`http://host/service/OPERATION/PARAM_NAME/PARAM_VALUE`

For example:

```

@WebService(endpointInterface = "org.mule.samples.echo.components.EchoService",
    serviceName = "echo")
public class EchoComponent implements EchoService
{
    public String echo(String string)
    {
        return string;
    }
}

```

The above Echo service is hosted in Mule on the endpoint `cxf:http://localhost:65081/services/EchoUMO`, so you can access the service from a simple web browser by typing the following:

`http://localhost:65081/services/EchoUMO/echo/string/Hello`

This will send the value "Hello" for the `string` parameter to the `echo()` method.



Due to a bug in CXF, this is only supported with the JAX-WS frontend.

Your Rating:  Results:  1 rates

## Using MTOM

### Using MTOM

[ Writing an MTOM-enabled WSDL ] [ Generating Server Stubs and/or a Client ] [ Configuring the CXF Inbound Endpoint ] [ Configuring the CXF client ]

This page describes the basics of how to use MTOM inside your service. For more information, go to the [MTOM documentation for CXF](#). Portions of the examples on this page are from that guide.

For a WSDL-first service, the general process is as follows:

1. Write your WSDL
2. Generate your server stubs
3. Configure the CXF endpoint in Mule

### **Writing an MTOM-enabled WSDL**

To get started, write a WSDL like you would normally. Use `xsd:base64Binary` schema types to represent any binary data that you want to send.

Normally, when creating XML schema types for binary data, you would write a schema type like this:

```
<element name="yourData" type="xsd:base64Binary" />
```

To tell CXF to treat this binary type as an attachment, you must add an `xmime:expectedContentTypes` attribute to it:

```
<schema targetNamespace="http://mediStor.org/types/"  
       xmlns="http://www.w3.org/2001/XMLSchema"  
       xmlns:xmime="http://www.w3.org/2005/05/xmlmime">  
  
<element name="yourData" type="xsd:base64Binary" xmime:expectedContentTypes=  
        "application/octet-stream"/>  
...  
</schema>
```

### **Generating Server Stubs and/or a Client**

After writing your WSDL, you run `wSDL2Java` on it to generate server stubs and a client (if you are creating outbound endpoints). To do this, download the CXF distribution, extract it, and then enter the following command:

```
$ ./apache-cxf-2.1.2-incubator/bin/wSDL2Java -d outputDirectory path/to/your.wsdl
```

### **Configuring the CXF Inbound Endpoint**

The above command generates a server stub interface. This is the interface that your service must implement. Each method will correspond to an operation in your WSDL. If the `base64Binary` type is an argument to one of your operations, CXF will generate a method like this:

```
public void yourOperation(DataHandler handler) {  
...  
}
```

You can use this `DataHandler` method to access the attachment data by calling `"handler.getInputStream();"`.

When you configure the CXF service, set the `mtomEnabled` attribute to true to enable MTOM:

```
<cxf:jaxws-service serviceClass="com.example.MtomService" mtomEnabled="true" />
```

### **Configuring the CXF client**

The configuration of an MTOM client is exactly like the configuration of a normal CXF outbound endpoint, except that you must specify the `mtomEnabled` attribute. For example:

```
<cxfr:jaxws-client  
    clientClass="org.mule.example.employee.EmployeeDirectory_Service"  
    operation="addEmployee"  
    wsdlPort="EmployeeDirectoryPort"  
    wsdlLocation="classpath:employeeDirectory.wsdl"  
    mtomEnabled="true"/>
```

Your Rating: 

Results:  0 rates

## Jersey Module Reference

### Jersey Module Reference

[ Classpath Settings ] [ Writing a service ] [ Deploy the Web Service ] [ Consume a RESTful Web Service ] [ JSON Support ] [ Exception Mappers ] [ Further help ]

Jersey is a JAX-RS (JSR-311) implementation. JAX-RS is a specification that provides a series of annotations and classes which make it possible to build RESTful services. The Mule Jersey transport makes it possible to deploy these annotated classes inside Mule 3.x.

In addition to the annotation capabilities, Jersey contains many useful features:

- The ability to integrate with XML data-binding frameworks such as JAXB
- The ability to produce/consume JSON easily
- The ability to integrate with the JSP presentation tier
- Integration with Abdera for Atom support.



Currently implicit views are not supported.

#### Classpath Settings

The latest Jersey module uses Jersey 1.3.1.

#### Writing a service

Writing JAX-RS services is an expansive topic and will not be covered in this guide. However, the Jersey website has an excellent set of samples, and the [JAX-RS specification](#) is helpful as well.

We will, however, take a look at a simple hello world service. This example requires the installation of Apache Xalan JARs.

The first step to create a JAX-RS service is to create a class which represents your HTTP resource. In our case we'll create a "HelloWorldResource" class. Methods on this class will be called in response to GET/POST/DELETE/PUT invocations on specific URLs.

The @Path annotation allows you to bind a class/resource to a specific URL. In the sample below we're binding the HelloWorldResource class to the "/helloworld" URL.

```

package org.mule.transport.jersey;

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

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

    @GET
    @Produces("text/plain")
    @Path("/{name}")
    public String sayHelloWithUri(@PathParam("name") String name) {
        return "Hello " + name;
    }
}

```

Looking at the "sayHelloWithUri" method we see several annotations involved:

- @GET specifies that this method is only called on @GET requests to the URL.
- @Produces specifies that this method is producing a resource with a mime type of "text/plain".
- @Path binds this method to the URL "/helloworld/{name}". The {name} is a URI template. Anything in this portion of the URL will be mapped to a URI parameter named "name" (see below)
- @PathParam binds the first parameter of the method to the URI parameter in that path named "name".

## Deploy the Web Service

Once you've written your service, you can create a `jersey:resources` component which contains a set of Jersey resources. URL. Below is a very simple configuration which does this:

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xmlns:jersey="http://www.mulesoft.org/schema/mule/jersey"
      xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/jersey
          http://www.mulesoft.org/schema/mule/jersey/3.0/mule-jersey.xsd
          http://jersey.apache.org/core http://jersey.apache.org/schemas/core.xsd">

    <flow name="HelloWorld">
        <inbound-endpoint address="http://localhost:8080/jersey"/>
        <jersey:resources>
            <component class="org.mule.transport.jersey.HelloWorldResource" />
        </jersey:resources>
    </flow>
</mule>

```

## Consume a RESTful Web Service

Once you run this configuration in Mule, you can hit the url: `http://localhost:8080/jersey/helloworld/Dan` and you should see this response in your browser: 'Hello Dan'

## JSON Support

If you want to use JSON, see [http://blogs.sun.com/enterprisetechtips/entry/configuring\\_json\\_for\\_restful\\_web](http://blogs.sun.com/enterprisetechtips/entry/configuring_json_for_restful_web). You need to add the jersey-json module (available on the [Jersey website](#)) and specify `@Produces/@Consumes("application/json")`.

## Exception Mappers

Starting with Mule 3.1.2 it is possible to register exception mappers inside the `resources` element. Exception mappers allow mapping generic exceptions that may be thrown in the component class to HTTP response codes.

The following configuration maps a `HelloWorldException` that may be thrown during the execution of `HelloWorldResource` to HTTP error 503 (Service Unavailable):

```
<jersey:resources>
    <component class="org.mule.module.jersey.HelloWorldResource" />
    <jersey:exception-mapper class="org.mule.module.jersey.exception.HelloWorldExceptionMapper" />
</jersey:resources>
```

### HelloWorldExceptionMapper.java

```
public class HelloWorldExceptionMapper implements ExceptionMapper<HelloWorldException>
{
    public Response toResponse(HelloWorldException exception)
    {
        int status = Response.Status.SERVICE_UNAVAILABLE.getStatusCode();
        return Response.status(status).entity(exception.getMessage()).type("text/plain").build();
    }
}
```



If you want to transform or send the request from your jersey component to next resource/flow then you need to use ContainerResponse cr = (ContainerResponse) message.getInvocationProperty("jersey\_response"); String messageString = (String) cr.getResponse().getEntity(); message.setPayload(messageString); This will convert org.mule.module.jersey.MuleResponseWriter\$1 type to String, which you can forward to your next resource.

## Further help

For more information on how to use Jersey, see the [project website](#).

Your Rating:

Results: 0 rates

## JSON Module Reference

### JSON Support Reference

JSON, short for JavaScript Object Notation, is a lightweight data interchange format. It is a text-based, human-readable format for representing simple data structures and associative arrays (called objects).

#### Object bindings

Mule support binding JSON data to objects and marshaling Java object to JSON using the [Jackson](#) Framework. Jackson uses annotations to describe how data is mapped to a Java object model. For example, lets say we have an JSON file that describes a person. When we receive that JSON data we want to convert it into a `Person` object. The JSON looks like this-

```
{
    "name": "John Doe",
    "dob": "01/01/1970",
    "emailAddresses": [
        {
            "type": "home",
            "address": "john.doe@gmail.com"
        },
        {
            "type": "work",
            "address": "jdoe@bigco.com"
        }
    ]
}
```

And we have an object `Person` we want to create from the JSON data. We use annotations to describe how to perform the mapping. We use the `@JsonAutoDetect` to say that field member names map directly to JSON field names -

```
@JsonAutoDetect
public class Person
{
    private String name;
    private String dob;

    private List<EmailAddress> emailAddresses;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getDob() { return dob; }

    public void setDob(String dob) { this.dob = dob; }

    public List<EmailAddress> getEmailAddresses() { return emailAddresses; }

    public void setEmailAddresses(List<EmailAddress> emailAddresses) { this.emailAddresses =
emailAddresses; }
}
```

The `EmailAddress` object that is used in the `emailAddresses` is just another JavaBean with the `@JsonAutoDetect` annotation.

At this point Mule can figure out whether to perform a JSON transforms based on the parameters of the method being called. For example -

```
public class PersonComponent {
    public void processPerson(@Payload Person person)
    {
        //tickle
    }
}
```

Here we would receive the contents of the `people.json` file above on an endpoint, Mule would see that `Person.class` is an annotated JSON object and that we had received JSON data from the JMS queue and perform the conversion.

### Global JSON Mapper

Jackson performs mappings through the `ObjectMapper`. This is an object that can be configured with other configuration about how to serialise data and define mixins that add annotations to objects that you cannot change directly. It is possible to define a global `ObjectMapper`; a single mapper that will be used for all JSON transforms in your application. This is not required since Mule will automatically create a mapper for a transformer, but using a global mapper can be useful if you need to configure specific properties on the mapper or use mixins. To create a shared `ObjectMapper` Add the following to your Mule configuration file -

```

<json:mapper name="myMapper">
    <json:mixin mixinClass="org.mule.module.json.transformers.FruitCollectionMixin"
        targetClass="org.mule.module.json.transformers.FruitCollection"/>
    <json:mixin mixinClass="org.mule.module.json.transformers.AppleMixin"
        targetClass="org.mule.tck.testmodels.fruit.Apple"/>
</json:mapper>

```

### Intercepting JSON Transforms

So far we have discussed how Mule will perform automatic JSON transforms. Sometimes you may want to intercept the transform, to do this just create a transformer with a method return or parameter type of your JSON class -

```

@Transformer(sourceTypes = {InputStream.class})
public Person toPerson(String json, ObjectMapper mapper) throws JAXBException
{
    return (Person)mapper.readValue(in, Person.class);
}

```

The `ObjectMapper` instance will either be created for you or the global context for your application will be used. One reason for doing this would be to strip out some JSON elements and create objects from a subset of the JSON received. For more information about transforms see the [Using Transformers](#) section.

### JsonPath

There is no standard language currently for querying JSON data graphs in the same way XPATH can query XML documents. Mule has created a simple query syntax for working with JSON data in Java, called JsonPath.

This query syntax provides a simple way to navigate a JSON data structure. The following JSON data will be used to demonstrate how JSON Path queries can be used.

```
{
    "name": "John Doe",
    "dob": "01/01/1970",
    "emailAddresses": [
        {
            "type": "home",
            "address": "john.doe@gmail.com"
        },
        {
            "type": "work",
            "address": "jdoe@bigco.com"
        }
    ]
}
```

To select a child entry use:

```
name
```

To access array data, use square braces with an index value i.e.

```
emailAddresses[0]/type
```

Or where the route element is an array:

```
[0]/arrayElement
```

Also, multi-dimensional arrays can be accessed using:

```
filters[1]/init[1][0]
```

This is rare, but if a Json property name contains a '/' the name needs to be single quoted i.e.

```
results/'http://foo.com'/value
```

### JsonPath in Expressions

JSON Path can be used in [Mule expressions](#) to query JSON message payloads for filtering or enrichment.

For example, to use JSON Path to perform content based routing:

```
<choice>
  <when expression="emailAddresses[0]/type = 'home'" evaluator="json">
    <append-string-transformer message="Home address is #[json:emailAddresses[0]/address]" />
  </when>
  <when expression="emailAddresses[0]/type = 'work'" evaluator="json">
    <append-string-transformer message="Work address is #[json:emailAddresses[0]/address]" />
  </when>
  <otherwise>
    <append-string-transformer message=" No email address found" />
  </otherwise>
</choice>
```

The expression evaluator name is 'json', the expression is any valid JSON Path expression. Note that when doing boolean expressions such as in the example above, a few operators are supported:

Operator	Example
=	emailAddresses0/type = 'foo' or emailAddresses0/flag = true
!=	emailAddresses0/type != null or emailAddresses0/flag != false

String comparisons need to be in single quotes, 'null' is recognised as null, and boolean comparisons are supported. If checking numeric values just treat them as a string.

### Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Transport ([schemadoc:page-title not set](#))

The JSON module contains a number of tools to help you read, transform, and write JSON.

### Transformers

These are transformers specific to this transport. Note that these are added automatically to the Mule registry at start up. When doing automatic transformations these will be included when searching for the correct transformers.

Name	Description

json-to-object-transformer	A transformer that will convert a JSON encoded object graph to a java object. The object type is determined by the 'returnClass' attribute. Note that this transformers supports Arrays and Lists. For example, to convert a JSON string to an array of org.foo.Person, set the <code>returnClass=org.foo.Person[]</code> . The JSON engine can be configured using the <code>jsonConfig</code> attribute. This is an object reference to an instance of: <code>net.sf.json.JsonConfig</code> . This can be created as a spring bean.
object-to-json-transformer	Converts a java object to a JSON encoded object that can be consumed by other languages such as Javascript or Ruby. The JSON Object mapper can be configured using the <code>mapper-ref</code> attribute. This is an object reference to an instance of: <code>org.codehaus.jackson.Mapper</code> . This can be created as a spring bean. Usually the default mapper is sufficient. Often users will want to configure exclusions or inclusions when serializing objects. This can be done by using the Jackson annotations directly on the object (see <a href="http://jackson.codehaus.org/1.3.0/javadoc/org/codehaus/jackson/annotate/package-frame.html">http://jackson.codehaus.org/1.3.0/javadoc/org/codehaus/jackson/annotate/package-frame.html</a> ) If it is not possible to annotate the object directly, mixins can be used to add annotations to an object using AOP. There is a good description of this method here: <a href="http://www.cowtowncoder.com/blog/archives/08-01-2009_08-31-2009.html">http://www.cowtowncoder.com/blog/archives/08-01-2009_08-31-2009.html</a> . To configure mixins for your objects, either configure the <code>mapper-ref</code> attribute or register them with the transformer using the <code>&lt;serialization-mixin&gt;</code> element. The <code>returnClass</code> for this transformer is usually <code>java.lang.String</code> , <code>byte[]</code> can also be used. At this time the transformer does not support streaming.

## Filters

Filters can be used on inbound endpoints to control which data is received by a service.

Name	Description
is-json-filter	A filter that will determine if the current message payload is a JSON encoded message.

## Mapper

The Jackson mapper to use with a JSON transformer. This isn't required but can be used to configure mixins on the mapper.

### Attributes of <mapper...>

Name	Type	Required	Default	Description
name	string	no		The name of the mapper that is used to make a reference to it by the transformer elements.

### Child Elements of <mapper...>

Name	Cardinality	Description
mixin	0..1	

Your Rating:  Results:  0 rates

## Acegi Module Reference

### Acegi Module Reference

[ [Introduction](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Configuration Reference](#) ] [ [Acegi Module](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ] [ [Extending this Module or Transport](#) ] [ [Best Practices](#) ] [ [Notes](#) ]

Your Rating:  Results:  1 rates



NOTE: Acegi is replaced by Spring Security and deprecated. It may not be supported in future versions of Mule. This configuration reference information is provided with the understanding that if you are currently using the Mule Acegi module, you will eventually want to [upgrade from Acegi to Spring Security](#).

## Introduction

Acegi provides a number of authentication and authorization providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO. The Mule Acegi security manager implementation delegates to [Acegi](#) to provide authorization and authentication functions.

## Namespace and Syntax

XML namespace:

```
xmlns: acegi "http://www.mulesoft.org/schema/mule/acegi"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/acegi
```

## Considerations

This module is documented for reference and to support existing implementations, but should not be used.

## Features

The Mule Acegi module enables you to hook your Mule implementation up to a number of standard security schemes. This Mule module was designed to allow you to hook into security providers, authentication and authorization services, and it is powerful enough to allow you to configure method-level authorization on components, meaning that users with different roles can only invoke certain service methods.

## Usage

The Mule Acegi module is deprecated.

### ***Securing Service Components***

To secure MethodInvocations, developers must add a properly configured `MethodSecurityInterceptor` into the application context. The beans requiring security are chained into the interceptor. This chaining is accomplished using Spring's `ProxyFactoryBean` or `BeanNameAutoProxyCreator`. Alternatively, Acegi security provides a `MethodDefinitionSourceAdvisor`, which you can use with Spring's `DefaultAdvisorAutoProxyCreator` to automatically chain the security interceptor in front of any beans defined against the `MethodSecurityInterceptor`.

In addition to the `daoAuthenticationProvider` and `inMemoryDaoImpl` beans (see [Configuring Security](#)), the following beans must be configured:

- `MethodSecurityInterceptor`
- `AuthenticationManager`
- `AccessDecisionManager`
- `AutoProxyCreator`
- `RoleVoter`

### ***The MethodSecurityInterceptor***

The `MethodSecurityInterceptor` is configured with a reference to an:

- `AuthenticationManager`
- `AccessDecisionManager`

Following is a security interceptor for intercepting calls made to the methods of a component that has an interface `myComponentIfc`, which defines two methods: `delete` and `writeSomething`. Roles are set on these methods as seen below in the property `objectDefinitionSource`.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:acegi="http://www.mulesoft.org/schema/mule/acegi"
      xmlns:https="http://www.mulesoft.org/schema/mule/https"
      xmlns:http="http://www.mulesoft.org/schema/mule/http"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/acegi
          http://www.mulesoft.org/schema/mule/acegi/3.1/mule-acegi.xsd
          http://www.mulesoft.org/schema/mule/https
          http://www.mulesoft.org/schema/mule/https/3.1/mule-https.xsd
          http://www.mulesoft.org/schema/mule/http
          http://www.mulesoft.org/schema/mule/http/3.1/mule-http.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">
    ...
    <bean id="myComponentSecurity" class=
"org.acegisecurity.intercept.method.aopalliance.MethodSecurityInterceptor">
        <property name="authenticationManager" ref="authenticationManager"/>
        <property name="accessDecisionManager" ref="accessDecisionManager"/>
        <property name="objectDefinitionSource">
            <value>
                com.foo.myComponentIfc.delete=ROLE_ADMIN
                com.foo.myComponentIfc.writeSomething=ROLE_ANONYMOUS
            </value>
        </property>
    </bean>
</mule>

```

### The AuthenticationManager

An AuthenticationManager is responsible for passing requests through a chain of AuthenticationProvider objects.

```

<bean id="authenticationManager" class='org.acegisecurity.providers.ProviderManager'>
    <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider"/>
        </list>
    </property>
</bean>

```

### The AccessDecisionManager

This bean specifies that a user can access the protected methods if they have any one of the roles specified in the objectDefinitionSource.

```

<bean id="accessDecisionManager" class='org.acegisecurity.vote.AffirmativeBased'>
    <property name="decisionVoters">
        <list>
            <ref bean="roleVoter"/>
        </list>
    </property>
</bean>

```

### The AutoProxyCreator

This bean defines a proxy for the protected bean. When an application asks Spring for a `myComponent` bean, it will get this proxy instead.

```

<bean id="autoProxyCreator" class=
"org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
    <property name="interceptorNames">
        <list>
            <value>myComponentSecurity</value>
        </list>
    </property>
    <property name="beanNames">
        <list>
            <value>myComponent</value>
        </list>
    </property>
    <property name='proxyTargetClass' value="true" />
</bean>

```

When using BeanNameAutoProxyCreator to create the required proxy for security, the configuration must contain the property proxyTargetClass set to true. Otherwise, the method passed to MethodSecurityInterceptor.invoke is the proxy's caller, not the proxy's target.

### The RoleVoter

The RoleVoter class will vote if any ConfigAttribute begins with ROLE\_. The RoleVoter is case sensitive on comparisons as well as the ROLE\_ prefix.

- It will vote to grant access if there is a GrantedAuthority, which returns a String representation (via the getAuthority( ) method) exactly equal to one or more ConfigAttributes starting with ROLE\_.
- If there is no exact match of any ConfigAttribute starting with ROLE\_, the RoleVoter will vote to deny access.
- If no ConfigAttribute begins with ROLE\_, the voter will abstain.

```
<bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter"/>
```

### Setting Security Properties on the Security Provider

You can add any additional properties to the security provider in the securityProperties map. For example, this map can be used to change Acegi's default security strategy into one of the following:

- MODE\_THREADLOCAL, which allows the authentication to be set on the current thread (this is the default strategy used by Acegi)
- MODE\_INHERITABLETHREADLOCAL, which allows authentication to be inherited from the parent thread
- MODE\_GLOBAL, which allows the authentication to be set on all threads

### Securing Components in Asynchronous Systems

Acegi security strategies are particularly useful with an asynchronous system, since we have to add a property on the security provider for the authentication to be set on more than one thread.

In this case, we would use MODE\_GLOBAL as seen in the example below.

```

<acegi:security-manager>
    <acegi:delegate-security-provider name="memory-dao" delegate-ref="daoAuthenticationProvider">
        <acegi:security-property name="securityMode" value="MODE_GLOBAL" />
    </acegi:delegate-security-provider>
</acegi:security-manager>

```

### Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

## Acegi Module

Acegi provides a number of authentication and authorization providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and

DAO. The Mule Acegi security manager implementation delegates to Acegi to provide authorization and authentication functions.

## **Security manager**

### **Attributes of <security-manager...>**

Name	Type	Required	Default	Description
id		no		
name		no		

### **Child Elements of <security-manager...>**

Name	Cardinality	Description
delegate-security-provider	0..1	An Acegi-based security provider that delegates authorization to some other provider.

## **Delegate security provider**

An Acegi-based security provider that delegates authorization to some other provider.

### **Attributes of <delegate-security-provider...>**

Name	Type	Required	Default	Description
name	name (no spaces)	yes		
delegate-ref	string	yes		

### **Child Elements of <delegate-security-provider...>**

Name	Cardinality	Description
security-property	0..*	

## **Http security filter**

This appears to authenticate users via information in standard HTTP headers.

### **Attributes of <http-security-filter...>**

Name	Type	Required	Default	Description
realm	string	yes		
securityProviders	string	no		The delegate-security-provider to use for authenticating. Use this element in case you have multiple security managers defined in your configuration.

### **Child Elements of <http-security-filter...>**

Name	Cardinality	Description

## **Schema**

The Schema reference doc is below:

Namespace "http://www.mulesoft.org/schema/mule/acegi"

Targeting Schemas (1):

mule-acegi.xsd

Targeting Components:

3 global elements, 1 local element, 4 complexTypes

Schema Summary	
mule-acegi.xsd	<p>Acegi provides a number of authentication and authorization providers such as JAAS, LDAP, CAS (Yale Central Authentication service), and DAO.</p> <p>Target Namespace:</p> <p style="padding-left: 20px;"><a href="http://www.mulesoft.org/schema/mule/acegi">http://www.mulesoft.org/schema/mule/acegi</a></p> <p>Defined Components:</p> <p style="padding-left: 20px;">3 global elements, 1 local element, 4 complexTypes</p> <p>Default Namespace-Qualified Form:</p> <p style="padding-left: 20px;">Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:</p> <p style="padding-left: 20px;"><a href="http://www.mulesoft.org/schema/mule/acegi/3.1/mule-acegi.xsd">http://www.mulesoft.org/schema/mule/acegi/3.1/mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Imports Schemas (3):</p> <p style="padding-left: 20px;"><a href="#">mule-schemadoc.xsd</a>, <a href="#">mule.xsd</a>, <a href="#">xml.xsd</a></p>
All Element Summary	
delegate-security-provider	<p>An Acegi-based security provider that delegates authorization to some other provider.</p> <p>Type: <a href="#">delegateSecurityProviderType</a></p> <p>Content: complex, 2 attributes, 1 element</p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Used: at 1 location</p>
http-security-filter	<p>This appears to authenticate users via information in standard HTTP headers.</p> <p>Type: <a href="#">httpSecurityFilterType</a></p> <p>Content: empty, 2 attributes, attr. wildcard</p> <p>Subst.Gr:may substitute for elements: <a href="#">mule:abstract-security-filter</a>, <a href="#">mule:abstract-message-processor</a></p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
security-manager	<p>Type: <a href="#">securityManagerType</a></p> <p>Content: complex, 2 attributes, 1 element</p> <p>Subst.Gr:may substitute for element <a href="#">mule:abstract-security-manager</a></p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Used: never</p>
security-property	<p>Type: <a href="#">securityProperty</a></p> <p>Content: empty, 2 attributes</p> <p>Defined: locally within <a href="#">delegateSecurityProviderType</a> complexType in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p>
Complex Type Summary	
delegateSecurityProviderType	<p>Content: complex, 2 attributes, 1 element</p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Includes: definitions of 1 attribute, 1 element</p> <p>Used: at 1 location</p>
httpSecurityFilterType	<p>Content: empty, 2 attributes, attr. wildcard</p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Includes: definitions of 2 attributes</p> <p>Used: at 1 location</p>
securityManagerType	<p>This is the security provider type that is used to configure Acegi related functionality.</p> <p>Content: complex, 2 attributes, 1 element</p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Includes: definition of 1 element</p> <p>Used: at 1 location</p>
securityProperty	<p>Content: empty, 2 attributes</p> <p>Defined: globally in <a href="#">mule-acegi.xsd</a>; see <a href="#">XML source</a></p> <p>Includes: definitions of 2 attributes</p> <p>Used: at 1 location</p>

## Javadoc API Reference

The Javadoc for this transport can be found here: [ACEGI](#).

## Maven

The Acegi Module can be included with the following dependency:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-acegi</artifactId>
</dependency>
```

## Extending this Module or Transport

Not applicable.

## Best Practices

See Notes below.

## Notes



NOTE: Acegi is replaced by Spring Security and deprecated. It may not be supported in future versions of Mule. This configuration reference information is provided with the understanding that if you are currently using the Mule Acegi module, you will eventually want to [upgrade from Acegi to Spring Security](#).

Your Rating:

Results: 1 rates

## JAAS Module Reference

### JAAS Module Reference

This module provides security via JAAS.

cache: Unexpected program error: java.lang.NullPointerException

#### Security Manager

This is the security provider type that is used to configure JAAS related functionality.

#### *Child Elements of <security-manager...>*

Name	Cardinality	Description
security-provider	0..1	This is the security provider type that is used to configure JAAS related functionality.
password-encryption-strategy	0..*	

#### Security Provider

This is the security provider type that is used to configure JAAS related functionality.

#### *Attributes of <security-provider...>*

Name	Type	Required	Default	Description
loginContextName	string	no		

credentials	string	no	
loginConfig	string	no	
loginModule	string	no	

## Jaas Security Filter

Authenticates users via JAAS.

Your Rating:  Results:  0 rates

## JBoss Transaction Manager Reference

cache: Unexpected program error: java.lang.NullPointerException

### JBoss Transaction Manager

This module enables Mule to use the JBoss transaction manager (previously Arjuna) to configure **X A transactions**. Developers can configure one Transaction Manager per Mule instance. For more information, see [JBoss Transactions](#).

#### Transaction Manager

To configure an instance of the JBoss transaction manager within Mule, add this element to your Mule XML config file. You can configure arbitrary properties on the transaction manager that will be passed on to the underlying transaction manager. For example:

```
<jbossts:transaction-manager>
  <property key="test" value="TEST" />
</jbossts:transaction-manager>
```

You can then declare XA transactions on endpoints supporting XA transactions, and all those transactions will be managed by the JBoss transaction manager.

#### Additional Properties for the JBoss Transaction Manager

You can configure many properties for the JBoss Transaction Manager. These configurable properties are detailed [in the Environment class](#).

The following is an example that demonstrates how to configure properties related to transaction timeout.

#### Configuring Transaction Timeout Properties

There are two properties that relate to timeouts for JBoss transactions. (Note that the information for timeout properties comes from [here](#)).

- com.arjuna.ats.arjuna.coordinator.defaultTimeout
  - This defaultTimeout property defines the timeout period for each transaction created for this manager. Transactions that have not terminated before the timeout value expires are automatically rolled back. The default value is 60 seconds.
- com.arjuna.ats.arjuna.coordinator.txReaperTimeout
  - JBoss TS uses a separate reaper thread that monitors all locally created transactions and forces them to roll back if their timeouts elapse. To prevent this reaper thread from consuming application time, it only runs periodically. By default, the reaper thread is set to monitor locally transactions every 120000 milliseconds. You can override this default value by setting the txReaperTimeout property. Note that if the defaultTimeout property value is less than the txReaperTimeout property value, the reaper thread checks for transactions once every defaultTimeout seconds.

```
<jbossts:transaction-manager >
  <property key="com.arjuna.ats.arjuna.coordinator.defaultTimeout" value="47" /><!-- this is in
seconds -->
  <property key="com.arjuna.ats.arjuna.coordinator.txReaperTimeout" value="108000"/><!-- this is in
milliseconds -->
</jbossts:transaction-manager>
```

Please note that the `timeout` parameter in the `xa-transaction` tag, shown in the code sample below, is ignored for JBoss transactions because these transactions are configured globally in the `jbossts:transaction-manager`. Note, too, that this code is shown with a strike-through line to indicate that you should **ignore** this parameter.

```
<xa-transaction action="ALWAYS_BEGIN" timeout="60000" />
```

Your Rating: ★★★★★ Results: ★★★★★ 0 rates

## Scripting Module Reference

### Scripting Module Reference

[ Script Configuration Builder ] [ Component ] [ Script ] [ Script Context Bindings ] [ Groovy Refreshable ] [ Lang ]

The scripting module provides facilities for using scripting languages in Mule. Any scripting languages that supports JSR-223 can be used inside Mule. Scripts can be used as implementations of service components or transformers. Also, scripts can be used for expression evaluations, meaning message routing can be controlled using script evaluations on the current message. You can even configure Mule instances from scripts.

#### Script Configuration Builder

The `ScriptConfigurationBuilder` allows developers to create a Mule instance from a JSR-223 compliant script. To load the manager from Groovy:

```
ConfigurationBuilder builder = new ScriptConfigurationBuilder("groovy", ".../conf/mule-config.groovy");
MuleContext muleContext = new DefaultMuleContextFactory().createMuleContext(builder);
```

Or to start the server from the command line:

```
mule -M-Dorg.mule.script.engine=groovy
-builder org.mule.module.scripting.builders.ScriptConfigurationBuilder
-config ../conf/mule-config.groovy
```

For more information about configuring a Mule instance from code or script see Starting Mule with the Configuration.

cache: Unexpected program error: java.lang.NullPointerException

#### Component

Defines a script component backed by a JSR-223 compliant script engine such as Groovy, JavaScript, or Ruby. Scripting allows you to either directly embed your script inside the XML config or reference a script using Spring's dynamic language support:  
<http://static.springframework.org/spring/docs/2.5.x/reference/dynamic-language.html>.

#### Attributes of <component...>

Name	Type	Required	Default	Description
script-ref	string	no		A reference to a script object bean, that is, a <script:script ...> definition.

#### Child Elements of <component...>

Name	Cardinality	Description
script	0..1	A script to be executed by a JSR-223 compliant script engine such as Groovy, JavaScript(Rhino), Python, Ruby, or Beanshell.
java-interface-binding	0..*	A binding associates a Mule endpoint with an injected Java interface (this is like using Spring to inject a bean, but instead of calling a method on the bean a message is sent to an endpoint). Script bindings will only work with Java-based scripting languages. Right now there is no validation on when languages support Java bindings because there are so many scripting languages.

The following example demonstrates how to configure a Groovy script component with an in-line script:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:script="http://www.mulesoft.org/schema/mule/scripting"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/scripting
          http://www.mulesoft.org/schema/mule/scripting/3.0/mule-scripting.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd">

    <vm:connector name="vmConnector"/>

    <script:script name="myScript" engine="groovy">
        return "$payload Received"
    </script:script>

    <model>
        <service name="inlineScript">
            <inbound>
                <inbound-endpoint address="vm://in1"/>
            </inbound>
            <script:component>
                <script:script engine="groovy">
                    return "$payload Received"
                </script:script>
            </script:component>
            <outbound>
                <pass-through-router>
                    <outbound-endpoint address="vm://out1"/>
                </pass-through-router>
            </outbound>
        </service>
    ...

```

The following example demonstrates how to orchestrate message flows using bindings. The example calls out to two different services and passes the results on to the outbound router.

```

<service name="scriptWithBindings">
    <inbound>
        <inbound-endpoint ref="client_request"/>
    </inbound>
    <script:component>
        <script:script engine="groovy">
            msg = CalloutService.doSomething(payload)
            return CalloutService.doSomethingElse(msg)
        </script:script>
        <script:java-interface-binding interface="org.mule.components.script.CalloutService"
method="doSomething">
            <outbound-endpoint ref="callout_1" synchronous="true"/>
        </script:java-interface-binding>
        <script:java-interface-binding interface="org.mule.components.script.CalloutService"
method="doSomethingElse">
            <outbound-endpoint ref="callout_2" synchronous="true"/>
        </script:java-interface-binding>
    </script:component>
    <outbound>
        <pass-through-router>
            <outbound-endpoint ref="client_response"/>
        </pass-through-router>
    </outbound>
</service>

<service name="Callout1">
    <inbound>
        <inbound-endpoint ref="callout_1"/>
    </inbound>
    <test:component appendString=" Received by #[mule:context.serviceName]"/>
</service>

<service name="Callout2">
    <inbound>
        <inbound-endpoint ref="callout_2"/>
    </inbound>
    <test:component appendString=" Received by #[mule:context.serviceName]"/>
</service>

```

cache: Unexpected program error: java.lang.NullPointerException

## Script

Represents a script that can be used as a component for a service or a transformer. The script text can be pulled in from a script file or can be embedded inside this element. A script can be executed by any JSR-223 compliant script engine such as Groovy, JavaScript(Rhino), Python, Ruby, or Beanshell.

### Attributes of <script...>

Name	Type	Required	Default	Description
name	string	no		The name used to identify this script object. This is used when you want to reference this script object from a component or transformer.
engine	string	no		The name of the script engine being used. All scripting languages that support JSR-223 have a script engine name such as groovy, ruby, python, etc. If this value is not set, but a script file is configured, Mule will attempt to load the correct script engine according to the script file's extension.
file	string	no		The script file to load for this object. The file can be on the classpath or local file system.

### Child Elements of <script...>

Name	Cardinality	Description
text	0..1	Used for embedding script code inside the XML. This is useful for simple scripts where you are just mocking up a quick application.

## Script Context Bindings

When run inside Mule, scripts have a number of objects available to them in the script context:

Name	Description
log	A logger that can be used to write to Mule's log file.
muleContext	A reference to the MuleContext object.
registry	A convenience shortcut to Mule registry (otherwise available via <code>muleContext.registry</code> ).
eventContext	A reference to the event context. This allows you to dispatch events programmatically from your script.
message	The current message.
originalPayload	The payload of the current message before any transforms.
payload	The transformed payload of the current message if a transformer is configured on the service. Otherwise this is the same value as <code>originalPayload</code> .
src	Same as <code>payload</code> , kept for backward compatibility.
service	A reference to the current service object.
id	The current message ID.
result	A placeholder object where the result of the script can be written. Usually it's better to just return a value from the script unless the script method doesn't have a return value.
	 If your script needs to return null, you must set <code>result=null</code> instead of simply returning null
message properties	Any message properties can be used as variables for the script.

cache: Unexpected program error: java.lang.NullPointerException

To use Groovy as an example, the following transformer configuration will convert a comma-separated string of values to a `java.util.List`.

```
<script:transformer name="stringReplaceWithParams">
    <script:script engine="groovy">
        <property key="oldStr" value="l"/>
        <property key="newStr" value="x"/>
        <script:text>
            return payload.toString().replaceAll("$oldStr", "$newStr")
        </script:text>
    </script:script>
</script:transformer>
```

cache: Unexpected program error: java.lang.NullPointerException

## Groovy Refreshable

A wrapper for a component object that allows the underlying object to be reloaded at runtime. This makes it possible to hot-deploy new component logic without restarting.

### Attributes of `<groovy-refreshable...>`

Name	Type	Required	Default	Description
name	string	no		The name for this refreshable groovy bean wrapper.
refreshableBean-ref	string	no		The reference to a <code>groovy.lang.Groovy</code> object to use for this component.
methodName	string	no		The entrypoint method to invoke when a message is received for the object.

cache: Unexpected program error: java.lang.NullPointerException

## Lang

This element allows the <http://www.springframework.org/schema/lang> namespace to be embedded. Within this element developers can include the Spring lang namespace.

Your Rating:  0 rates

Results:  0 rates

## Spring Extras Module Reference

### Spring Extras Module

This module provides extensions for using the Spring framework with Mule, such as using the Spring container to build components managed by Mule.

Package	Description
org.mule.module.spring.events	A Spring EventMulticaster that allows any Spring bean to send and receive Mule events through the ApplicationContext and event listeners.
org.mule.module.spring.i18n	Spring messages that can be localized.
org.mule.module.spring.remoting	Classes for using Spring remoting. For more information, see the <a href="#">Spring Remoting example</a> .
org.mule.module.spring.transaction	Provides classes for a transaction factory and transaction manager factory.

Your Rating:  0 rates

Results:  0 rates

## SXC Module Reference

### SXC Module Reference

The SXC module contains an outbound router and a filter that use the SXC project for streaming XPath routing.

SXC allows listening for XPath expressions as the document is being parsed. As soon as an expression is found, an event is fired, and parsing is stopped. This allows for much more efficient XPath evaluation. XPath evaluators such as Jaxen work with a DOM model, so even when working with lazy-loading DOMs, such as AXIOM, there is more overhead than in just reading directly off the XML stream.

SXC supports a limited subset of XPath expressions. For details, see the [SXC documentation](#). To request support for a missing XPath feature, please file a [SXC JIRA](#).

### Using the SXC Outbound Router and Filter

SXC requires a special filtering outbound router, inside of which you configure the SXC filter and any other filters that do not work on the XML payload itself (such as AndFilter, OrFilter, and MessagePropertyFilter). For example, this configuration routes a message based on an XML attribute:

```
<sxc:filtering-router>
    <outbound-endpoint address="vm://log" />
    <sxc:filter pattern="//purchaseOrder[@country]" />
    <sxc:namespace prefix="test" uri="http://foo" />
</sxc:filtering-router>
```

Following is another example of a filter that looks for messages where the billing address is within the United States:

```
<sxc:filtering-router>
    ...
    <sxc:filter pattern="/customer/billingaddress/country[text() = 'US']" />
    ...
</sxc:filtering-router>
```

## XML Module Reference

### XML Module Reference

[ XML Formats ] [ Transformers ] [ Filters ] [ Splitters ] [ XML Parsers ]

The XML module contains several tools to help you read, transform, and write XML.

In addition to the functionality described on this page, you can also use the [SXC Module Reference](#), which enables efficient XPath XML routing.

#### XML Formats

Mule understands a wide variety of XML Java representations:

- org.w3c.dom.Document, org.w3c.dom.Element
- org.dom4j.Document
- javax.xml.transform.Source
- InputStream, String, byte[]
- OutputHandler
- XMLStreamReader
- org.mule.module.xml.transformer.DelayedResult

Any transformer that accepts XML as an input will also understand these types.

#### Transformers

There are several standard transformers that process XML inside Mule.

Transformer	Description
XmToObject <-> ObjectToXml	Converts XML to a Java object and back again using <a href="#">XStream</a> .
JAXB XmToObject <-> JAXB ObjectToXml	Converts XML to a Java object and back again using the <a href="#">JAXB</a> binding framework (ships with JDK6)
XSLT	Transforms XML payloads using XSLT.
XQuery	Transforms XML payloads using <a href="#">XQuery</a> .
DomToXml <-> XmlToDom	Converts DOM objects to XML and back again.
XmlToXMLStreamReader	Converts XML from a message payload to a StAX XMLStreamReader.
XPath Extractor	Queries and extracts object graphs using XPath expressions using JAXP.
JXPath Extractor	Queries and extracts object graphs using XPath expressions using JXPath.
XmPrettyPrinter	Allows you to output the XML with controlled formatting, including trimming white space and specifying the indent.

#### ***Efficient Transformations with DelayedResult***

Mule contains a special XML output format called `DelayedResult`. This format allows very efficient XML transformations by delaying any XML serialization until an `OutputStream` is available.

For example, here is an XSLT transformer set up to use `DelayedResult`:

```
<mxml:xslt-transformer name="transform-in"
                        xsl-file="xslt/transform.xslt"
                        returnClass="org.mule.module.xml.transformer.DelayedResult"/>
```

If the result of this transformation were being sent to an HTTP client, the HTTP client would ask Mule for an `OutputHandler` and pass in the `OutputStream` to it. Only then would Mule perform the transformation, writing the output directly to the `OutputStream`.

If `DelayedResult` were not used, the XML result would first be written to an in-memory buffer before being written to the `OutputStream`. This will cause your XML processing to be slower.

## Filters

The XML module contains various XPath filters. For general details on how to use filters, see [Using Filters](#).

### XPath Filter

The XPath filter uses the JAXP libraries to filter XPath expressions. This filter is available as of Mule 2.2.

The following configuration routes messages to the "vm://echo" endpoint when the value of "/e:purchaseOrder/e:shipTo/@country" is "US".

```
<outbound>
  <filtering-router>
    <outbound-endpoint address="vm://echo" synchronous="true"/>
    <mule-xml>xpath-filter pattern="/e:purchaseOrder/e:shipTo/@country" expectedValue="US">
      <mule-xml:namespace prefix="e" uri="http://www.example.com"/>
    </mule-xml:jxpath-filter>
  </filtering-router>
  ...
</outbound>
```

### Schema Validation Filter

The schema validation filter uses the JAXP libraries to validate your message against a schema. This filter is available as of Mule 2.2.

The following configuration will validate your message against a schema called `schema.xsd` and a schema called `anotherSchema.xsd`.

```
<mule-xml:schema-validation-filter schemaLocations="com/myapp/schemas/schema.xsd,
com/myapp/schemas/anotherSchema.xsd" />
```

### Jaxen Filter

The Jaxen filter uses the Jaxen library to filter messages based on XPath expressions.

The following configuration routes messages to the "vm://echo" endpoint when the value of "/e:purchaseOrder/e:shipTo/@country" is "US".

```
<outbound>
  <filtering-router>
    <outbound-endpoint address="vm://echo" synchronous="true"/>
    <mule-xml:jaxen-filter pattern="/e:purchaseOrder/e:shipTo/@country" expectedValue="US">
      <mule-xml:namespace prefix="e" uri="http://www.example.com"/>
    </mule-xml:jaxen-filter>
  </filtering-router>
  ...
</outbound>
```

### JXPath Filter

The JXPath filter is very similar to the Jaxen filter. It is still used for historical purposes (it existed before the Jaxen filter).

```

<outbound>
    <filtering-router>
        <outbound-endpoint address="vm://echo" synchronous="true" />
        <mule-xml:jxpath-filter pattern="/e:purchaseOrder/e:shipTo/@country"
            expectedValue="US">
            <mule-xml:namespace prefix="e" uri="http://www.example.com" />
        </mule-xml:jxpath-filter>
    </filtering-router>
    ...
</outbound>

```

## Splitters

The XML module contains two splitters, a filter-based splitter and a round-robin splitter. For more information on these splitters, see [Outbound Routers](#).

## XML Parsers

In most cases, [SAX](#) is used to parse your XML. If you are using CXF or the [XmlToXMLStreamReader](#), [Stax](#) is used instead.

If you're using SAX, the SAX XML parser is determined by your JVM. If you want to change your SAX implementation, see <http://www.saxproject.org/quickstart.html>.

Your Rating:  Results:  0 rates

## DomToXml Transformer

### DOM/XML Transformers

Mule contains several transformers that convert between a W3C DOM object and its serialized representation. The DomToXml transformer converts DOM objects to XML, the XmlToDom transformer converts XML strings to DOM objects, and the DomToOutputHandler transformer converts from a DOM to an OutputHandler serialization.

These transformers support the [standard transformer attributes](#) plus the following:

cache: Unexpected program error: java.lang.NullPointerException

#### ***Dom To Xml Transformer***

Converts an XML payload (Document, XML stream, Source, etc.) to a serialized String representation.

#### Attributes of <dom-to-xml-transformer...>

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.

cache: Unexpected program error: java.lang.NullPointerException

#### ***Dom To Output Handler Transformer***

Converts an XML payload (Document, XML stream, Source, etc.) to an OutputHandler for efficient serialization.

#### Attributes of <dom-to-output-handler-transformer...>

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.

cache: Unexpected program error: java.lang.NullPointerException

#### ***Xml To Dom Transformer***

Transforms an XML message payload to an org.w3c.dom.Document.

#### Attributes of <xml-to-dom-transformer...>

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.

#### Example

To use the DOM/XML transformers, you add them to your Mule XML configuration as follows:

```
<xm:dom-to-xml-transformer name="DomToXml" />
<xm:xml-to-dom-transformer name="xmlToDom" returnClass="org.w3c.dom.Document" />
<xm:xml-to-output-handler-transformer name="xmlToOutputHandler" />
```

You can then reference them by name from endpoints:

```
<vm:inbound-endpoint name="testEndpoint" path="another.queue" connector-ref="vmConnector1"
transformer-refs="DomToXml" />
...
<vm:outbound-endpoint ref="xml-dom-out" transformer-refs="xmlToDom" />
...
```

Your Rating:

Results: 0 rates

## JAXB Bindings

### JAXB Bindings

Java Architecture for XML Binding (JAXB) allows Java developers to map Java classes to XML representations. JAXB provides two main features: the ability to marshal Java objects into XML and the inverse, i.e. to unmarshal XML back into Java objects. In other words, JAXB allows storing and retrieving data in memory in any XML format, without the need to implement a specific set of XML loading and saving routines for the program's class structure.

Mule support binding frameworks such as JAXB and Jackson. These frameworks use annotations to describe how data is mapped to a Java object model. For example, lets say we have an XML file that describes a person. When we receive that Xml we want to convert it into a Person object. The XML looks like this-

```
<person>
  <name>John Doe</name>
  <dob>01/01/1970</dob>
  <emailAddresses>
    <emailAddress>
      <type>home</type>
      <address>john.doe@gmail.com</address>
    </emailAddress>
    <emailAddress>
      <type>work</type>
      <address>jdoe@bigco.com</address>
    </emailAddress>
  </emailAddresses>
</person>
```

And we have an object Person we want to create from the XML. We use standard JAXB annotations to describe how to perform the mapping (The EmailAddress object is just another JavaBean with JAXB Annotations) -

```

@XmlRootElement(name = "person")
@XmlAccessorType(XmlAccessType.FIELD)
public class Person
{
    private String name;
    private String dob;

    @XmlElementWrapper(name = "emailAddresses")
    @XmlElement(name = "emailAddress")
    private List<EmailAddress> emailAddresses;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getDob() { return dob; }

    public void setDob(String dob) { this.dob = dob; }

    public List<EmailAddress> getEmailAddresses() { return emailAddresses; }

    public void setEmailAddresses(List<EmailAddress> emailAddresses) { this.emailAddresses =
emailAddresses; }
}

```

At this point Mule can figure out whether to perform a JAXB transform based on the parameters of the method being called. For example to convert the incoming XML message to a Person object -

```

public class PersonComponent {
    public void processPerson(@Payload Person person)
    {
        //tickle
    }
}

```

Here we would receive the people.xml on a JMS queue, Mule would see that Person.class is an annotated JAXB object and that we had received XML from the JMS queue and perform the conversion.

#### **Generated JAXB Classes**

Most often JAXB classes are generated from schema and this doesn't make a difference to Mule. In fact if Person was part of a set of generated classes, Mule would create a JAXBContext for all the generated classes and make that context available for the transform. When creating the context it looks for either a `jaxb.index` file or an `ObjectFactory` class in the package of the annotated class. Typically the JAXB code generator will create the `jaxb.index` file or the `ObjectFactory` class for you.

#### **Hand-rolled JAXB Classes**

If you just annotate your own classes and do not create a `jaxb.index` or `ObjectFactory` class, Mule will load just the required class into a JAXBContext for transformation.

#### **Global JAXBContext**

It is possible to define a global JAXBContext; a single context that will be used for all transforms in your application. This context will always be used instead of Mule creating one for you. This can be useful if you need to configure specific properties on the context. To create a shared JAXBContext you can create a Spring bean in your Mule XML configuration file -

```

<mule ...>

    <spring:bean name="myJaxb" class="javax.xml.bind.JAXBContext" factory-method="newInstance">
        <!-- colon-separated (:) list of package names where JAXB classes exist -->
        <spring:constructor-arg value="org.mule.jaxb.model"/>
    </spring:bean>
</mule>

```



### JAXB without annotations

Currently Mule only recognizes JAXB classes that have been annotated, which means binding information defined in a jaxb.xml descriptor is not discovered. This is a feature enhancement, please vote on [MULE-4974](#) if you need this feature.

## Intercepting JAXB Transforms

So far we have discussed how Mule will perform automatic JAXB transforms. Sometimes you may want to intercept the transform, to do this Just create a transformer with a method return or param type of your JAXB class -

```

@Transformer(sourceTypes = {String.class, InputStream.class})
public Person toPerson(Document doc, JAXBContext context) throws JAXBException
{
    return (Person) context.createUnmarshaller().unmarshal(doc);
}

```

The JAXBContext instance will either be created for you or the global context for your application will be used. One reason for doing this would be to strip out some XML elements and create objects from a subset of the XML received. For more information about transforms see the [Using Transformers](#) section.

## Using from Mule XML

As you might expect you can explicitly configure JAXB transformers in the Mule XML. For more information see [JAXB Transformers](#).

Your Rating:

Results: 1 rates

## JAXB Transformers

### JAXB Transformers

[ JAXB Bindings ]

The JAXB transformers allow objects to be serialized to XML and back again using the JAXB binding framework. To configure a transformer that will convert XML to a Person object use -

```

<mulexml:jaxb-xml-to-object-transformer name="XmlToPerson" jaxbContext-ref="myJaxb" returnClass=
"org.mule.jaxb.model.Person"/>

```

You can then reference this transformer from an endpoint:

```

<jms:inbound-endpoint queue="another.queue" transformer-refs="XmlToPerson" />

```

The returnClass is a common transformer attribute and defines that this transformer will create a Person object.

Note that we have a reference to a JAXBContext via the jaxbContext-ref attribute, you can create this context object in your configuration file -

```
<mulexml:jaxb-context name="myJAXB" packageNames="org.mule.jaxb.model"/>
```



You can always use Spring to create objects as well. To create the JAXBContext using spring you could add the following to your Mule XML configuration too.

```
<spring:bean name="myJAXB" class="javax.xml.bind.JAXBContext" factory-method="newInstance">
    <!-- comma-separated list of package names where JAXB classes exist -->
    <spring:constructor-arg value="org.mule.jaxb.model"/>
</spring:bean>
```

The opposite transformer would allow you to convert from a Person object to XML -

```
<mulexml:jaxb-object-to-xml-transformer name="PersonToXml" jaxbContext-ref="myJAXB"/>
```

## JAXB Bindings

Mule offers automatic JAXB Bindings so you don't even need use these transformers for most scenarios.

Your Rating:

Results: 0 rates

## JXPath Extractor Transformer

### JXPath Extractor Transformer

The JXPath extractor transformer evaluates an XPath expression against the current message and returns the result. By default, a single result will be returned. If multiple values are expected, set the `singleResult` property to false, which will return a list of values. This property is available for strings only (not XML nodes).

You configure the JXPath extractor transformer as follows:

```
<jxpath-extractor-transformer name="invoice" expression="/book/title" singleResult="false"/>
```

Your Rating:

Results: 0 rates

## XML Namespaces

### XML Namespaces

When dealing with XML documents in Mule you need to declare any namespaces used by the document. You can specify a namespace globally so that it can be used by XPath expressions across Mule. You can declare the namespace in any XML file in your Mule instance. To declare a namespace, include the `mule-xml.xsd` schema in your XML file:

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mulexml="http://www.mulesoft.org/schema/mule/xml"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/xml
          http://www.mulesoft.org/schema/mule/xml/3.1/mule-xml.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.1/mule.xsd">

```

Next, specify the `<namespace-manager>` element, and then add one or more `<namespace>` elements within it to declare the prefix and URL of each namespace you want to add to the namespace manager. If you already declared a namespace at the top of the file in the `<mule>` element, you can set the `includeConfigNamespaces` attribute to `true` to have the namespace manager pick up those namespaces as well.

```

<mulexml:namespace-manager includeConfigNamespaces="true">
    <mulexml:namespace prefix="foo" uri="http://foo.com"/>
</mulexml:namespace-manager>

```

You can also declare a namespace locally in an expression filter, router, or transformer using the `<namespace>` element without the `<namespace-manager>` element. You can then use that prefix within the XPath expression. For example, the following Jaxen filter declares a namespace with the prefix "e", which is then used in the filter expression:

```

<outbound>
    <filtering-router>
        <outbound-endpoint address="vm://echo" synchronous="true"/>
        <mulexml:jaxen-filter pattern="/e:purchaseOrder/e:shipTo/@country" expectedValue="US">
            <mulexml:namespace prefix="e" uri="http://www.example.com"/>
        </mulexml:jaxen-filter>
    </filtering-router>
    ...
</outbound>

```

If you had a global namespace with the "e" prefix, the local namespace URI would override the global namespace URI.

You can specify the namespace on any XML-based functionality in Mule, including the JXPath filter, Jaxen filter, XPath filter, filter-based splitter, expression splitter, round-robin splitter, JXPath extractor transformer, and XPath expression transformer in the [XML Module Reference](#), [XPath Annotation](#), as well as the SXC filter and filtering router in the [SXC Module Reference](#).

Your Rating:  Results:  0 rates

## XmlObject Transformers

### XML-Object Transformers

[ [Object to XML](#) ] [ [XML to Object](#) ] [ [Testing the Transformers](#) ]

This pair of transformers converts XML code to serialized objects and back again. For serialization of Java XML objects, see [DomToXml Transformer](#).

#### **Object to XML**

The Object to XML transformer converts any object to XML using XStream. You configure this transformer using the `<object-to-xml-transformer>` element. It takes the standard transformer attributes plus one additional attribute, `acceptUMOMessage`, which specifies whether to serialize the whole message to XML and not just its payload. This is useful with transports such as TCP where message headers are not supported and would otherwise be lost.



In Mule 2.2, the `acceptUMOMessage` attribute is named `acceptMuleMessage`.

For example:

```
<xml:object-to-xml-transformer name="ObjectToXml" acceptUMOMessage="true"/>
```

You can then reference this transformer from an endpoint:

```
<vm:inbound-endpoint path="another.queue" transformer-ref="ObjectToXml" />
```

### XML to Object

The XML to Object transformer converts XML created by the Object to XML transformer in to a Java object graph using XStream. You configure this transformer using the `<xm:xml-to-object-transformer>` element. It takes the standard transformer attributes.

For example:

```
<xm:xml-to-object-transformer name="XmlToObject" />
```

### Testing the Transformers

The transformers can be tested using functional tests. For example, the following functional test uses `FunctionalTestCase`, which is part of Mule's [Test support](#), to test the Object to XML transformer.

```
public class MuleEndpointConfigurationTestCase extends FunctionalTestCase
{
    protected String getConfigResources()
    {
        return "org/mule/test/integration/test-endpoints-config.xml";
    }

    ...
    public void testComponent4Endpoints() throws Exception
    {
        // test inbound
        Service service = muleContext.getRegistry().lookupService("TestComponent4");
        assertNotNull(service);
        assertNotNull(service.getInboundRouter().getEndpoints());
        assertEquals(1, service.getInboundRouter().getEndpoints().size());
        ImmutableEndpoint endpoint =
        (ImmutableEndpoint)service.getInboundRouter().getEndpoints().get(0);
        assertNotNull(endpoint);
        assertEquals(VMConnector.VM, endpoint.getConnector().getProtocol().toLowerCase());
        assertEquals("queue4", endpoint.getEndpointURI().getAddress());
        assertFalse(endpoint.getTransformers().isEmpty());
        assertTrue(endpoint.getTransformers().get(0) instanceof ObjectToXml);
        assertTrue(endpoint instanceof InboundEndpoint);
    }
}
```

Your Rating: 

Results:  5 rates

## XmIToXMLStreamReader Transformer

### XmIToXMLStreamReader Transformer

The XmIToXMLStreamReader transformer converts XML representations to a StAX XMLStreamReader. XMLStreamReaders allow XML to be parsed as a series of events that are "pulled" from the stream. It is very efficient.

This transformer supports the following input formats:

- javax.xml.transform.Source.class
- org.xml.sax.InputSource.class
- org.dom4j.Document.class
- org.w3c.dom.Document.class
- org.w3c.dom.Element.class
- org.mule.module.xml.transformer.DelayedResult.class
- String
- byte[]
- InputStream

## Examples

To use the transformer, you must declare a custom transformer element:

```
<custom-transformer name="XmlToXSR" class="org.mule.module.xml.transformer.XmlToXMLStreamReader"/>
```

You can also create a "reversible" XMLStreamReader:

```
<custom-transformer name="XmlToXSR" class="org.mule.module.xml.transformer.XmlToXMLStreamReader">  
    <property key="reversible" value="true"/>  
</custom-transformer>
```

This allows you to cache XML events and replay them:

```
MuleMessage message = ...;  
ReversibleXMLStreamReader xsr = (ReversibleXMLStreamReader) message.getPayload();  
  
// start caching events  
xsr.setTracking(true);  
  
// parse....  
while (...) { xsr.next(); }  
  
// Go back to the beginning of the XML document  
xsr.reset();  
  
....  
  
// Don't cache events any more  
xsr.setTracking(false);
```

Your Rating:  Results:  0 rates

## XPath Extractor Transformer

### XPath Extractor Transformer

New in Mule 2.2, the XPath extractor transformer evaluates an XPath expression against the current message and returns the result using the JAXP libraries. By default, a string result of the XPath expression is returned. The transformer can be configured to return other types of results such as a Node, NodeSet, Boolean, or Number.

You configure the XPath transformer as follows:

```
<xpath-extractor-transformer name="title" expression="/book/title" resultType="NODESET"/>
```

Your Rating:  Results:  0 rates

## XQuery Support

### XQuery Support

[ Configuration ] [ Configuration options ] [ <xquery-transformer ...> ] [ Example ]

The XQuery Module gives users the ability to perform XQuery transformations on XML messages in Mule. This works in a very similar way to the XSLT Transformer shipped with Mule.

#### Configuration

To use the XQuery transformer you need to add it to your Mule Xml configuration

```
<mxml:xquery-transformer name="xquery">
    <mxml:xquery-text>
        <![CDATA[
            declare variable $document external;
            declare variable $title external;
            declare variable $rating external;

            <cd-listings title='{$title}' rating='{$rating}'> {
                for $cd in $document/catalog/cd
                    return <cd-title>{data($cd/title)}</cd-title>
            } </cd-listings>
        ]]>
    </mxml:xquery-text>

    <mxml:context-property key="title" value="#[mule.message:header('ListingTitle')]" />
    <mxml:context-property key="rating" value="#[mule.message:header('ListingRating')]" />

</mxml:xquery-transformer>
```

Here we are configuring a transformer using in-line XQuery expressions.

We also define 2 <context-property> elements -

```
<mxml:context-property key="title" value="#[mule.message:header('ListingTitle')]" />
<mxml:context-property key="rating" value="#[mule.message:header('ListingRating')]" />
```

These properties are pulled from the current message and made available in the XQuery context so that they can be referenced in your XQuery statements. These can be object references or you can use Mule Expressions to get information from the current message.

#### Configuration options

cache: Unexpected program error: java.lang.NullPointerException

#### <xquery-transformer ...>

An Xml transformer uses XQuery to transform the message payload. Transformation objects are pooled for better performance. You can set transformation context properties on the transformer and can pull these properties from the message using Expression Evaluators.

#### Attributes

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.
maxIdleTransformers	integer	no		Transformers are pooled for better throughput, since performing and XQuery transformation can be expensive. This attribute controls how many instances will remain idle in the transformer pool.

maxActiveTransformers	integer	no		The total number of XQuery transformers that will get pooled at any given time.
xquery-file	string	no		The full path to the XQuery template file to use when performing the transformation. This can be a path on the local file system or on the classpath. This attribute is not required if the <xquery-text> element has been set.
configuration-ref	string	no		A reference to a Saxon configuration object to configure the transformer (configured as a Spring bean). If not set, the default Saxon configuration is used.

#### Child Elements

Name	Cardinality	Description
context-property	0..*	A property that will be made available to the XQuery transform context. Expression Evaluators can be used to grab these properties from the message at runtime.
xquery-text	0..1	The inline XQuery script definition. This is not required if the <xquery-file> attribute is set.

#### Example

Now your configured XQuery transformer can be referenced by an endpoint. In the following example, you can drop an XML file into a directory on the local machine (see the inbound file endpoint) and the result will be written to `System.out`.

The test data looks like -

```
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
  ...
</catalog>
```

The result written to `System.out` will look like -

```
<cd-listings title="MyList" rating="6">
  <cd-title>Empire Burlesque</cd-title>
  <cd-title>Hide your heart</cd-title>
  ...
</cd-listings>
```

The full configuration for this examples looks like -

```
<mule xmlns="http://www.mulesource.org/schema/mule/core"
      xmlns:mxm="http://www.mulesource.org/schema/mule/xml"
      xmlns:vm="http://www.mulesource.org/schema/mule/vm"
      xmlns:stdio="http://www.mulesource.org/schema/mule/stdio"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
          http://www.mulesource.org/schema/mule/stdio
          http://www.mulesource.org/schema/mule/vm
          http://www.mulesource.org/schema/mule/xml
          http://www.mulesource.org/schema/mule/xml
          http://www.mulesource.org/schema/mule/core
          http://www.mulesource.org/schema/mule/core">

    <mxm:xquery-transformer name="xquery">
        <mxm:xquery-text>
            <![CDATA[
                declare variable $document external;
                declare variable $title external;
                declare variable $rating external;

                <cd-listings title='{$title}' rating='{$rating}'> {
                    for $cd in $document/catalog/cd
                    return <cd-title>{data($cd/title)}</cd-title>
                } </cd-listings>
            ]]>
        </mxm:xquery-text>

        <mxm:context-property key="title" value="#[mule.message:header('ListingTitle')]" />
        <mxm:context-property key="rating" value="#[mule.message:header('ListingRating')]" />
    </mxm:xquery-transformer>

    <model name="main">
        <service name="Echo">
            <inbound>
                <!-- this endpoint is used by the functional test -->
                <vm:inbound-endpoint path="test.in" transformer-refs="xquery" synchronous="true" />
            </inbound>

            <echo-component />

            <outbound>
                <multicasting-router>
                    <vm:outbound-endpoint path="test.out" />
                    <stdio:outbound-endpoint system="OUT" />
                </multicasting-router>
            </outbound>
        </service>
    </model>
</mule>
```

### Testing XQuery

This can be tested using the following functional test.

```

public class XQueryFunctionalTestCase extends FunctionalTestCase
{
    protected String getConfigResources()
    {
        //Our Mule configuration file
        return "org/mule/test/integration/xml/xquery-functional-test.xml";
    }

    public void testMessageTransform() throws Exception
    {
        //We're using Xml Unit to compare results
        //Ignore whitespace and comments
        XMLUnit.setIgnoreWhitespace(true);
        XMLUnit.setIgnoreComments(true);

        //Read in src and result data
        String srcData = IOUtils.getResourceAsString("cd-catalog.xml", getClass());
        String resultData = IOUtils.getResourceAsString("cd-catalog-result-with-params.xml",
getClass());

        //Create a new Mule Client
        MuleClient client = new MuleClient();

        //These are the message properties that will get passed into the XQuery context
        Map props = new HashMap();
        props.put("ListTitle", "MyList");
        props.put("ListRating", new Integer(6));

        //Invoke the service
        MuleMessage message = client.send("vm://test.in", srcData, props);
        assertNotNull(message);
        assertNull(message.getExceptionPayload());
        //Compare results
        assertTrue(XMLUnit.compareXML(message.getPayloadAsString(), resultData).similar());
    }
}

```

Your Rating: 

Results:  0 rates

## XQuery Transformer

### XQuery Support

[\[ Configuration \]](#) [\[ Configuration options \]](#) [\[ Example \]](#)

The XQuery Module gives users the ability to perform XQuery transformations on XML messages in Mule. This works in a very similar way to the XSLT Transformer shipped with Mule.

#### **Configuration**

To use the XQuery transformer you need to add it to your Mule Xml configuration

```

<mxml:xquery-transformer name="xquery">
    <mxml:xquery-text>
        <![CDATA[
            declare variable $document external;
            declare variable $title external;
            declare variable $rating external;

            <cd-listings title='{$title}' rating='{$rating}'> {
                for $cd in $document/catalog/cd
                    return <cd-title>{data($cd/title)}</cd-title>
            } </cd-listings>
        ]]>
    </mxml:xquery-text>

    <mxml:context-property key="title" value="#[header:ListingTitle]"/>
    <mxml:context-property key="rating" value="#[header:ListingRating]"/>

</mxml:xquery-transformer>

```

Here we are configuring a transformer using in-line XQuery expressions.

We also define 2 `<context-property>` elements -

```

<mxml:context-property key="title" value="#[header:ListingTitle]"/>
<mxml:context-property key="rating" value="#[header:ListingRating]"/>

```

These properties are pulled from the current message and made available in the XQuery context so that they can be referenced in your XQuery statements. These can be object references or you can use [Mule Expressions](#) to get information from the current message.

### **Configuration options**

cache: Unexpected program error: java.lang.NullPointerException

### **`<xquery-transformer ...>`**

An Xml transformer uses XQuery to transform the message payload. Transformation objects are pooled for better performance. You can set transformation context properties on the transformer and can pull these properties from the message using Expression Evaluators.

#### **Attributes**

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.
maxIdleTransformers	integer	no		Transformers are pooled for better throughput, since performing and XQuery transformation can be expensive. This attribute controls how many instances will remain idle in the transformer pool.
maxActiveTransformers	integer	no		The total number of XQuery transformers that will get pooled at any given time.
xquery-file	string	no		The full path to the XQuery template file to use when performing the transformation. This can be a path on the local file system or on the classpath. This attribute is not required if the <code>&lt;xquery-text&gt;</code> element has been set.

configuration-ref	string	no		A reference to a Saxon configuration object to configure the transformer (configured as a Spring bean). If not set, the default Saxon configuration is used.
-------------------	--------	----	--	--

#### Child Elements

Name	Cardinality	Description
context-property	0..*	A property that will be made available to the XQuery transform context. Expression Evaluators can be used to grab these properties from the message at runtime.
xquery-text	0..1	The inline XQuery script definition. This is not required if the <xquery-file> attribute is set.

#### Example

Now your configured XQuery transformer can be referenced by an endpoint. In the following example, you can drop an XML file into a directory on the local machine (see the inbound file endpoint) and the result will be written to `System.out`.

The test data looks like -

```
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>
  ...
</catalog>
```

The result written to `System.out` will look like -

```
<cd-listings title="MyList" rating="6">
  <cd-title>Empire Burlesque</cd-title>
  <cd-title>Hide your heart</cd-title>
  ...
</cd-listings>
```

The full configuration for this example looks like -

```

<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:mxm="http://www.mulesoft.org/schema/mule/xml"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd
          http://www.mulesoft.org/schema/mule/xml http://www.mulesoft.org/schema/mule/xml/3.0/mule-xml.xsd
          http://www.mulesoft.org/schema/mule/core
          http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd">

    <mxm:xquery-transformer name="xquery">
        <mxm:xquery-text>
            <![CDATA[
                declare variable $document external;
                declare variable $title external;
                declare variable $rating external;

                <cd-listings title='{$title}' rating='{$rating}'>
                    for $cd in $document/catalog/cd
                    return <cd-title>{data($cd/title)}</cd-title>
                } </cd-listings>
            ]]>
        </mxm:xquery-text>

        <mxm:context-property key="title" value="#[header:ListingTitle]"/>
        <mxm:context-property key="rating" value="#[header:ListingRating]"/>
    </mxm:xquery-transformer>

    <model name="main">
        <service name="Echo">
            <inbound>
                <!-- this endpoint is used by the functional test -->
                <vm:inbound-endpoint path="test.in" transformer-refs="xquery" synchronous="true"/>
            </inbound>

            <echo-component/>

            <outbound>
                <multicasting-router>
                    <vm:outbound-endpoint path="test.out"/>
                    <stdio:outbound-endpoint system="OUT"/>
                </multicasting-router>
            </outbound>
        </service>
    </model>
</mule>

```

## Testing it

This can be tested using the following functional test -

```

public class XQueryFunctionalTestCase extends FunctionalTestCase
{
    protected String getConfigResources()
    {
        //Our Mule configuration file
        return "org/mule/test/integration/xml/xquery-functional-test.xml";
    }

    public void testMessageTransform() throws Exception
    {
        //We're using Xml Unit to compare results
        //Ignore whitespace and comments
        XMLUnit.setIgnoreWhitespace(true);
        XMLUnit.setIgnoreComments(true);

        //Read in src and result data
        String srcData = IOUtils.getResourceAsString("cd-catalog.xml", getClass());
        String resultData = IOUtils.getResourceAsString("cd-catalog-result-with-params.xml",
getClass());

        //Create a new Mule Client
        MuleClient client = new MuleClient(muleContext);

        //These are the message properties that will get passed into the XQuery context
        Map props = new HashMap();
        props.put("ListTitle", "MyList");
        props.put("ListRating", new Integer(6));

        //Invoke the service
        MuleMessage message = client.send("vm://test.in", srcData, props);
        assertNotNull(message);
        assertNull(message.getExceptionPayload());
        //Compare results
        assertTrue(XMLUnit.compareXML(message.getPayloadAsString(), resultData).similar());
    }
}

```

Your Rating: 

Results:  0 rates

## XSLT Transformer

cache: Unexpected program error: java.lang.NullPointerException

**<xslt-transformer ...>**

The XSLT transformer uses XSLT to transform the message payload. Transformation objects are pooled for better performance. You can set transformation context properties on the transformer and can pull these properties from the message using Expression Evaluators. This works in a very similar way to the [XQuery Transformer](#) on [MuleForge](#).

### Attributes

Name	Type	Required	Default	Description
outputEncoding	string	no		The encoding to use for the resulting XML/Text.
maxIdleTransformers	integer	no		Transformers are pooled for better throughput, since performing and XSL transformation can be expensive. This attribute controls how many instances will remain idle in the transformer pool.

maxActiveTransformers	integer	no		The total number of XSLT transformers that will get pooled at any given time.
xsl-file	string	no		The full path to the XSL template file to use when performing the transformation. This can be a path on the local file system or on the classpath. This attribute is not required if the <xslt-text> element has been set.
uriResolver	name (no spaces)	no		The URI resolver to use when validating the XSL output. If not set, a default resolver will be used that checks for resources on the local file system and classpath.
transformerFactoryClass	name (no spaces)	no		The fully qualified class name of the {{javax.xml.TransformerFactory}} instance to use. If not specified, the default JDK factory {{TransformerFactory.newInstance()}} will be used.

#### Child Elements

Name	Cardinality	Description
xslt-text	0..1	The inline XSLT script definition. This is not required if the {{xslt-file}} attribute is set.
context-property	0..*	A property that will be made available to the transform context. Expression Evaluators can be used to grab these properties from the message at runtime.

#### Example

The following example demonstrates how to configure an inline XSLT transformer pulling parameters from the current message.

To use the XSLT transformer, you add it to your Mule XML configuration as follows:

```

<mulexml:xslt-transformer name="xslt">
  <mulexml:xslt-text>
    <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
      <xsl:output method="xml"/>
      <xsl:param name="title"/>
      <xsl:param name="rating"/>
      <xsl:template match="catalog">
        <xsl:element name="cd-listings">
          <xsl:attribute name="title">
            <xsl:value-of select="$title"/>
          </xsl:attribute>
          <xsl:attribute name="rating">
            <xsl:value-of select="$rating"/>
          </xsl:attribute>
          <xsl:apply-templates/>
        </xsl:element>
      </xsl:template>

      <xsl:template match="cd">
        <xsl:element name="cd-title">
          <xsl:value-of select = "title" />
        </xsl:element>
      </xsl:template>
    </xsl:stylesheet>
  </mulexml:xslt-text>
  <mulexml:context-property key="title" value="#[header:ListTitle]"/>
  <mulexml:context-property key="rating" value="#[header:ListRating]"/>

```

This example configures a transformer using inline XSLT expressions. It also defines two context parameters:

```

<mulexml:context-property key="title" value="#[header:ListTitle]"/>
<mulexml:context-property key="rating" value="#[header:ListRating]"/>

```

These parameters are pulled from the current message and made available in the XSLT context so that they can be referenced in your XSLT statements. You can use any valid expression. In this example, the header evaluator is used to pull a header from the current message.

Your configured XSLT transformer can be referenced by an endpoint. In the following example, the result is written to `System.out`. The test data looks like this:

```

<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <price>9.90</price>
    <year>1988</year>
  </cd>

```

The result written to `System.out` looks like this:

```
<cd-listings title="MyList" rating="6">
<cd-title>Empire Burlesque</cd-title>
<cd-title>Hide your heart</cd-title>
<!-- ... </cd-listings> -->
```

The full configuration for this example is shown below.

```

<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesoft.org/schema/mule/core"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:mule="http://www.mulesoft.org/schema/mule/core"
      xmlns:mulexml="http://www.mulesoft.org/schema/mule/xml"
      xmlns:vm="http://www.mulesoft.org/schema/mule/vm"
      xmlns:stdio="http://www.mulesoft.org/schema/mule/stdio"
      xmlns:spring="http://www.springframework.org/schema/beans"
      xsi:schemaLocation="
          http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/3.0/mule.xsd
          http://www.mulesoft.org/schema/mule/vm http://www.mulesoft.org/schema/mule/vm/3.0/mule-vm.xsd
          http://www.mulesoft.org/schema/mule/stdio
          http://www.mulesoft.org/schema/mule/xml/3.0/mule-xml.xsd
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <mulexml:xslt-transformer name="xslt">
        <mulexml:xslt-text>
            <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
                <xsl:output method="xml"/>
                <xsl:param name="title"/>
                <xsl:param name="rating"/>
                <xsl:template match="catalog">
                    <xsl:element name="cd-listings">
                        <xsl:attribute name="title">
                            <xsl:value-of select="$title"/>
                        </xsl:attribute>
                        <xsl:attribute name="rating">
                            <xsl:value-of select="$rating"/>
                        </xsl:attribute>
                        <xsl:apply-templates/>
                    </xsl:element>
                </xsl:template>
            </xsl:stylesheet>
        </mulexml:xslt-text>
        <mulexml:context-property key="title" value="#[header:ListTitle]"/>
        <mulexml:context-property key="rating" value="#[header:ListRating]"/>
    </mulexml:xslt-transformer>

    <model name="main">
        <service name="Echo">
            <inbound>
                <!-- this endpoint is used by the functional test -->
                <vm:inbound-endpoint path="test.in" transformer-refs="xslt" synchronous="true"/>
            </inbound>
            <echo-component/>
            <outbound>
                <pass-through-router>
                    <stdio:outbound-endpoint system="OUT"/>
                </pass-through-router>
            </outbound>
        </service>
    </model>
</mule>

```

## Testing the Transformer

This transformer can be tested using the following functional test. Note that it uses `FunctionalTestCase`, which is part of Mule's Test support.

```

public class XSLTWikiDocsTestCase extends FunctionalTestCase
{
    protected String getConfigResources()
    {
        return "org/mule/test/integration/xml/xslt-functional-test.xml";
    }

    public void testMessageTransform() throws Exception
    {
        //We're using Xml Unit to compare results
        //Ignore whitespace and comments
        XMLUnit.setIgnoreWhitespace(true);
        XMLUnit.setIgnoreComments(true);

        //Read in src and result data
        String srcData = IOUtils.getResourceAsString(
            "org/mule/test/integration/xml/cd-catalog.xml", getClass());
        String resultData = IOUtils.getResourceAsString(
            "org/mule/test/integration/xml/cd-catalog-result-with-params.xml", getClass());

        //Create a new Mule Client
        MuleClient client = new MuleClient(muleContext);

        //These are the message properties that will get passed into the XQuery context
        Map<String, Object> props = new HashMap<String, Object>();
        props.put("ListTitle", "MyList");
        props.put("ListRating", new Integer(6));

        //Invoke the service
        MuleMessage message = client.send("vm://test.in", srcData, props);
        assertNotNull(message);
        assertNull(message.getExceptionPayload());
        //Compare results
        assertTrue(XMLUnit.compareXML(message.getPayloadAsString(), resultData).similar());
    }
}

```

Your Rating: 

Results:  2 rates

## Data Bindings Reference

### Data Bindings Reference

Data binding frameworks such as JAXB or Jackson (JSON) allow developers easily work with XML and JSON by binding the data to object graphs.

- [JAXB Bindings](#)
- [JSON Bindings](#)

Your Rating: 

Results:  0 rates

## BPM Module Reference

### BPM Module Reference

[ [Introduction](#) ] [ [Namespace and Syntax](#) ] [ [Features](#) ] [ [Usage](#) ] [ [BPMS Support](#) ] [ [Writing a BPMS Plug-in](#) ] [ [Reference](#) ] [ [Notes](#) ]

#### Introduction

BPM stands for Business Process Management and simplistically can be thought of as a system which automates business processes. Typically a BPM system will have a graphical interface which allows you to model your business processes visually so that a business analyst or other non-programmer can easily understand them and verify that they indeed reflect the reality of your business.

Mule's support for BPM allows you to send/receive messages to/from a running process. A message from Mule can start or advance a process, the message can be stored as a process variable, and a running process can send messages to any endpoint in your Mule application.

Mule can interact with many kinds of BPM systems. Integrations are readily available for several popular implementations and integrating with a new system is straightforward.

### Considerations

If the business logic for your Mule application is relatively simple, does not change very often, and is mostly stateless, you are probably best off implementing it using Mule's orchestration functionality such as [flows](#), [routers](#), and [custom components](#).

However, you might consider using BPM to model your business logic in the following cases:

- If the execution in your flow has many discrete steps and/or multiple paths based on decision criteria, modeling it as a process may help better visualize it.
- If your business logic is in constant flux, BPM might be a good way to decouple the logic from your integration.
- If your business logic or processes need to be reviewed or updated by non-programmers, BPM makes that possible.
- If a flow does not go from start to finish within a few minutes, it's probably a good candidate for BPM because intermediate process state is persisted so it does not need to be held in memory.
- If your integration requires human input, BPM lets you add a human step to your process via task management and a web app.
- If your application needs a step-by-step audit trail, BPM might be a good option because process history is kept in a database.

It is likely that a BPM engine would add some performance overhead. If high-performance is critical to your application, you should consider using Mule's orchestration instead to model your process.

### Namespace and Syntax

XML namespace:

```
xmlns:bpm "http://www.mulesoft.org/schema/mule/bpm"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd
```

Syntax:

```
<bpm:process processName="myProcess" processDefinition="myProcess.xml" />
```

### Features

- Incoming Mule events can launch new processes, advance or terminate running processes.
- A running process can send synchronous or asynchronous messages to any Mule endpoint.
- Synchronous responses from Mule are automatically fed back into the running process and can be stored into process variables.
- Mule can interact with different running processes in parallel.

To see BPM with Mule in action (using JBoss jBPM), take a look at the [Loan Broker BPM Example](#) (available in the full Mule distribution).

### Usage

Import the XML schema for this module as follows:

```
xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd"
```

Then you can use the `<bpm:process>` element as follows, generally preceded by an inbound endpoint. See the example configuration, below.

```
<bpm:process processName="myProcess" processDefinition="myProcess.xml" />
```

- At a minimum, when triggering a BPM process, you must specify the `processName` and `processDefinition` attributes in the `<bpm:process>` element. The syntax of these elements will be specific to the BPMS implementation you are using.
- Incoming Mule messages and properties can be stored as process variables, and process variables can be sent as Mule messages. How to configure this depends on the specific BPMS and its process definition language. For supported implementations, like jBPM, please see the relevant module documentation.
- Incoming messages from Mule to the BPMS are correlated based on the message property `MULE_BPM_PROCESS_ID`. If a process already exists with this ID, then the message will advance the existing process one step. Otherwise, a new process will be created and started. Any outgoing messages generated by the process will have `MULE_BPM_PROCESS_ID` set on them. In the case of an asynchronous response, it is important that your application maintain this property in the response message so that it gets correlated with the correct process instance. If you wish to use a different message property for tracking the process ID, you can specify it using the `processIdField` attribute.
- The BPMS being used (any system which implements the BPMS interface described below) must be declared at the top of your Mule configuration. This may be a dedicated XML element if one is provided for your BPMS (such as `<bpm:jbpms>`) or a Spring bean in the case of a custom implementation or integration. If more than one BPMS is being used in your application, you must specify the `bpms-ref` attribute on the `<bpm:process>` element to disambiguate, otherwise the BPMS will be found automatically from Mule's object registry (akin to the `connector-ref` attribute on an `<endpoint>`).

## Configuration Examples

### Example configuration

```
<mule ... xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
    xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
    http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd" ...>

    <spring:bean id="testBpms" class="com.foo.FooBPMS" />

    <flow name="MessagesToProcess">
        <jms:inbound-endpoint queue="queueA" />
        <bpm:process processName="myProcess" processDefinition="myProcess.xml" />
    </flow>

    <flow name="SomeSynchronousService">
        <inbound-endpoint ref="callService" exchange-pattern="REQUEST-RESPONSE" />
        <component class="com.foo.SomeService" />
    </flow>
</mule>
```

In this configuration, an inbound message on the JMS queue `queueA` will trigger the process defined by the file "myProcess.xml" , whose format depends on the underlying BPMS. The inbound message is stored as a process variable. The BPMS which deploys and runs the process is that defined by the class `com.foo.FooBPMS` . Since there is only one implementation in the application, no explicit reference is required. A step in the process definition might send a message to the endpoint `callService` , in which case the synchronous response from `com.foo.SomeService` could be also be stored as a process variable. Note that since this logic occurs in the process definition, it is not visible in the Mule configuration.

## Multiple BPMS's

```
<spring:bean id="bpms1" class="com.foo.FooBPMS" />

<spring:bean id="bpms2" class="com.bar.BarBPMS" />

<flow name="ProcessFlow1">
    ...cut...
    <bpm:process processName="process1" processDefinition="process1.def" bpms-ref="bpms1" />
</flow>

<flow name="ProcessFlow2">
    ...cut...
    <bpm:process processName="process2" processDefinition="process2.cfg" bpms-ref="bpms2" />
</flow>
```

This configuration snippet illustrates how to use the `bpms-ref` attribute to disambiguate between more than one BPMS's. If there is only one BPMS available, this attribute is unnecessary.

## Example configuration with <service>

```
<mule ...cut...
<model>
    ...cut...
    <service name="MessagesToProcess">
        <inbound>
            <jms:inbound-endpoint queue="queueA" />
        <inbound>
            <bpm:process processName="myProcess" processDefinition="myProcess.xml" />
        </service>
    </model>
</mule>
```

New implementations are recommended to use `flows`, but Mule 2.x users will be more familiar with services.

## BPMS Support

The Mule distribution includes native support for [JBoss jBPM](#), a popular embeddable BPMS. For information see [JBoss jBPM Module Reference](#).

Several other BPMS solutions are also already supported and maintained on the Muleforge. These include:

- Apache [Activiti](#)
- BonitaSoft [Bonita](#)

Support for [JBoss jBPM](#) is included in the Mule distribution, for information see [JBoss jBPM Module Reference](#). Support for other BPM products such as [Activiti](#) and [Bonita](#) may be found on the [MuleForge](#).

## Writing a BPMS Plug-in

One of the basic design principles of Mule is to promote maximum flexibility for the user. Based on this, the user should ideally be able to "plug in" any BPM system or even their own custom BPMS implementation to use with Mule. Unfortunately, there is no standard JEE specification to enable this. Therefore, Mule simply defines its own simple interface.

```

public interface BPMS
{
    public Object startProcess(Object processType, Object transition, Map processVariables) throws
Exception;

    public Object advanceProcess(Object processId, Object transition, Map processVariables) throws
Exception;

    // MessageService contains a callback method used to generate Mule messages from your process.
    public void setMessageService(MessageService msgService);
}

```

Any BPM system that implements the interface ( `org.mule.module.bpm.BPMS` ) can "plug in" to Mule via the BPM module. Creating a connector for an existing BPM system can be as simple as creating a wrapper class that maps this interface to the native APIs of that system.

## Reference

### **Configuration Reference**

cache: Unexpected program error: java.lang.NullPointerException

#### Process

A process backed by a BPMS such as jBPM.

##### **Attributes of <process...>**

Name	Type	Required	Default	Description
bpms-ref	string	no		An optional reference to the underlying BPMS. This is used to disambiguate in the case where more than one BPMS is available.
processName	string	yes		The logical name of the process. This is used to look up the running process instance from the BPMS.
processDefinition	string	yes		The resource containing the process definition, this will be used to deploy the process to the BPMS. The resource type depends on the BPMS being used.
processIdField	string	no		This field will be used to correlate Mule messages with processes. If not specified, it will default to MULE_BPM_PROCESS_ID.

##### **Child Elements of <process...>**

Name	Cardinality	Description
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

## XML Schema

Complete schema reference documentation.

## Maven

If you are using Maven to build your application, use the following groupId/artifactId to include this module as a dependency:

```

<dependency>
    <groupId>org.mule.modules</groupId>
    <artifactId>mule-module-bpm</artifactId>
</dependency>

```

## Notes

- This module is designed for BPM engines that provide a Java API. If you need to integrate with a BPEL engine, you can do so using standard web services.
- As of Mule 3.0.1, the recommended way to interact with a BPM system is via the <bpm:process> component / message processor. Usage of the legacy BPM transport is still supported for 3.0.x but has been removed for 3.1. Documentation for the legacy BPM transport is [here](#).

Your Rating: 

Results:  0 rates

## JBoss jBPM Module Reference

### JBoss jBPM Module Reference

[ [Introduction](#) ] [ [Namespace and Syntax](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Configuration Examples](#) ] [ [Reference](#) ]

#### Introduction

JBoss jBPM is a best-of-breed open source BPMS and is well-integrated with Mule. One advantage of jBPM is that it is embedded directly in the Mule runtime, allowing for faster performance. For general information on jBPM and how to configure it, refer to the [jBPM User Guide](#)



This module provides a "Plug-in" for JBoss jBPM to be used with Mule's BPM support. If you have not yet read the general documentation for Mule's [BPM Support](#), please read that first and then come back to this page.

#### Namespace and Syntax

XML namespace:

```
xmlns:bpm "http://www.mulesoft.org/schema/mule/bpm"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd
```

Syntax:

```
<bpm:j bpm />  
<bpm:process processName="myProcess" processDefinition="my-process.jpd1.xml" />
```

#### Features

- Simple declaration of jBPM as the BPMS in your Mule configuration using sensible defaults.
- Custom elements for jBPM's process definition language (jPDL) which allow you to easily integrate Mule into your business processes.

Refer to [BPM Module Reference](#) for a list of general features offered by Mule's BPM support.

To see jBPM with Mule in action, take a look at the [Loan Broker BPM Example](#) (available in the full Mule distribution).



The jBPM libraries are bundled with the Mule distribution. As of Mule 3.0.1, jBPM 4.4 is the latest supported version.

#### Usage

Using jBPM with Mule consists of a few things:

- Configuring jBPM
- Configuring Hibernate and the database used to store process state
- Declaring jBPM as the BPMS to use in your Mule configuration
- Interacting with Mule from your process definition

## jBPM Configuration

The default configuration file for jBPM is called `jbpm.cfg.xml`. You will need to include this file as part of your Mule application. If defaults are ok for you, then it could be as simple as the following.

### jBPM Configuration (`jbpm.cfg.xml`)

```
<jbpm-configuration>
    <import resource="jbpm.default.cfg.xml" />
    <import resource="jbpm.jpd1.cfg.xml" />
    <import resource="jbpm.tx.hibernate.cfg.xml" />

    <process-engine-context>
        <object class="org.mule.module.jbpm.MuleMessageService" />
    </process-engine-context>
</jbpm-configuration>
```

Note that you need to define the `MuleMessageService` within `<process-engine-context>` otherwise jBPM will not be able to "see" Mule.

For more configuration options, refer to the [jBPM documentation](#).

## Database Configuration

jBPM uses Hibernate to persist the state of your processes, so you will need to provide a [database supported by Hibernate](#) and include any client jars as part of your Mule application. You also need to provide the file `jbpm.hibernate.cfg.xml` with the appropriate Hibernate settings for your chosen database.

For example, a simple in-memory Derby database might use these settings:

### Derby settings

```
<property name="hibernate.dialect">org.hibernate.dialect.DerbyDialect</property>
<property name="hibernate.connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</property>
<property name="hibernate.connection.url">jdbc:derby:memory:muleEmbeddedDB</property>
<property name="hibernate.hbm2ddl.auto">create-drop</property>
```

while an Oracle database might use these settings:

### Oracle settings

```
<property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:user/pass@server:1521:dbname</property>
```

One very important Hibernate setting to pay attention to is `hibernate.hbm2ddl.auto`. If this is set to `create`, Hibernate will automatically create the DB schema for jBPM at startup if it does not yet exist in your database. If it is set to `create-drop`, the schema will also be deleted at shutdown, which is useful in test environments.

For more configuration options, refer to the [jBPM documentation](#) and/or [Hibernate documentation](#).

## Mule configuration

Using jBPM in your Mule configuration is then as simple as including the `<bpm:jbpm>` element. The default configuration file is assumed to be `jbpm.cfg.xml`, otherwise you can specify it with the `configurationResource` attribute.

Default config
<bpm:jbpm />
Custom config
<bpm:jbpm name="jbpm" configurationResource="custom-jbpm-config.cfg.xml"/>

### Process definition (jPDL)

For lack of a good standard in the BPM community, jBPM has traditionally used its own DSL for process definitions called **jPDL**. You will find that it is very easy to learn, and there is an [Eclipse plug-in](#) called the Graphical Process Designer which allows you to create your process definitions visually as well.



In future versions, the preferred definition language will likely be [BPMN 2.0](#), which is now a widely-accepted standard in the BPM community. Mule currently support BPMN-defined processes through the Activiti BPM Module.

Mule provides two custom elements for jBPM's process definition language (jPDL). You can use these in your process definition along with other standard jPDL elements such as `<state>`, `<java>`, `<script>`, `<decision>`.

Element	Usage	Description	Required Attributes
<code>&lt;mule-send&gt;</code>	<code>&lt;mule-send expr="" endpoint="" exchange-pattern="" var="" type=""&gt;</code>	Activity which sends a message with the payload <code>expr</code> to the Mule <code>endpoint</code> . If <code>exchange-pattern</code> = request-response (the default value), the send will block and the response message will be stored into <code>var</code> . If the message is not of <code>type</code> , an exception will be thrown. <code>expr</code> can be a literal value or an expression which references process variables.	The only mandatory attributes are <code>expr</code> and <code>endpoint</code> , the rest are optional.
<code>&lt;mule-receive&gt;</code>	<code>&lt;mule-receive var="" endpoint="" type=""&gt;</code>	Wait state which expects a message to arrive from the Mule <code>endpoint</code> and stores it into <code>var</code> . If the message is not of <code>type</code> , an exception will be thrown. <code>&lt;mule-receive&gt;</code> can replace <code>&lt;start&gt;</code> as the first state of a process and this way you can store the message which initiated the process into a variable.	The attributes are all optional.

### Configuration Examples

Example Mule Configuration
<pre> &lt;mule ...cut...   xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"   xsi:schemaLocation="...cut...     http://www.mulesoft.org/schema/mule/bpm     http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd"&gt;    &lt;bpm:jbpm name="jbpm" /&gt;    &lt;flow name="ToBPMs"&gt;     &lt;composite-source&gt;       &lt;inbound-endpoint ref="CustomerRequests" /&gt;       &lt;inbound-endpoint ref="CreditProfiles" /&gt;     &lt;/composite-source&gt;     &lt;bpm:process processName="LoanBroker" processDefinition="loan-broker-process.jpd1.xml" /&gt;   &lt;/flow&gt;   ...cut... &lt;/mule&gt; </pre>

Import the BPM schema.

Declare jBPM as the BPMs implementation to use.

Incoming messages on these endpoints start/advance the process and are stored as process variables.

The process defined in loan-broker-process.jpd1.xml will get deployed to jBPM at startup.

#### Example jPDL Process Definition

```
<process name="LoanBroker" xmlns="http://jbpm.org/4.3/jpd1">

    <mule-receive name="incomingCustomerRequest" endpoint="CustomerRequests" type="foo.messages.CustomerQuoteRequest" var="customerRequest">
        <transition to="sendToCreditAgency" />
    </mule-receive>

    <mule-send name="sendToCreditAgency"
        expr="#{customerRequest.customer}" endpoint="CreditAgency" exchange-pattern="one-way">
        <transition to="sendToBanks" />
    </mule-send>

    <decision name="sendToBanks">
        <transition to="sendToBigBank">
            <condition expr="#{customerRequest.loanAmount >= 20000}" />
        </transition>
        <transition to="sendToMediumBank">
            <condition expr="#{customerRequest.loanAmount >= 10000}" />
        </transition>
        ...cut...
    </decision>

    ...cut...
    <end name="loanApproved" />
</process>
```

An incoming message is expected on the endpoint CustomerRequests of type foo.messages.CustomerQuoteRequest and is stored into the process variable customerRequest.

A new message is sent to the endpoint CreditAgency whose payload is an expression using the process variable customerRequest. <decision> is a standard jPDL element.

The decision logic uses the process variable customerRequest.

#### Example Mule Configuration with <service>

```
<mule ...cut...
    <bpm:jbpm name="jbpm" />

    <model>
        <service name="ToBPMS">
            <inbound>
                <inbound-endpoint ref="CustomerRequests" />
                <inbound-endpoint ref="CreditProfiles" />
            </inbound>
            <bpm:process processName="LoanBroker" processDefinition="loan-broker-process.jpd1.xml" />
        </service>
        ...cut...
    </model>
</mule>
```

New implementations are recommended to use **flows**, but Mule 2.x users will be more familiar with services.

## Reference

### **Configuration Reference**

cache: Unexpected program error: java.lang.NullPointerException

### **Jbpm**

#### **Attributes of <jbpm...>**

Name	Type	Required	Default	Description
name	name (no spaces)	no		An optional name for this BPMS. Refer to this from the "bpms-ref" field of your process in case you have more than one BPMS available.
configurationResource	string	no		The configuration file for jBPM, default is "jbpm.cfg.xml" if not specified.
processEngine-ref	string	no		A reference to the already-initialized jBPM ProcessEngine. This is useful if you use Spring to configure your jBPM instance. Note that the "configurationResource" attribute will be ignored in this case.

#### Child Elements of <jbpm...>

Name	Cardinality	Description

#### XML Schema

This module uses the schema from the [BPM Module](#), it does not have its own schema.

Import the BPM schema as follows:

```
xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
http://www.mulesoft.org/schema/mule/bpm/3.1/mule-bpm.xsd"
```

Refer to [BPM Module Reference](#) for detailed information on the elements of the BPM schema.

#### Maven

If you are using Maven to build your application, use the following groupId/artifactId to include this module as a dependency:

```
<dependency>
<groupId>org.mule.modules</groupId>
<artifactId>mule-module-jbpm</artifactId>
</dependency>
```

Your Rating:  Results:  0 rates

## Atom Module Reference

### Atom Module Reference

[ [Introduction](#) ] [ [Namespace and Syntax](#) ] [ [Considerations](#) ] [ [Features](#) ] [ [Example Configurations](#) ] [ [Implementing an AtomPub Server](#) ] [ [Publishing to the Atom Component](#) ] [ [Route Filtering](#) ] [ [Configuration Reference](#) ] [ [Schema](#) ] [ [Javadoc API Reference](#) ] [ [Maven](#) ]

Your Rating:  Results:  2 rates

#### Introduction

The name Atom applies to a pair of related standards. The Atom Syndication Format is an XML language used for web feeds, while the Atom Publishing Protocol (AtomPub or APP) is a simple HTTP-based protocol for creating and updating web resources. Mule contains support for both.

#### Namespace and Syntax

XML namespace

```
http://www.mulesoft.org/schema/mule/atom
```

XML Schema location

```
http://www.mulesoft.org/schema/mule/atom http://www.mulesoft.org/schema/mule/atom/3.1/mule-atom.xsd
```

## Considerations

### Features

The Mule Atom Module includes an implementation of [Apache Abdera](#), making it possible to integrate easily with Atom feeds and Atom Publishing Protocol servers from within a Mule configuration.

The module current supports the following capabilities:

- Consuming Atom feeds
- Publishing Atom entries
- Server side AtomPub support as a Mule service

### ETags

The Atom Http endpoint respects ETags and the 304 Not Modified response code by default, so you don't need to worry about consuming unnecessary updates.

## Example Configurations

### **Consuming Feeds and other Atom resources**

One of the most common patterns is for integrating with Atom is polling for feed updates. With Mule this can be done quite easily. You must first write your class which receives an Atom Entry:

```
EntryReceiver.java
```

```
package org.mule.module.atom.example;

import org.apache.abdera.model.Feed;

public class EntryReceiver {
    public void processEntry(Entry entry) throws Exception {
        //process the entry
    }
}
```

Use an atom feed splitter to split an incoming feed into individual messages so the component method will be invoked for each entry in the feed. The flow configuration would look like so:

```
<flow name="eventConsumer">
<poll frequency="10000">
    <http:outbound-endpoint address="http://localhost:9002/events"/>
</poll>
<atom:feed-splitter/>
<atom:entry-last-updated-filter/>
<component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>
```

The `atom:entry-last-updated-filter` is optional. It should only be used to filter older entries from the feed. Note that the `atom:entry-last-updated-filter` should come after the `<atom:feed-splitter/>` since you need to split the feed into entries so that the filter can process them. Also note that we do not set a `lastUpdate` date on the filter. This implies the default behavior that all available entries will be read before the processing of any entries that are new as of the last read.

### **Accessing the Feed itself**

If you need access to the feed itself, parse it using the object-to-feed transformer:

```

<flow name="eventConsumer">
    <poll frequency="10000">
        <http:outbound-endpoint address="http://localhost:9002/events"/>
    </poll>
    <atom:object-to-feed-transformer/>
    <component class="org.mule.module.atom.event.FeedReceiver"/>
</flow>

```

Now your component will only be invoked once for each feed change no matter how many entries were added or updated. The method on your component should expect a `org.apache.abdera.model.Feed` object.

#### FeedReceiver.java

```

package org.mule.module.atom.example;

import org.apache.abdera.model.Feed;

public void processFeed(Feed feed) throws Exception {
    //do stuff
}

```

If you want access to the feed object while using the feed splitter, it is available via a header on the current message called 'feed.object'.

### **Accessing feeds over other Protocols**

You can receive feeds and process them using other Mule connectors such as JMS, File or XMPP. To do this, the atom feed needs to be served over the connector. For instance, an atom document is sent over JMS or polled from a file. The atom schema defines a `<atom:feed-splitter/>` message processor that can split messages received from an endpoint, like so:

#### Consuming an Atom feed over JMS

```

<flow name="feedSplitterConsumer">
    <jms:inbound-endpoint queue="feed.in">
        <atom:feed-splitter/>
    </jms:inbound-endpoint>
    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

#### Consuming an Atom feed from a file

```

<flow name="feedSplitterConsumer">
    <file:inbound-endpoint path="${mule.working.dir}" pollingFrequency="1000" >
        <file:filename-wildcard-filter pattern="*.atom"/>
        <atom:feed-splitter/>
    </file:inbound-endpoint>
    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

### **Implementing an AtomPub Server**

Abdera's server side component centers on the notion of a Provider. A Provider manages a service's Workspaces and Collections.

You can create an AtomPub service in Mule by using the `<atom:component/>` XML element and reference an Abdera service context.

### **Creating the Abdera Service Context**

The following example shows how to create an Abdera context that builds a JCR repository to store atom entries. These entries can then be served as a feed.

### abdera-config.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:a="http://abdera.apache.org"
       xsi:schemaLocation="
           http://abdera.apache.org http://abdera.apache.org/schemas/abdera-spring.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <a:provider id="provider">
        <a:workspace title="JCR Workspace">
            <ref bean="jcrAdapter" />
        </a:workspace>
    </a:provider>

    <bean id="jcrRepository" class="org.apache.jackrabbit.core.TransientRepository" destroy-method="shutdown"/>

    <bean id="jcrAdapter"
          class="org.apache.abdera.protocol.server.adapters.jcr.JcrCollectionAdapter" init-method="initialize">
        <property name="author" value="Mule"/>
        <property name="title" value="Event Queue"/>
        <property name="collectionNodePath" value="entries"/>
        <property name="repository" ref="jcrRepository"/>
        <property name="credentials">
            <bean class="javax.jcr.SimpleCredentials">
                <constructor-arg>
                    <value>username</value>
                </constructor-arg>
                <constructor-arg>
                    <value>password</value>
                </constructor-arg>
            </bean>
        </property>
        <property name="href" value="events"/>
    </bean>
</beans>
```

The `<a:provider>` creates an Abdera DefaultProvider and allows you to add workspaces and collections to it. This provider reference is used by the the `<atom:component/>` in Mule to store any events sent to the component.

```
<flow name="atomPubEventStore">
    <http:inbound-endpoint address="http://localhost:9002"/>
    <atom:component provider-ref="provider"/>
</flow>
```

### Publishing to the Atom Component

You may also want to publish Atom entries or media entries to the `<atom:component/>` or to an external AtomPub collection. Here is a simple outbound endpoint which creates an Abdera Entry via the `entry-builder-transformer` and POSTs it to the AtomPub collection:

```

<outbound-endpoint address="http://localhost:9002/events" mimeType="application/atom+xml;type=entry"
connector-ref="HttpConnector">
    <atom:entry-builder-transformer>
        <atom:entry-property name="author" evaluator="string" expression="Ross Mason"/>
        <atom:entry-property name="content" evaluator="payload" expression="" />
        <atom:entry-property name="title" evaluator="header" expression="title"/>
        <atom:entry-property name="updated" evaluator="function" expression="now"/>
        <atom:entry-property name="id" evaluator="function" expression="uuid"/>
    </atom:entry-builder-transformer>
</outbound-endpoint>

```

You could also create the Entry manually for more flexibility and send it as your Mule message payload. Here's a simple example of how to create an Abdera Entry:

**Create an Abdera Entry**

```

package org.mule.providers.abdera.example;

import java.util.Date;

import org.apache.abdera.Abdera;
import org.apache.abdera.factory.Factory;
import org.apache.abdera.model.Entry;
import org.mule.transformer.AbstractTransformer;

public class EntryTransformer extend AbstractTransformer {
    public Object doTransform(Object src, String encoding) {
        Factory factory = Abdera.getInstance().getFactory();

        Entry entry = factory.newEntry();
        entry.setTitle("Some Event");
        entry.setContent("Foo bar");
        entry.setUpdated(new Date());
        entry.setId(factory.newUuidUri());
        entry.addAuthor("Dan Diephouse");

        return entry;
    }
}

```

You can also post Media entries quite simply. In this case it will take whatever your message payload is and post it to the collection as a media entry. You can supply your own Slug via configuration or by setting a property on the mule message.

**Post Message Payload as Media Entry**

```

<flow name="blobEventPublisher">
    <inbound-endpoint ref="quartz.in"/>
    <component class="org.mule.module.atom.event.BlobEventPublisher"/>

    <outbound-endpoint address="http://localhost:9002/events"
        exchange-pattern="request-response" mimeType="text/plain">
        <message-properties-transformer scope="outbound">
            <add-message-property key="Slug" value="Blob Event"/>
        </message-properties-transformer>
    </outbound-endpoint>
</flow>

```

## Route Filtering

The atom module also includes an `<atom:route-filter/>`. This allows ATOM requests to be filtered by request path and HTTP verb. The route attribute defines a type of URI Template loosely based on Ruby on Rails style Routes. For example:

```
"feed" or ":feed/:entry"
```

For reference, see the [Ruby On Rails routing](#).

For example, this filter can be used for content-based routing in Mule:

```
<flow name="customerService">
    <inbound-endpoint address="http://localhost:9002" exchange-pattern="request-response" />
    <choice>
        <when>
            <atom:route-filter route="/bar/:foo" />
            <outbound-endpoint address="vm://queue1" exchange-pattern="request-response" />
        </when>
        <when>
            <atom:route-filter route="/baz" verbs="GET,POST" />
            <outbound-endpoint address="vm://queue2" exchange-pattern="request-response" />
        </when>
    </choice>
</flow>
```

## Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

### Component

Represents an Abdera component.

#### Attributes of <component...>

Name	Type	Required	Default	Description
provider-ref	string	no		The id of the Atom provider that is defined as Spring bean.

#### Child Elements of <component...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

### Feed splitter

Will split the entries of a feed into single entry objects. Each entry will be a separate message in Mule.

#### Child Elements of <feed-splitter...>

Name	Cardinality	Description
------	-------------	-------------

### Filters

cache: Unexpected program error: java.lang.NullPointerException

#### Entry last updated filter

Will filter ATOM entry objects based on their last update date. This is useful for filtering older entries from the feed. This filter works only on Atom Entry objects not Feed objects.

#### Attributes of <entry-last-updated-filter...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

lastUpdate	string	no		The date from which to filter events from. Any entries that were last updated before this date will not be accepted. The date format is: yyyy-MM-dd hh:mm:ss, for example 2008-12-25 13:00:00. If only the date is important you can omit the time part. You can set the value to 'now' to set the date and time that the server is started. Do not set this attribute if you want to receive all available entries then any new entries going forward. This is the default behaviour and suitable for many scenarios.
acceptWithoutUpdateDate	boolean	no	true	Whether an entry should be accepted if it doesn't have a Last Update date set.

#### Child Elements of <entry-last-updated-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Feed last updated filter

Will filter the whole ATOM Feed based on its last update date. This is useful for processing a feed that has not been updated since a specific date.

This filter works only on Atom Feed objects.

Typically it is better to set the lastUpdated attribute on an inbound ATOM endpoint with splitFeed=false rather than use this filter, however, this filter can be used elsewhere in a flow.

#### Attributes of <feed-last-updated-filter...>

Name	Type	Required	Default	Description
lastUpdate	string	no		The date from which to filter events from. Any entries that were last updated before this date will not be accepted. The date format is: yyyy-MM-dd hh:mm:ss, for example 2008-12-25 13:00:00. If only the date is important you can omit the time part. You can set the value to 'now' to set the date and time that the server is started. Do not set this attribute if you want to receive all available entries then any new entries going forward. This is the default behaviour and suitable for many scenarios.
acceptWithoutUpdateDate	boolean	no	true	Whether a Feed should be accepted if it doesn't have a Last Update date set.

#### Child Elements of <feed-last-updated-filter...>

Name	Cardinality	Description
------	-------------	-------------

cache: Unexpected program error: java.lang.NullPointerException

#### Route filter

Allows ATOM requests to be filtered by request path and HTTP verb.

#### Attributes of <route-filter...>

Name	Type	Required	Default	Description
route	string	no		The URI request path made for an ATOM request. This matches against the path of the request URL. The route attribute defines a type of URI Template loosely based on Ruby on Rails style Routes. For example: "feed" or ":feed/:entry". For reference, see the Ruby On Rails routing <a href="http://guides.rubyonrails.org/routing.html">http://guides.rubyonrails.org/routing.html</a>
verbs	string	no		A comma-separated list of HTTP verbs that will be accepted by this filter. By default all verbs are accepted.

#### Child Elements of <route-filter...>

Name	Cardinality	Description
------	-------------	-------------

#### Transformers

cache: Unexpected program error: java.lang.NullPointerException

## Entry builder transformer

A transformer that uses expressions to configure an Atom Entry. The user can specify one or more expressions that are used to configure properties on the bean.

### Attributes of <entry-builder-transformer...>

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

### Child Elements of <entry-builder-transformer...>

Name	Cardinality	Description
entry-property	0..1	

cache: Unexpected program error: java.lang.NullPointerException

## Object to feed transformer

Transforms the payload of the message to a `org.apache.abdera.model.Feed` instance.

### Child Elements of <object-to-feed-transformer...>

Name	Cardinality	Description
------	-------------	-------------

## Schema

The schema for the ATOM module appears [here](#). Its structure is shown below.

Namespace "`http://www.mulesoft.org/schema/mule/atom`"

Targeting Schemas (1):

`mule-atom.xsd`

Targeting Components:

8 global elements, 1 local element, 6 complexTypes, 1 attribute group

Schema Summary	
<code>mule-atom.xsd</code>	<p>The Mule ATOM support makes it possible to integrate easily with Atom feeds and Atom Publishing Protocol servers via the Apache Abdera project.</p> <p>Target Namespace:</p> <p><a href="http://www.mulesoft.org/schema/mule/atom">http://www.mulesoft.org/schema/mule/atom</a></p> <p>Defined Components:</p> <p>8 global elements, 1 local element, 6 complexTypes, 1 attribute group</p> <p>Default Namespace-Qualified Form:</p> <p>Local Elements: qualified; Local Attributes: unqualified</p> <p>Schema Location:</p> <p><a href="http://www.mulesoft.org/schema/mule/atom/3.1/mule-atom.xsd">http://www.mulesoft.org/schema/mule/atom/3.1/mule-atom.xsd</a>; see XML source</p> <p>Imports Schemas (4):</p> <p><code>mule-schemadoc.xsd, mule.xsd, spring-beans-2.5.xsd, xml.xsd</code></p>
All Element Summary	
<code>component</code>	<p>Represents an Abdera component.</p> <p>Type: <code>atomComponentType</code></p> <p>Content: complex, 2 attributes, attr. wildcard, 7 elements</p> <p>Subst.Gr:may substitute for elements: <code>mule:abstract-component</code>, <code>mule:abstract-message-processor</code></p> <p>Defined: globally in <code>mule-atom.xsd</code>; see XML source</p> <p>Used: never</p>

entry-builder-transformer	A transformer that uses expressions to configure an Atom Entry. Type: <a href="#">entryBuilderTransformerType</a> Content: complex, 5 attributes, attr. wildcard, 1 element Subst.Gr:may substitute for elements: <a href="#">mule:abstract-transformer</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
entry-last-updated-filter	Will filter ATOM entry objects based on their last update date. Type: <a href="#">entryLastUpdateFilterType</a> Content: empty, 3 attributes, attr. wildcard Subst.Gr:may substitute for elements: <a href="#">mule:abstract-filter</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
entry-property	Type: <a href="#">anonymous complexType</a> Content: empty, 5 <a href="#">attributes</a> Defined: locally within <a href="#">entryBuilderTransformerType</a> complexType in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 <a href="#">attributes</a>
feed-last-updated-filter	Will filter the whole ATOM Feed based on its last update date. Type: <a href="#">feedLastUpdateFilterType</a> Content: empty, 3 attributes, attr. wildcard Subst.Gr:may substitute for elements: <a href="#">mule:abstract-filter</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
feed-splitter	Will split the entries of a feed into single entry objects. Type: <a href="#">mule:baseSplitterType</a> Content: complex, 1 attribute, attr. wildcard, 1 element Subst.Gr:may substitute for elements: <a href="#">mule:abstract-intercepting-message-processor</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
inbound-endpoint	Type: <a href="#">inboundEndpointType</a> Content: complex, 14 attributes, attr. wildcard, 12 elements Subst.Gr:may substitute for element <a href="#">mule:abstract-inbound-endpoint</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
object-to-feed-transformer	Transforms the payload of the message to a {{org.apache.abdera.model.Feed}} instance. Type: <a href="#">mule:abstractTransformerType</a> Content: empty, 5 attributes, attr. wildcard Subst.Gr:may substitute for elements: <a href="#">mule:abstract-transformer</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
route-filter	Allows ATOM requests to be filtered by request path and HTTP verb. Type: <a href="#">routeFilterType</a> Content: empty, 3 attributes, attr. wildcard Subst.Gr:may substitute for elements: <a href="#">mule:abstract-filter</a> , <a href="#">mule:abstract-message-processor</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: never
Complex Type Summary	
atomComponentType	Content:complex, 2 attributes, attr. wildcard, 7 elements Defined:globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Used: at 1 <a href="#">location</a>
entryBuilderTransformerType	Content: complex, 5 attributes, attr. wildcard, 1 <a href="#">element</a> Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Includes:definition of 1 <a href="#">element</a> Used: at 1 <a href="#">location</a>
entryLastUpdateFilterType	Content: empty, 3 attributes, attr. wildcard Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 <a href="#">attributes</a> Used: at 1 <a href="#">location</a>
feedLastUpdateFilterType	Content: empty, 3 attributes, attr. wildcard Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a> Includes:definitions of 2 <a href="#">attributes</a> Used: at 1 <a href="#">location</a>
inboundEndpointType	Content: complex, 14 <a href="#">attributes</a> , attr. wildcard, 12 elements Defined: globally in <a href="#">mule-atom.xsd</a> ; see <a href="#">XML source</a>

	<p>Includes:definitions of 3 attributes Used: at 1 location</p>
routeFilterType	<p>Content: empty, 3 attributes, attr. wildcard Defined: globally in <a href="#">mule-atom.xsd</a>; see <a href="#">XML source</a> Includes:definitions of 2 attributes Used: at 1 location</p>
Attribute Group Summary	
componentAttributes	<p>Content: 1 attribute Defined: globally in <a href="#">mule-atom.xsd</a>; see <a href="#">XML source</a> Includes:definition of 1 attribute Used: at 1 location</p>

XML schema documentation generated with DocFlex/XML SDK 1.8.1b6 using DocFlex/XML XSDDoc 2.2.1 template set. All content model diagrams generated by [Altova XMLSpy](#) via DocFlex/XML XMLSpy Integration.

## Javadoc API Reference

The Javadoc for this module can be found here: [atom](#)

## Maven

The ATOM Module can be included with the following dependency:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-atom</artifactId>
  <version>3.1.0</version>
</dependency>
```

## Extending this Module

### Best Practices

### Notes

**For more information see:**

- [Your First AtomPub Server](#)
- [Abdera Spring Integration](#).
- [Abdera User's Guide](#)

### Points of Etiquette When Polling Atom Feeds

1. Make use of HTTP cache. Send Etag and LastModified headers. Recognize 304 Not modified response. This way you can save a lot of bandwidth. Additionally some scripts recognize the LastModified header and return only partial contents (ie. only the two or three newest items instead of all 30 or so).
2. Don't poll RSS from services that supports RPC Ping (or other PUSH service, such as PubSubHubBub). I.e. if you're receiving PUSH notifications from a service, you don't have to poll the data in the standard interval — do it once a day to check if the mechanism still works or not (ping can be disabled, reconfigured, damaged, etc). This way you can fetch RSS only on receiving notification, not every hour or so.
3. Check the TTL (in RSS) or cache control headers (Expires in ATOM), and don't fetch until resource expires.
4. Try to adapt to frequency of new items in each single RSS feed. If in the past week there were only two updates in particular feed, don't fetch it more than once a day. AFAIR Google Reader does that.
5. Lower the rate at night hours or other time when the traffic on your site is low.

Your Rating: 

Results:  2 rates

## Drools Module Reference

### [Mule 3.2] Drools Module Reference

[ [Introduction](#) ] [ [Namespace and Syntax](#) ] [ [Features](#) ] [ [Usage](#) ] [ [Reference](#) ]

## Introduction

### Business Rules

In many applications, the business logic changes more frequently than the rest of the application code. A Rules Engine executes business rules that have been externalized from the rest of the application. This externalization of business logic makes the application more adaptable to change, and may even allow non-technical personnel to update the business logic without the intervention of a developer.

Rules engines introduce a lot of terminology which can be confusing for users not familiar with the Business Rules "world":

**Facts** are stored in a **working memory** (something like an in-memory database). A **fact** is a piece of data, e.g. "age = 40", or in an object-oriented rules engine it may be a Java object with properties.

**Rules** are defined in a **knowledge base**. A **rule** consists of conditions (which typically depend on facts in the working memory) and actions which are executed when the conditions are true (similar to an "if-then" statement). The action executed by a rule may change facts in the working memory which then cause other rules to fire.

### Complex Event Processing

Complex Event Processing (CEP) refers to an application where a stream or cloud of multiple events is analyzed by business rules in order to detect complex patterns or relationships within those events, generally in real-time.

### Considerations

If the business logic for your Mule application is relatively simple, you are probably best off implementing it using Mule's orchestration functionality such as [flows](#), [routers](#), and [custom components](#).

However, you might consider using Rules to model your business logic in the following cases:

- The business logic is overly complex or time-consuming when defined procedurally
- The business logic contains a lot of if-then-else statements
- The business logic changes often
- The business logic needs to be maintained by people who don't (or shouldn't) have access to the application itself (to recompile/redeploy it)

You also might consider modeling your business logic as a [process](#), depending on the nature of the algorithms involved.

### Drools

Drools is a best-of-breed open source [Rules Engine](#) which also offers Complex Event Processing.

This module provides integration between Mule and a Java Rules Engine, namely Drools.

### Namespace and Syntax

XML namespace:

```
xmlns:bpm "http://www.mulesoft.org/schema/mule/bpm"
```

XML Schema location:

```
http://www.mulesoft.org/schema/mule/bpm/3.2/mule-bpm.xsd
```

Syntax:

```
<bpm:drools />  
<bpm:rules rulesDefinition="myRules.drl" />
```

## Features

- Incoming Mule messages are asserted as new facts in Drools' working memory.
- Mule messages can be generated as a result of Rules firing.
- In CEP mode, Mule messages can be received as an event stream and used to trigger complex operations such as temporal reasoning and pattern detection.

To see Drools with Mule in action, take a look at the [CEP Example](#) (available in the full Mule distribution).



The Drools libraries are bundled with the Mule distribution. As of Mule 3.2, Drools 5.0 is the latest supported version.

## Usage

Import the XML schema for the general BPM module as follows:

### Import schema

```
xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
http://www.mulesoft.org/schema/mule/bpm/3.2/mule-bpm.xsd"
```



Since they are closely related, the BPM namespace/schema is shared between the Drools module and the BPM module.

Declare Drools as the Rules Engine to use with Mule:

### Declare Drools as the Rules Engine

```
<bpm:drools />
```



Just as with BPM, Rules integration in Mule is based on a pluggable interface such that any Java Rules Engine which implements this interface could be used instead of Drools. However, the only such implementation currently available is Drools.

Add Rules to your Mule application, generally (though not necessarily) preceded by an inbound-endpoint:

### Rules element

```
<bpm:rules rulesDefinition="myRules.drl" />
```

Create a Rules Definition file and (optionally) generate Mule messages from it:

### Rules Definition file

```
global org.mule.module.bpm.MessageService mule;

...cut...

rule "sudden drop"
when
    $st : StockTick( $dt : delta >= $threshold )
then
    mule.generateMessage("alerts", "Some alert message", null, MessageExchangePattern.ONE WAY);
end
```

## Configuration Examples

## Basic configuration

```
<mule ... xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
  xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
  http://www.mulesoft.org/schema/mule/bpm/3.2/mule-bpm.xsd" ...>

  <bpm:drools />

  <flow name="RulesInput">
    <jms:inbound-endpoint queue="input.queue" />
    <bpm:rules rulesDefinition="myRules.drl" />
  </flow>
</mule>
```

This is a simple config where incoming JMS messages on a queue () are inserted as facts into the Drools working memory ()�.

## CEP configuration

```
<mule ... xmlns:bpm="http://www.mulesoft.org/schema/mule/bpm"
  xsi:schemaLocation="http://www.mulesoft.org/schema/mule/bpm
  http://www.mulesoft.org/schema/mule/bpm/3.2/mule-bpm.xsd" ...>

  <spring:bean name="companies" class="org.mule.example.cep.CompanyRegistry" factory-method=
  "getCompanies" />

  <bpm:drools />

  <flow name="processStockTicks">
    <inbound-endpoint ref="stockTick" />
    <bpm:rules rulesDefinition="broker.drl"
      cepMode="true" entryPoint="StockTick stream"
      initialFacts-ref="companies" />
  </flow>
</mule>
```

Here a Collection of initial facts () is inserted into the working memory at startup. The Collection is provided by the factory-method of a Spring bean (). Drools is set to CEP mode (), which means that messages will be inserted as an Event Stream rather than Facts. The Entry Point for the Event Stream is also specified ()�.

## Reference

### Configuration Reference

cache: Unexpected program error: java.lang.NullPointerException

#### Rules

A service backed by a rules engine such as Drools.

#### Attributes of <rules...>

Name	Type	Required	Default	Description
rulesEngine-ref	string	no		A reference to the underlying Rules Engine.
rulesDefinition	string	yes		The resource containing the rules definition. This will be used to deploy the ruleset to the Rules Engine.
initialFacts-ref	string	no		A reference to a collection of initial facts to be asserted at startup.
cepMode	boolean	no		Are we using the knowledge base for CEP (Complex Event Processing)? (default = false)
entryPoint	string	no		Entry point for event stream (used by CEP).

#### **Child Elements of <rules...>**

Name	Cardinality	Description
annotations	0..1	
abstract-interceptor	0..1	A placeholder for an interceptor element.
interceptor-stack	0..1	A reference to a stack of interceptors defined globally.

#### **XML Schema**

Complete schema reference documentation.

#### **Maven**

If you are using Maven to build your application, use the following groupId/artifactIds to include the necessary modules:

```
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-bpm</artifactId>
</dependency>
<dependency>
  <groupId>org.mule.modules</groupId>
  <artifactId>mule-module-drools</artifactId>
</dependency>
```

Your Rating: 

Results:  0 rates

## **ATOM Module**

### **ATOM Module Reference**

[ Consuming Feeds and other Atom resources ] [ ETags ] [ Accessing the Feed itself ] [ Mixing Http and Atom endpoints ] [ Processing feeds without the Atom endpoint ] [ Accessing feeds over other Protocols ] [ Over JMS ] [ Over File ] [ Implementing an AtomPub Server ] [ Creating the Abdera Service Context ] [ Publishing to the Atom Component ] [ Route Filtering ]

The Mule ATOM support makes it possible to integrate easily with Atom feeds and Atom Publishing Protocol servers via the Apache Abdera project.

The module current supports:

- Consuming Atom feeds
- Publishing of Atom entries
- Server side AtomPub support as a Mule service

### **Consuming Feeds and other Atom resources**



The use of RSS endpoints is deprecated. RSS endpoints will be removed in Mule 3.2.0. Use the approach described in the Processing feeds without the Atom endpoint section.

One of the most common patterns is for integrating with Atom is polling for feed updates. With Mule this can be done quite easily. You must first write your class which receives an Atom Entry:

### EntryReceiver.java

```
package org.mule.module.atom.example;

import org.apache.abdera.model.Feed;

public class EntryReceiver {

    public void processEntry(Entry entry) throws Exception {
        //process the entry
    }
}
```

By default the Atom feed will be split into entries for you, so for each entry the component method will be invoked with the next entry in the feed. The flow configuration would look like:

```
<flow name="eventConsumer">
    <atom:inbound-endpoint address="http://localhost:9002/events" />
    <component class="org.mule.module.atom.event.EntryReceiver" />
</flow>
```

### ETags

The Atom Http endpoint respects ETags and the 304 Not Modified response code by default, so you don't need to worry about consuming unnecessary updates.

### **Accessing the Feed itself**

If you want access to the feed itself, it is available via a header on the current message called 'feed.object'.

If you want want to process the Feed in its entirety (not have it split into Entry objects) you can set the the following flag on the endpoint -

```
<atom:inbound-endpoint address="http://localhost:9002/events" splitFeed="false"/>
```

Now your component will only be invoked once for each feed change no matter how many entries were added or updated. The method on your component should expect a `org.apache.abdera.model.Feed` object or just a `org.w3c.dom.Document` if you want to process the XML model or even an `java.io.InputStream` or `java.lang.String` to get a raw representation.

### FeedReceiver.java

```
public void processFeed(@Payload Feed feed) throws Exception
{
    //do stuff
}
```

### Mixing Http and Atom endpoints

The `atom:endpoint` elements are a shortcut for creating a polling http endpoint and splitting a feed. If you are using another non-polling HTTP endpoint in your configuration you will need to configure a HTTP polling endpoint as well and reference it on your atom endpoint.

```

<http:connector name="HttpConnector"/>

<http:polling-connector name="PollingHttpConnector" pollingFrequency="1000" discardEmptyContent="false"/>

<flow name="eventConsumer">
    <atom:inbound-endpoint address="http://localhost:9002/events" connector-ref="PollingHttpConnector" />
    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

### **Processing feeds without the Atom endpoint**

For reference the follow example shows the explicit configuration for reading an atom feed without using the `atom:endpoint`.

```

<http:polling-connector name="PollingHttpConnector" pollingFrequency="1000" discardEmptyContent="false"/>

<flow name="feedConsumer">
    <http:inbound-endpoint address="http://rossmason.blogspot.com/feeds/posts/default" connector-ref="PollingHttpConnector">
        <atom:feed-splitter/>
        <atom:entry-last-updated-filter/>
    </http:inbound-endpoint>

    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

Note that the `atom:entry-last-updated-filter` should come after the `<atom:feed-splitter/>` since you need to split the feeds so that the filter can process them. Also note that we do not set a `lastUpdate` date on the filter which implies the default behaviour that all available entries will be read and then only new entries since the last read will be processed.

### **Accessing feeds over other Protocols**

You can process feeds and process them using other Mule connectors such as JMS, File or XMPP. To do this the atom feed needs to be served over the connector, i.e. an atom document is sent over JMS or polled from a file. The atom schema defines a `<atom:feed-splitter/>` that can split messages received from an endpoint.

#### **Over JMS**

```

<flow name="feedSplitterConsumer">
    <jms:inbound-endpoint queue="feed.in">
        <atom:feed-splitter/>
    </jms:inbound-endpoint>

    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

#### **Over File**

```

<flow name="feedSplitterConsumer">
    <file:inbound-endpoint path="${mule.working.dir}" pollingFrequency="1000" >
        <file:filename-wildcard-filter pattern="*.atom"/>
        <atom:feed-splitter/>
    </file:inbound-endpoint>

    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>

```

## Implementing an AtomPub Server

Abdera's service side component centers around the notion of a Provider. A Provider is manage's a service's Workspaces and Collections.

You can create an AtomPub service in Mule by using the `<atom:component/>` XML element and referencing an Abdera service context.

### Creating the Abdera Service Context

The following example shows how to create an Abdera context that builds a JCR repository to store atom entries. These entries can then be served as a feed.

```
abdera-config.xml

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:a="http://abdera.apache.org"
       xsi:schemaLocation="
           http://abdera.apache.org
           http://abdera.apache.org/schemas/abdera-spring.xsd
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <!-- atom -->
    <a:provider id="provider">

        <a:workspace title="JCR Workspace">
            <ref bean="jcrAdapter" />
        </a:workspace>

    </a:provider>

    <bean id="jcrRepository" class="org.apache.jackrabbit.core.TransientRepository" destroy-method="shutdown">

        <bean id="jcrAdapter"
              class="org.apache.abdera.protocol.server.adapters.jcr.JcrCollectionAdapter" init-method="initialize">
            <property name="author" value="Mule"/>
            <property name="title" value="Event Queue"/>
            <property name="collectionNodePath" value="entries"/>
            <property name="repository" ref="jcrRepository"/>
            <property name="credentials">
                <bean class="javax.jcr.SimpleCredentials">
                    <constructor-arg>
                        <value>username</value>
                    </constructor-arg>
                    <constructor-arg>
                        <value>password</value>
                    </constructor-arg>
                </bean>
            </property>
            <property name="href" value="events" />
        </bean>
    </bean>

</beans>
```

The `<a:provider>` creates an Abdera DefaultProvider and allows you to add Workspaces and Collections to it. This provider reference is used by the `<atom:component provider-ref="provider" />` in Mule to store any events sent to the component.

```
<flow name="atomPubEventStore">
    <inbound-endpoint address="http://localhost:9002"/>
    <atom:component provider-ref="provider" />
</flow>
```

## Publishing to the Atom Component

You may also want to publish Atom entries or media entries to the `atom:component` or to an external AtomPub collection. Here is a simple outbound endpoint which creates an Abdera Entry via the entry-builder-transformer and POSTs it to the AtomPub collection.

```
<outbound-endpoint address="http://localhost:9002/events" mimeType="application/atom+xml;type=entry"
connector-ref="HttpConnector">
    <atom:entry-builder-transformer>
        <atom:entry-property name="author" evaluator="string" expression="Ross Mason"/>
        <atom:entry-property name="content" evaluator="payload" expression=""/>
        <atom:entry-property name="title" evaluator="header" expression="title"/>
        <atom:entry-property name="updated" evaluator="function" expression="now"/>
        <atom:entry-property name="id" evaluator="function" expression="uuid"/>
    </atom:entry-builder-transformer>
</outbound-endpoint>
```

You could also create the Entry manually for more flexibility and send it as your Mule message payload. Here's a simple example of how to create an Abdera Entry:

```
package org.mule.providers.abdera.example;

import java.util.Date;

import org.apache.abdera.Abdera;
import org.apache.abdera.factory.Factory;
import org.apache.abdera.model.Entry;
import org.mule.transformer.AbstractTransformer;

public class EntryTransformer extend AbstractTransformer {
    public Object doTransform(Object src, String encoding) {
        Factory factory = Abdera.getInstance().getFactory();

        Entry entry = factory.newEntry();
        entry.setTitle("Some Event");
        entry.setContent("Foo bar");
        entry.setUpdated(new Date());
        entry.setId(factory.newUuidUri());
        entry.addAuthor("Dan Diephouse");

        return entry;
    }
}
```

You can also post Media entries quite simply. In this case it will take whatever your message payload is and post it to the collection as a media entry. You can supply your own Slug via configuration or by setting a property on the mule message.

```
<flow name="blobEventPublisher">
    <inbound-endpoint ref="quartz.in"/>
    <component class="org.mule.module.atom.event.BlobEventPublisher"/>

    <outbound-endpoint address="http://localhost:9002/events"
        exchange-pattern="request-response" mimeType="text/plain">
        <message-properties-transformer scope="outbound">
            <add-message-property key="Slug" value="Blob Event"/>
        </message-properties-transformer>
    </outbound-endpoint>
</flow>
```

## Route Filtering

The atom module also includes a `atom:route-filter`. This allows ATOM requests to be filtered by request path and HTTP verb. The route attribute defines a type of URI Template loosely based on Ruby on Rails style Routes. For example:

```
"feed" or ":feed/:entry"
```

For reference, see the [Ruby On Rails routing](#).

For example, this filter can be used for content-based routing in Mule:

```
<flow name="customerService">
    <inbound-endpoint address="http://localhost:9002" exchange-pattern="request-response" />
    <choice>
        <when>
            <atom:route-filter route="/bar/:foo" />
            <outbound-endpoint address="vm://queue1" exchange-pattern="request-response" />
        </when>
        <when>
            <atom:route-filter route="/baz" verbs="GET,POST" />
            <outbound-endpoint address="vm://queue2" exchange-pattern="request-response" />
        </when>
    </choice>
</flow>
```

For more information see:

- [Your First AtomPub Server](#)
- [Abdera Spring Integration](#)
- [Abdera User's Guide](#)

[ Consuming Feeds and other Atom resources ] [ ETags ] [ Accessing the Feed itself ] [ Mixing Http and Atom endpoints ] [ Processing feeds without the Atom endpoint ] [ Accessing feeds over other Protocols ] [ Over JMS ] [ Over File ] [ Implementing an AtomPub Server ] [ Creating the Abdera Service Context ] [ Publishing to the Atom Component ] [ Route Filtering ]

## Schema Reference

### Configuration options

cache: Unexpected program error: java.lang.NullPointerException

#### <feed-splitter ...>

Will split the entries of a feed into single entry objects. Each entry will be a separate message in Mule.

#### Attributes

Name	Type	Required	Default	Description

#### Child Elements

Name	Cardinality	Description

#### <entry-last-updated-filter ...>

Will filter ATOM entry objects based on their last update date. This is useful for filtering older entries from the feed. This filter works only on Atom Entry objects not Feed objects.

#### Attributes

Name	Type	Required	Default	Description

lastUpdate	string	no		The date from which to filter events from. Any entries that were last updated before this date will not be accepted. The date format is: yyyy-MM-dd hh:mm:ss, for example 2008-12-25 13:00:00. If only the date is important you can omit the time part. You can set the value to 'now' to set the date and time that the server is started. Do not set this attribute if you want to receive all available entries then any new entries going forward. This is the default behaviour and suitable for many scenarios.
acceptWithoutUpdateDate	boolean	no	true	Whether an entry should be accepted if it doesn't have a Last Update date set.

### Child Elements

Name	Cardinality	Description
------	-------------	-------------

### <component ...>

Represents an Abdera component.

### Attributes

Name	Type	Required	Default	Description
provider-ref	string	no		The id of the Atom provider that is defined as Spring bean.

### Child Elements

Name	Cardinality	Description
------	-------------	-------------

### <route-filter ...>

Allows ATOM requests to be filtered by request path and HTTP verb.

### Attributes

Name	Type	Required	Default	Description
route	string	no		The URI request path made for an ATOM request. This matches against the path of the request URL. The route attribute defines a type of URI Template loosely based on Ruby on Rails style Routes. For example: "feed" or ":feed/:entry". For reference, see the Ruby On Rails routing <a href="http://guides.rubyonrails.org/routing.html">http://guides.rubyonrails.org/routing.html</a>
verbs	string	no		A comma-separated list of HTTP verbs that will be accepted by this filter. By default all verbs are accepted.

### Child Elements

Name	Cardinality	Description
------	-------------	-------------

Your Rating: 

Results:  0 rates

# RSS Module Reference

## RSS Module Reference

[ Consuming Feeds ] [ ETags ] [ Accessing the Feed itself ] [ Mixing Http and RSS endpoints ] [ Accessing feeds over other Protocols ] [ Over JMS ] [ Over File ] [ Processing feeds without the RSS endpoint ] [ Schema Reference ] [ Configuration options ]

The Mule RSS support makes it possible to integrate easily with RSS feeds via the Rome project.

### Consuming Feeds



The use of RSS endpoints is deprecated. RSS endpoints will be removed in Mule 3.2.0. Use the approach described in the Mixing Http and RSS endpoints section.

One of the most common patterns is for integrating with RSS is polling for feed updates. With Mule this can be done easily. You must first write your class which receives an RSS Syndication Entry:

**EntryReceiver.java**

```
package org.mule.module.rss;

import com.sun.syndication.feed.synd.SyndEntry;

public class EntryReceiver
{
    public void readEntry(@Payload SyndEntry entry) throws Exception
    {
        //do stuff
    }
}
```

By default the RSS feed will be split into entries for you, so for each entry the component method will be invoked with the next entry in the feed. The flow configuration would look like:

```
<flow name="eventConsumer">
  <rss:inbound-endpoint address="http://localhost:9002/events" />
  <component class="org.mule.module.rss.EntryReceiver" />
</flow>
```

### ETags

The Atom Http endpoint respects ETags and the 304 Not Modified response code by default, so you don't need to worry about consuming unnecessary updates.

### Accessing the Feed itself

If you want access to the feed itself, it is available via a header on the current message called 'feed.object'. You could have this injected into your entry point method using annotations -

**EntryReceiver.java**

```
public void processEntry(@Payload Entry entry, @Expr("#[header:invocation:feed.object]") Feed feed)
throws Exception
```



The @Expr annotation is only available in the the Mule 3.0.1 SNAPSHOT or later.

If you want want to process the Feed in its entirety (not have it split into Entry objects) you can set the the following flag on the endpoint -

```
<rss:inbound-endpoint address="http://localhost:9002/events" splitFeed="false"/>
```

Now your component will only be invoked once for each feed change no matter how many entries were added or updated. The method on your component should expect a `com.sun.syndication.feed.synd.SyndFeed` object or just a `org.w3c.dom.Document` if you want to process the XML model or even an `java.io.InputStream` or `java.lang.String` to get a raw representation.

#### FeedReceiver.java

```
public void processFeed(@Payload SyndFeed feed) throws Exception
{
    //do stuff
}
```

### Mixing Http and RSS endpoints

The `rss:endpoint` elements are a shortcut for creating a polling http endpoint and splitting a feed. If you are using another non-polling HTTP endpoint in your configuration you will need to configure a HTTP polling endpoint as well and reference it on your atom endpoint.

```
<http:connector name="HttpConnector"/>

<http:polling-connector name="PollingHttpConnector" pollingFrequency="1000" discardEmptyContent="false"/>

<flow name="eventConsumer">
    <http:inbound-endpoint address="http://localhost:9002/events" connector-ref="PollingHttpConnector"/>
    <component class="org.mule.module.rss.EntryReceiver"/>
</flow>
```

### Accessing feeds over other Protocols

You can process feeds and process them using other Mule connectors such as JMS, File or XMPP. To do this the atom feed needs to be served over the connector, i.e. an rss document is sent over JMS or polled from a file. The rss schema defines a `<rss:feed-splitter/>` that can split messages received from an endpoint.

#### Over JMS

```
<flow name="feedSplitterConsumer">
    <jms:inbound-endpoint queue="feed.in">
        <rss:feed-splitter/>
    </jms:inbound-endpoint>

    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>
```

#### Over File

```
<flow name="feedSplitterConsumer">
    <file:inbound-endpoint path="${mule.working.dir}" pollingFrequency="1000" >
        <file:filename-wildcard-filter pattern="*.rss"/>
        <rss:feed-splitter/>
    </file:inbound-endpoint>

    <component class="org.mule.module.atom.event.EntryReceiver"/>
</flow>
```

## Processing feeds without the RSS endpoint

For reference the follow example shows the explicit configuration for reading an atom feed without using the `rss:endpoint`.

```
<http:polling-connector name="PollingHttpConnector" pollingFrequency="1000" discardEmptyContent="false"/>

<flow name="feedConsumer">
    <http:inbound-endpoint address="http://rossmason.blogspot.com/feeds/posts/default" connector-ref="PollingHttpConnector">
        <rss:feed-splitter/>
        <rss:entry-last-updated-filter/>
    </http:inbound-endpoint>

    <component class="org.mule.module.rss.EntryReceiver"/>
</flow>
```

Note that the `rss:entry-last-updated-filter` should come after the `<rss:feed-splitter/>` since you need to split the feeds so that the filter can process them. Also note that we do not set a `lastUpdate` date on the filter which implies the default behaviour that all available entries will be read and then only new entries since the last read will be processed.

## Schema Reference

### Configuration options

cache: Unexpected program error: java.lang.NullPointerException

#### <feed-splitter ...>

Will split the entries of a feed into single entry objects. Each entry will be a separate message in Mule.

##### Attributes

Name	Type	Required	Default	Description
------	------	----------	---------	-------------

##### Child Elements

Name	Cardinality	Description
------	-------------	-------------

#### <entry-last-updated-filter ...>

Will filter RSS entry objects based on their last update date. This is useful for filtering older entries from the feed. This filter works only on RSS SyndEntry objects not SyndFeed objects.

##### Attributes

Name	Type	Required	Default	Description
lastUpdate	string	no		The date from which to filter events from. Any entries that were last updated before this date will not be accepted. The date format is: yyyy-MM-dd hh:mm:ss, for example 2008-12-25 13:00:00. If only the date is important you can omit the time part. You can set the value to 'now' to set the date and time that the server is started. Do not set this attribute if you want to receive all available entries then any new entries going forward. This is the default behaviour and suitable for many scenarios.
acceptWithoutUpdateDate	boolean	no	true	Whether an entry should be accepted if it doesn't have a Last Update date set.

## Child Elements

Name	Cardinality	Description
------	-------------	-------------

[ Consuming Feeds ] [ ETags ] [ Accessing the Feed itself ] [ Mixing Http and RSS endpoints ] [ Accessing feeds over other Protocols ] [ Over JMS ] [ Over File ] [ Processing feeds without the RSS endpoint ] [ Schema Reference ] [ Configuration options ]

Your Rating: 

Results:  0 rates

# Expressions Configuration Reference

## Expressions Configuration Reference

[ Expression Attributes ] [ Expression Syntax ] [ Spring Property Placeholders and Expressions ] [ Expression Evaluator Reference ] [ Expression Enricher Reference ]

This page provides reference information for configuring expressions. For an introduction to where expressions can be used in Mule ESB, see [Using Expressions](#).

### Expression Attributes

The XML configuration elements in Mule that support expressions have three common attributes that define the expression to execute, the evaluator to use, and the option of defining a custom evaluator.

Attribute	Description
expression	The expression to evaluate. The syntax of this attribute will change depending on the evaluator being used.
evaluator	The expression evaluator to use. Expression evaluators must be registered with the <a href="#">ExpressionEvaluatorManager</a> before they can be used. Using the custom evaluator allows you to define your custom evaluator via the <code>custom-evaluator</code> attribute. Note that some evaluators such as Xpath, Groovy, and Bean are loaded from other Mule modules ( <a href="#">XML</a> and <a href="#">Scripting</a> , respectively). These modules must be on your classpath before their evaluators can be used.
custom-evaluator	The name of a custom evaluator to use. This evaluator must be registered in the local registry before it can be used.

Expressions can be used in a number of other scenarios such as filters, routing, and endpoints.

### Expression Syntax

There are two ways of specifying expressions depending on where the expression is being used. Typically, expression-based elements such as the expression transformer, expression filter, and expression-based routers such as the expression message splitter, will have specific attributes for `expression`, `evaluator`, and `custom-evaluator`. For example:

```
<expression-filter evaluator="header" expression="my-header!=null"/>
```

For these elements, you can set only a single expression for the element. When defining expressions for things like property values, you can set multiple expressions using the following syntax:

```
#[<evaluator>:<expression>]
```

The syntax requires that you precede an expression with `#[`, and then define the evaluator followed by a colon (`:`) and the expression to execute. Finally, you terminate the expression with a `]`. You can define multiple expressions as a string. For example:

```
<message-properties-transformer>
  <add-message-property name="GUID" value="#[string:#[xpath:/msg/header/ID]-#[xpath:/msg/body/@ref]]"/>
</message-properties-transformer>
```

For a list of available evaluators, see [Expression Evaluator Reference](#) below.

## Optional Values

As of Mule 2.2, you can use an asterisk to indicate an optional property in the expression. For example the following expression would indicate that foo and car must be present, but bar is optional:

```
#*[headers:foo,bar*,car]
```

or

```
#*[mule:message.headers(foo,bar*,car)]
```

## Spring Property Placeholders and Expressions

Spring and Mule have had long-time support for property placeholders that allow developers to inject static properties into their configuration when their application starts up. The property values are defined in Java property files that are passed into the framework on start up. The following example shows a properties file that defines configuration information used by a Mule endpoint:

### MyApplication.properties

```
web.proxy.host=192.168.2.2
web.proxy.port=8081
```

The Mule configuration file can embed these properties as placeholders.

```
<mule ...>
  <http:connector name="HttpConnector" proxyHostname="${web.proxy.host}" proxyPort=
"${web.proxy.port}"/>
</mule>
```

These property placeholders are resolved during the start-up phase of your application. Mule expressions are evaluated continuously for every message received or sent.

## Expression Evaluator Reference

Following are the default expression evaluators that are loaded at runtime. Not all expression evaluators are supported by every type of expression-based object. For example, the attachment evaluator is available to routers but not filters.

Name	Example	Comments
attachment	#*[attachment:supporting-docs]	Not supported by expression filters.
attachments	#*[attachments:attach1,attach2]	Returns a java.util.Map of attachments. Not supported by expression filters.
attachments-list	#*[attachments-list:attach1,attach2]	Returns a java.util.List of attachments objects. Not supported by expression filters. You can specify * to retrieve all attachments or a wildcard pattern to select attachments by name.
bean	#*[bean:fruit.isWashed]	The bean property expression. Use "." or "/" as element delimiter.
endpoint	#*[endpoint:myEP.address]	Use endpointName.property. Not supported by expression filters.
exception-type	#*[exception-type:java.lang.RuntimeException]	Matches an exception type. Only supported by expression filters.

function	<code>#function:datestamp:dd-MM-yyyy</code>	Performs a function: now, date, datestamp, systime, uuid, hostname, ip, or count. Not supported by expression filters.
groovy	<code>#groovy:payload.fruit.washed</code>	Evaluates the expression using the Groovy language.
header	<code>#header:Content-Type</code>	Evaluates the specified part of the message header.
headers	<code>#headers:Content-Type,Content-Length</code>	Returns a <code>java.util.Map</code> of headers. Not supported by expression filters. You can specify <code>#headers:*</code> to get all headers.
headers-list	<code>#headers-list:Content-Type,Content-Length</code>	Returns a <code>java.util.List</code> of header values. Not supported by expression filters.
json	<code>#json://fruit</code>	See <a href="#">JsonExpressionEvaluator</a> for expression syntax
json-node	<code>#json-node://fruit</code>	(As of Mule 3.1) Returns the node object from the json expression as is. See <a href="#">JsonExpressionEvaluator</a> for expression syntax.
jxpath	<code>#jxpath:/fruit</code>	JXPath expression that works on both XML/DOM and Beans.
map-payload	<code>#map-payload:key</code>	Returns a value from within a <code>java.util.Map</code> payload. Not supported by expression filters.
message	<code>#message:correlationId</code>	Available expressions are <code>id</code> , <code>correlationId</code> , <code>correlationSequence</code> , <code>correlationGroupSize</code> , <code>replyTo</code> , <code>payload</code> , <code>encoding</code> , and <code>exception</code> . Not supported by expression filters.
ognl	<code>#ognl:[MULE:0].equals(42)</code>	Set the <code>evaluator</code> attribute on the <code>&lt;expression-filter&gt;</code> element to <code>ognl</code> when specifying an <a href="#">OGNL filter</a> .
payload	<code>#payload:com.foo.RequiredType</code>	If expression is provided, it will be a class to be class loaded. The class will be the desired return type of the payload. See <code>getPayload(Class)</code> in <a href="#">MuleMessage</a> . Not supported by expression filters.
payload-type	<code>#payload:java.lang.String</code>	Matches the type of the payload. Only supported by expression filters.
process	<code>#process:processorName:valueToProcess</code>	<b>Since Mule 3.1.0</b> Invokes a message processor within an expression. This processor can be any component, transformer, custom processor, processor chain or flow. This evaluator is most useful when used with a nested expression that determines the value that will be processed by the referenced message processor.
regex	<code>#regex:the quick brown (.*)</code>	Only supported by expression filters.
string	<code>#string:Value is #[xpath://foo] other value is #[header:foo].</code>	Evaluates the expressions in the string.
variable	<code>#variable:variableName</code>	Used for retrieving values of flow variables.
wildcard	<code>#wildcard:*.txt</code>	Only supported by expression filters.
xpath	<code>#xpath://fruit</code>	The expression is an <a href="#">XPath expression</a> .
xpath-node	<code>#xpath-node://fruit</code>	(As of Mule 2.2) Returns the node object from the XPath expression as is.

## Expression Enricher Reference

### (From 3.1.0)

Following are the default expression enrichers that are loaded at runtime.

Name	Example	Comments
variable	<code>#variable:variableName</code>	Used for storing variable values in a flow.
header	<code>#header:Content-Type</code>	Adds/overwrites the specified message header.

Your Rating:  5 stars

Results:  2 rates

## Schema Documentation

### Schema Documentation

[ [Recent Changes to Mule Schemas](#) ] [ [JavaDoc-Style HTML Output](#) ] [ [UML Diagrams](#) ] [ [Elements, Attributes, Annotations, and Substitutions](#) ]

With Mule 3.0 comes a release of new Javadoc-style schema documentation, making it easier to familiarize yourself with all the elements and attributes permissible in xml configuration files.

#### Recent Changes to Mule Schemas

If you are familiar with previous versions of Mule, you may want to consult this list of [schema changes](#).

#### JavaDoc-Style HTML Output

Framed HTML documentation provides most detailed and easy accessible information about all Mule 3.0 schema components and interconnections between them.

<http://www.mulesoft.org/docs/site/3.0.1/schemadocs>

<http://www.mulesoft.org/docs/site/3.1.0/schemadocs>

#### UML Diagrams

Diagrams of many of the logical structures you will use to write Mule configurations.

#### Elements, Attributes, Annotations, and Substitutions

Navigate among exploded views of Mule schemas, viewing all objects - inherited or native - in a single location.

Your Rating:  5 stars

Results:  0 rates

## Notes on Mule 3.0 Schema Changes

The following changes have been made to schemas within Mule 3.0. For details on the schemas themselves, consult the [schemadoc](#) added with Mule 3.0.

### Core Mule

- The `synchronous` attribute has been removed in favor of the `exchange-pattern` attribute.
- [Dynamic endpoints](#) can be designated on most transports.
- Transformers, routers, filters, and other constructs that affect the processing of messages are now all [Message Processors](#) and can be used more flexibly inside a Mule configuration.
- In addition to services, message processing can be configured using [flows](#) and [configuration patterns](#).

### BPM

- BPM now includes the element `jbpmp-connector` to specify a jBpm connector.

### CXF

- The CXF schema is quite different now that CXF has been changed from a transport to a module. For details, see [Upgrading CXF from Mule 2](#).

### E-mail

- The e-mail connectors ([IMAP](#), [IMAPS](#), [POP3](#), and [POP3S](#)) support a new attribute `moveToFolder`, which specifies a folder to move e-mail messages to after they have been read.

### File

- The legacy filename parser, used for Mule 1 compatibility, has been removed. The expression filename parser is now the default.

## HTTP

- The [HTTP transport](#) defines a new transformer element `body-to-parameter-map-transformer`, which transforms the body of an HTTP message into a map.
- HTTP outbound endpoints now have the boolean attribute `follow-redirects`, which applies to HTTP GETs only.

## JMS

- [JMS](#) connectors have the new boolean attribute `embeddedMode`, which tells Mule to avoid using certain JMS features.

## Management

- In the [Management module](#), the element `jmx-server` now takes the boolean attribute `createRmiRegistry`, which specifies whether to create an RMI registry if none exists.

## Quartz

- Quartz endpoints have a new boolean attribute `stateful`, which determines whether the current job is persistent.

## Servlet

- Inbound and global [Servlet](#) endpoints have a new attribute `path`, which specifies the path of the servlet to bind to.

## TCP

- A custom implementation for the [TCP](#) protocol can now be specified as a reference to a Spring bean.

## VM

- The connector no longer takes the attribute `queueEvents`. Queuing behavior is now determined by the endpoint's exchange pattern.

## XML

- The [XML](#) now includes JAXB and [XQuery Support](#) transformers and extensions.
- The `xml-to-object-transformer` element can now be further configured to customize XStream behavior.

## Mule ESB 3 and Test API Javadoc

### Mule ESB 3 Javadoc

You can find the javadoc for Mule ESB 3.1.1 [here](#).

You can find the javadoc for Mule ESB 3.1.0 [here](#).

You can find the javadoc for Mule ESB 3.0 [here](#).

You can find the javadoc for the Mule ESB 3.1.1 Test API [here](#).

You can find the javadoc for the Mule ESB 3.1.0 Test API [here](#).

Your Rating:  Results:  0 rates

## Release and Migration Notes

### Release and Migration Notes

#### Release Notes

- [Mule 3.1 Release Notes](#)
- [Mule 3.0.1 Release Notes](#)
- [Mule 3.0.0 Release Notes](#)
- For a list of issues fixed in the latest enterprise release, see [Mule ESB EE 2.2.6 Release Notes](#).
- For a list of issues fixed in the previous enterprise release, see [Mule EE 2.2 Release Notes](#).
- To see a list of issues fixed in the latest community release, see [Mule 2 Release Notes](#).

- For a list of defects that were fixed in each point release, click [here](#).

## Migration Notes

- [Migration Guide](#)

Your Rating:  5 stars

Results:  0 rates

# Suggested Reading

## Suggested Reading

This topic relates to the most recent version of Mule ESB

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

## ESB

### [ESB Introduction Part 1](#)

This first article in this series described the basic concepts and role of the Enterprise Service Bus (ESB). It focuses on describing scenarios and issues for ESB deployment to support a Service-Oriented Architecture (SOA). One or more of these scenarios might apply to the SOA and ESB needs of your organization. - by Rick Robinson

### [ESB Introduction Part 2](#)

In Part 2 of this series on the Enterprise Service Bus (ESB), the author describes and analyzes some commonly observed scenarios in which ESBs and other Service-Oriented Architecture (SOA) solutions are implemented. - by Rick Robinson

### [ESB Introduction Part 3](#)

In the third installment of this series, the author examines possible solutions for the various scenarios outlined in Part 2. The ideas on the role of the Bus as explained in Part 1 provide the foundation for the scenarios. - by Rick Robinson

[The ESB Learning Guide](#) - everything you want to know about ESB is here.

## Enterprise Integration

### [Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions](#) - by Gregor Hohpe, Bobby Woolf

Provides a consistent vocabulary and visual notation framework to describe large-scale integration solutions across many technologies. It also explores in detail the advantages and limitations of asynchronous messaging architectures.

## SEDA

### [SEDA](#)

SEDA is an acronym for staged event-driven architecture, and decomposes a complex, event-driven application into a set of stages connected by queues. This design avoids the high overhead associated with thread-based concurrency models, and decouples event and thread scheduling from application logic. Mule uses ideas from SEDA to provide a highly scalable server.

[An Architecture for Highly Concurrent, Well-Conditioned Internet Services](#) - (PDF) Dissertation by Matt Welsh that introduces SEDA

## JBI

[The Sun JBI Site](#)

## Concurrency

[Java Concurrency in Practice](#) by Brian Goetz

[Concurrent Programming in Java: Design Principles and Patterns](#) by Doug Lea

## Open Source Development Process

Producing Open Source Software: How to Run a Successful Free Software Project by Karl Fogel

The Cathedral and the Bazaar by Eric Raymond

Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management by Martin Michlmayr

## Open Source Java

The Server Side

# Using the Mule RESTpack

## Using the Mule RESTpack

[ Understanding REST ] [ Architecting RESTful Applications ] [ Mule REST Connectors ]

The [Mule RESTpack](#) is geared toward helping you build RESTful applications with Mule ESB. REST is a very powerful concept that enables scalable, decentralized growth.

This topic relates to the most recent version of Mule ESB

To see the corresponding topic in a previous version of Mule ESB, click [here](#)

## Understanding REST

REST can be confusing. For an introduction to its advantages, disadvantages, and notes on using it, see [Making Sense of REST](#).

## Architecting RESTful Applications

Mule is well suited for building RESTful applications, as it can easily bridge between the web and nearly anything else in your enterprise. For more information on architecture considerations when using REST with Mule, see [Architecting RESTful HTTP applications](#).

## Mule REST Connectors

The Mule RESTpack uses a series of connectors for building RESTful services. This section describes each connector and links to more information on downloading and using them.

### HTTP Connector

The Mule [HTTP Transport Reference](#) contains most of the basic HTTP functionality that the connectors in the RESTpack build on, including:

- Client and Server HTTP transport
- Support for running over a Servlet
- Support for polling resources via the PollingHttpMessageReceiver

The Mule HTTP transport is included with your Mule installation.

### Apache Abdera Connector

The Mule Abdera connector makes it possible to integrate easily with Atom feeds and Atom Publishing Protocol servers via the Apache Abdera project. The connector supports consuming Atom feeds, publishing of Atom entries and server side AtomPub service.

To download and learn about using the Mule Abdera connector, go to the [MuleForge Abdera page](#).

## **Jersey JAX-RS Connector**

Jersey is the open source JAX-RS (JSR 311) reference implementation for building RESTful web services via simple annotations. The Mule Jersey connector enables developers to embed these JAX-RS services inside of Mule.

To learn about using the Mule Jersey connector, go to the [Jersey Module Reference](#).

## **Restlet Connector**

Restlet is an open source REST framework, providing a powerful abstraction for building and consuming RESTful services. The Mule Restlet connector facilitates the deployment and consumption of RESTful services using the Restlet APIs inside of Mule. In addition, the transport exploits Restlet's simple URL routing engine to provide RESTful routing of messages to Mule services.

To download and learn about using the Restlet connector, go to the [MuleForge Restlet page](#).