Xml Configuration

We support an XML deployment descriptor for configuring the ActiveMQ Message Broker. There are many things which can be configured such as

- transport connectors which consist of transport channels and wire formats
- network connectors using network channels or discovery agents
- persistence providers & locations
- custom message containers (such as last image caching etc)

So we decided that using XML would make this configuration much easier. From version 4.0 onwards we use XBean to perform the XML configuration.

For details of the XML see the Xml Reference

Be careful with broker names and URIs
Make sure you do not use any strange characters in the names of brokers as they are converted to URIs which do not allow things like underscores in them etc.

Examples

The default ActiveMQ configuration: current default config.

xml

From a binary distribution, from version 1.1 onwards there is an *activemq* script allowing you to run a Message Broker as a stand alone process from the command line easily providing the $ACTIVEMQ_HOME/bin directory is on your PATH.

AMQ 4.x

if myConfig.xml is in the classpath

or to use the file path system

AMQ 3.x

Or to use the default config file its just

If you have a source distribution you can run a broker using Maven specifying one of these configuration files as follows
under the assembly module run :

If your classpath is set up correctly you can achieve the same thing from the command line

Configuring embedded brokers

You can also use the XML Configuration to configure embedded brokers. For example using the JNDI configuration mechanism you can do the following
BrokerXmlConfigFromJNDITest
Or of you want to explicitly configure the embedded broker via Java code you can do the following
BrokerXmlConfigStartTest

User Submitted Configurations

We have a page which allows users to submit details of their configurations.

- User Submitted Configurations

Background

Since ActiveMQ has so many strategy pattern plugins for transports, wire formats, persistence and many other things, we wanted to leave the configuration format open so that you the developer can configure and extend ActiveMQ in any direction you wish.

So we use the Spring XML configuration file format, which allows any beans / POJOs to be wired together and configured. However often Spring's XML can be kinda verbose at times, so we have implemented an ActiveMQ extension to the Spring XML which knows about the common, standard ActiveMQ things you're likely to do (e.g. tags like connector, wireFormat, serverTransport, persistence) - but at any time you can fall back to the normal Spring way of doing things (with tags like bean, property etc).

To see documentation of the XML file we use or to get access to the XSD/DTD see the Xml Reference