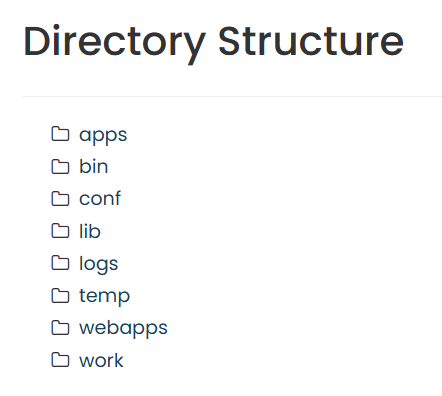
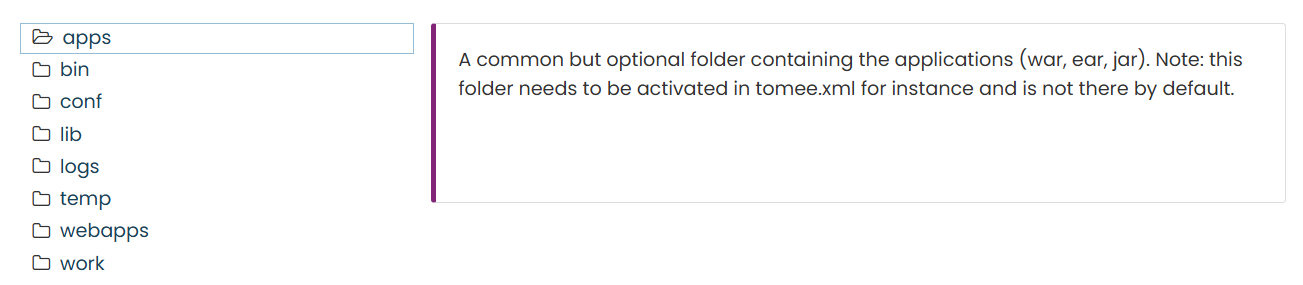
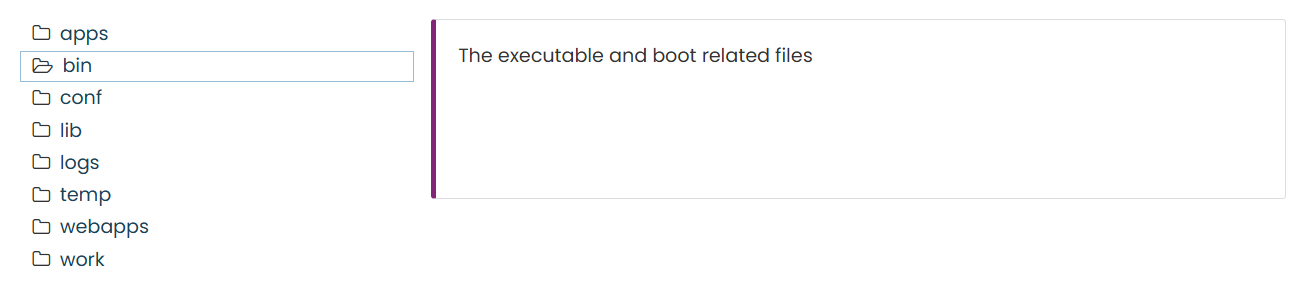
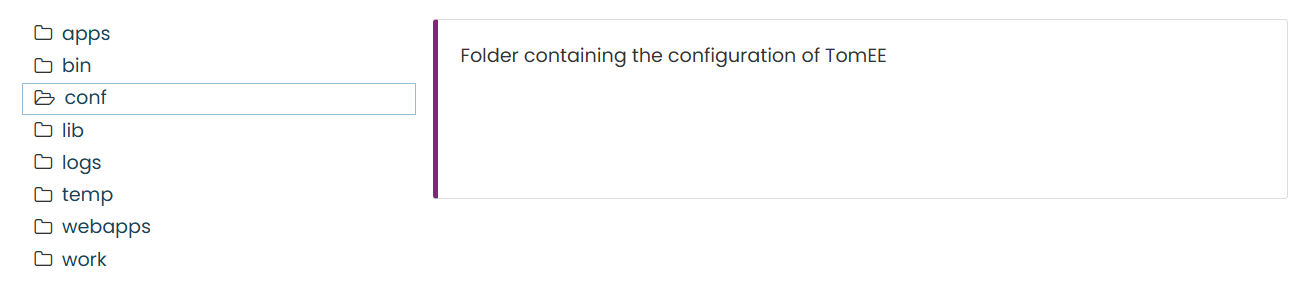
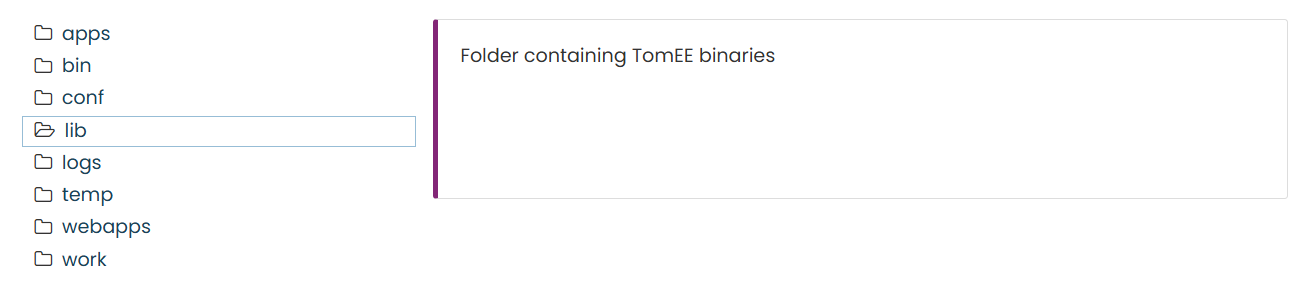
**Directory Structure and Configuration**

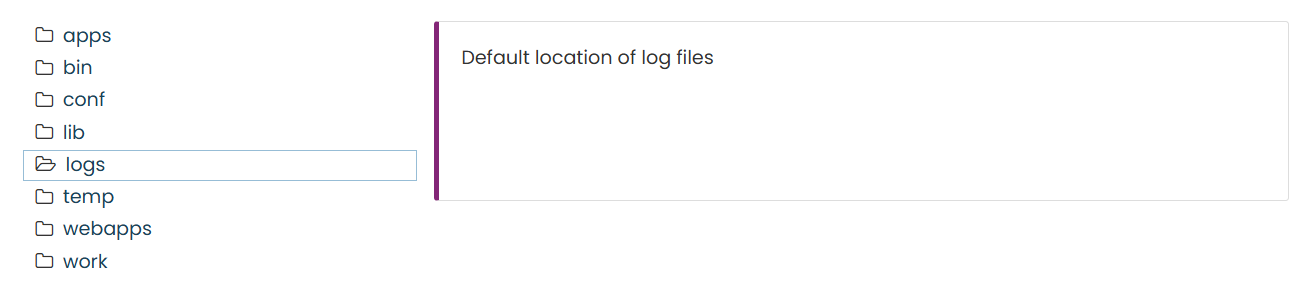


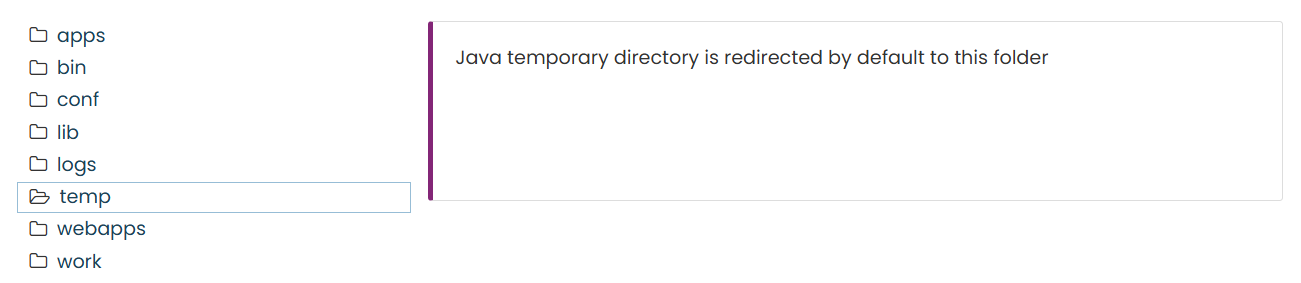


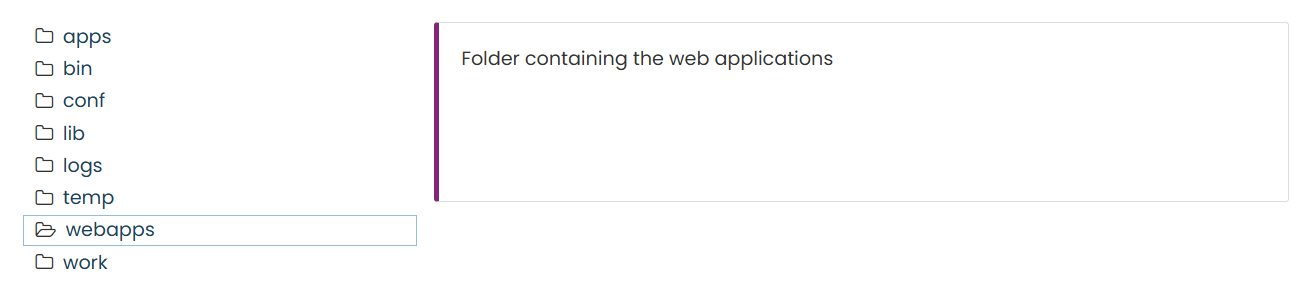


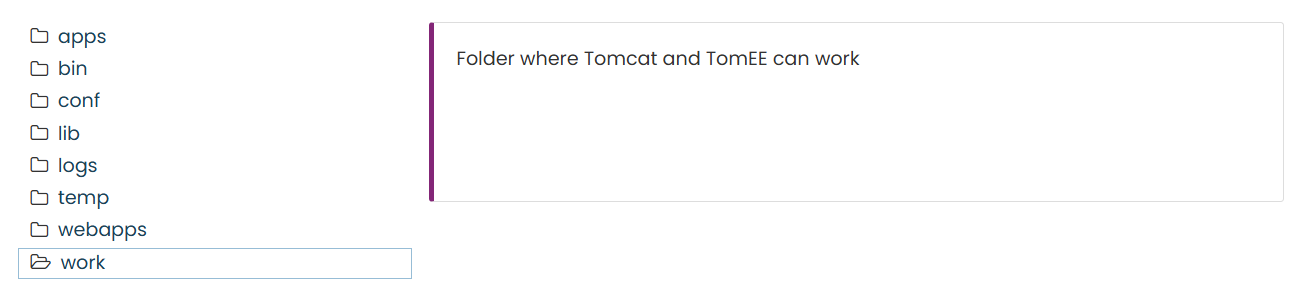












# Apache TomEE configuration

## Configuring Resources:

* Drivers are dropped into tomeeDir/lib
* Resources are configured in tomeeDir/conf/tomee.xml.
* The configurations take a very simple (XML+Property-file) syntax.
* Tag names match annotation names

For example,

@Resource DataSource moviesDatabase

is injected with the following resource:

<**Resource** id=**"moviesDatabase"** type=**"DataSource"**>

JdbcDriver org.hsqldb.jdbcDriver

JdbcUrl jdbc:mysql:localhost:3306/moviesdb

UserName sa

Password secret

JtaManaged true

</**Resource**>

# Deploying in TomEE

Deploying applications in TomEE is as simple as deploying them in Tomcat.

You could deploy your application in Eclipse just like how you would deploy with Tomcat.

Or you can simply package your application as a standard **WAR** file and copy it to the **[TomEE]/webapps** folder.

## Packaging

### **One archive**

The basic idea of this approach is that your Servlets and EJBs are together in your WAR file as one application.

* No classloader boundaries between Servlets and EJBs
* EJBs and Servlets can share all third-party libraries (like Spring!)
* No EAR required.
* Can put the web.xml and ejb-jar.xml in the same archive (the WAR file)
* EJBs can see Servlet classes and vice versa

### **Not quite J2EE (But it is Java EE 6)**

This is very different than J2EE or Java EE 5 as there are not several levels of separation and classloader hierarchy any more.  
This may take some getting used to and it is important to understand that this style of packaging is **not** J2EE compliant.  
You should not worry though, as it is an accepted feature of Java EE 6.

### **J2EE classloading rules:**

* You cannot ever have EJBs and Servlets in the same classloader.
* Three classloader minimum; a classloader for the ear, one for each ejb-jar, and one for each WAR file.
* Servlets can see EJBs, but EJBs cannot see Servlets.

To pull that off, J2EE has to kill you on packaging: - You cannot have EJB classes and Servlet classes in the same archive.

* You need at least three archives to combine Servlets and EJBs; 1 EAR containing 1 EJB jar and 1 Servlet WAR.
* Shared libraries must go in the EAR and be included in a specially formatted 'Class-Path' entry in the EAR’s MANIFEST file.

Critically speaking, forcing more than one classloader on an application is where J2EE "jumps the shark" for a large majority of people’s needs.

# Server Configuration

## Container

TomEE specific configuration (ie not inherited one from Tomcat) is based on properties. Therefore you can fully configure TomEE using properties in conf/system.properties. However for convenience it also provides a hybrid XML alternative a.k.a. conf/tomee.xml.

Container Configuration

Server

| **Name** | **Value** | **Description** |
| --- | --- | --- |
| openejb.embedded.remotable | bool | activate or not the remote services when available |
| .bind, <service prefix>.port, <service prefix>.disabled, <service prefix>.threads | host or IP, port, bool | override the host. Available for ejbd and httpejbd services (used by jaxws and jaxrs), number of thread to manage requests |
| openejb.embedded.initialcontext.close | LOGOUT or DESTROY | configure the hook called when closing the initial context. Useful when starting OpenEJB from a new InitialContext([properties]) instantiation. By default it simply logs out the logged user if it exists. DESTROY means clean the container. |
| jakarta.persistence.provider | string | override the JPA provider value |
| jakarta.persistence.transactionType | string | override the transaction type for persistence contexts |
| jakarta.persistence.jtaDataSource | string | override the JTA datasource value for persistence contexts |
| jakarta.persistence.nonJtaDataSource | string | override the non JTA datasource value for persistence contexts |
| openejb.descriptors.output | bool | dump memory deployment descriptors. Can be used to set complete metadata to true and avoid scanning when starting the container or to check the used configuration. |
| openejb.deployments.classpath.require.descriptor | CLIENT or EJB | can allow to filter what you want to scan (client modules or ejb modules) |
| openejb.descriptors.output.folder | path | where to dump deployement descriptors if activated. |
| openejb.strict.interface.declaration | bool | add some validations on session beans (spec validations in particular). false by default. |
| openejb.conf.file or openejb.configuration | string | OpenEJB configuration file path |
| openejb.debuggable-vm-hackery | bool | remove JMS informations from deployment |
| openejb.validation.skip | bool | skip the validations done when OpenEJB deploys beans |
| openejb.deployments.classpath.ear | bool | deploy the classpath as an ear |
| openejb.webservices.enabled | bool | activate or not webservices |
| openejb.validation.output.level | TERSE or MEDIUM or VERBOSE | level of the logs used to report validation errors |
| openejb.user.mbeans.list \* or a list of classes separated by , | list of mbeans to deploy automatically | openejb.deploymentId.format composition (+string) of {ejbName} {ejbType} {ejbClass} and {ejbClass.simpleName} default {ejbName}. The format to use to deploy ejbs. |
| openejb.deployments.classpath | bool | whether or not deploy from classpath |
| openejb.deployments.classpath.include and openejb.deployments.classpath.exclude | regex | regex to filter the scanned classpath (when you are in this case) |
| openejb.deployments.package.include and openejb.deployments.package.exclude | regex | regex to filter scanned packages |
| openejb.autocreate.jta-datasource-from-non-jta-one | bool | whether or not auto create the jta datasource if it doesn’t exist but a non jta datasource exists. Useful when using hibernate to be able to get a real non jta datasource. |
| openejb.altdd.prefix | string | prefix use for altDD (example test to use a test.ejb-jar.xml). |
| org.apache.openejb.default.system.interceptors | class names | list of interceptor (qualified names) separated by a comma or a space add these interceptor on all beans |
| openejb.jndiname.strategy.class | class name | an implementation of org.apache.openejb.assembler.classic.JndiBuilder.JndiNameStrategy |
| openejb.jndiname.failoncollision | bool | if a NameAlreadyBoundException is thrown or not when 2 EJBs have the same name |
| openejb.jndiname.format | string | composition of these properties: ejbType, ejbClass, ejbClass.simpleName, ejbClass.packageName, ejbName, deploymentId, interfaceType, interfaceType.annotationName, interfaceType.annotationNameLC, interfaceType.xmlName, interfaceType.xmlNameCc, interfaceType.openejbLegacyName, interfaceClass, interfaceClass.simpleName, interfaceClass.packageName default {deploymentId}{interfaceType.annotationName}. Change the name used for the ejb. |
| openejb.org.quartz.threadPool.class | class | qualified name which implements org.quartz.spi.ThreadPool the thread pool used by quartz (used to manage ejb timers) |
| openejb.localcopy | bool | default true. whether or not copy EJB arguments[/method/interface] for remote invocations. |
| openejb.cxf.jax-rs.providers | string | the list of the qualified name of the JAX-RS providers separated by comma or space. Note: to specify a provider for a specific service suffix its class qualified name by ".providers", the value follow the same rules. Note 2: default is a shortcut for jaxb and json providers. |
| openejb.wsAddress.format | string | composition of {ejbJarId}, ejbDeploymentId, ejbType, ejbClass, ejbClass.simpleName, ejbName, portComponentName, wsdlPort, wsdlService default /{ejbDeploymentId}. The WS name format. |
| org.apache.openejb.server.webservices.saaj.provider | axis2, sun or null | specified the saaj configuration |
| [<uppercase service name>.]<service id>.<name> or [<uppercase service name>.]<service id> | whatever is supported (generally string, int …​) | set this value to the corresponding service. example: [EnterpriseBean.]<ejb-name>.activation.<property>, [PERSISTENCEUNIT.]<persistence unit name>.<property>, [RESOURCE.]<name> |
| log4j.category.OpenEJB.options | DEBUG, INFO, …​ | active one OpenEJB log level. need log4j in the classpath |
| openejb.jmx.active | bool | activate (by default) or not the OpenEJB JMX MBeans |
| openejb.nobanner | bool | activate or not the OpenEJB banner (activated by default) |
| openejb.check.classloader | bool | if true print some information about duplicated classes |
| openejb.check.classloader.verbose | bool | if true print classes intersections |
| openejb.additional.exclude | string separated by comma | list of prefixes you want to exclude and are not in the default list of exclusion |
| openejb.additional.include | string separated by comma | list of prefixes you want to remove from thedefault list of exclusion |
| openejb.offline | bool | if true can create datasources and containers automatically |
| openejb.exclude-include.order | include-exclude or exclude-include | if the inclusion/exclusion should win on conflicts (intersection) |
| openejb.log.color | bool | activate or not the color in the console in embedded mode |
| openejb.log.color.<level in lowercase> | color in uppercase | set a color for a particular level. Color are BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, DEFAULT. |
| tomee.serialization.class.blacklist | string | default list of packages/classnames excluded for EJBd deserialization (needs to be set on server and client sides). Please see the description of Ejbd Transport for details. |
| tomee.serialization.class.whitelist | string | default list of packages/classnames allowed for EJBd deserialization (blacklist wins over whitelist, needs to be set on server and client sides). Please see the description of Ejbd Transport for details. |
| tomee.remote.support | boolean | if true /tomee webapp is auto-deployed and EJBd is active (true by default for 1.x, false for 7.x excepted for tomee maven plugin and arquillian) |
| openejb.crosscontext | bool | set the cross context property on tomcat context (can be done in the traditionnal way if the deployment is don through the webapp discovery and not the OpenEJB Deployer EJB) |
| openejb.jsessionid-support | bool | remove URL from session tracking modes for this context (see jakarta.servlet.SessionTrackingMode) |
| openejb.myfaces.disable-default-values | bool | by default TomEE will initialize myfaces with some its default values to avoid useless logging |
| openejb.web.xml.major | int | major version of web.xml. Can be useful to force tomcat to scan servlet 3 annotatino when deploying with a servlet 2.x web.xml |
| tomee.jaxws.subcontext | string | sub context used to bind jaxws web services, default is webservices |
| openejb.servicemanager.enabled | bool | run all services detected or only known available services (WS and RS |
| tomee.jaxws.oldsubcontext | bool | wether or not activate old way to bind jaxws webservices directly on root context |
| openejb.modulename.useHash | bool | add a hash after the module name of the webmodule if it is generated from the webmodule location, it avoids conflicts between multiple deployment (through ear) of the same webapp. Note: it disactivated by default since names are less nice this way. |
| openejb.session.manager | qualified name (string) | configure a session managaer to use for all contexts |
| tomee.tomcat.resource.wrap | bool | wrap tomcat resources (context.xml) as tomee resources if possible (true by default) |
| tomee.tomcat.datasource.wrap | bool | same as tomee.tomcat.resource.wrap for datasource (false by default). Note that setting it to true will create tomee datasources but can have the side effect to create twice singleton resources |
| openejb.environment.default | bool | should default JMS resources be created or not, default to false to ensure no port is bound or multiple resources are created and completely uncontrolled (doesn’t apply to datasources etc for compatibility). For tests only! |

* Client

| **Name** | **Value** | **Description** |
| --- | --- | --- |
| openejb.client.identityResolver | implementation of org.apache.openejb.client.IdentityResolver | default org.apache.openejb.client.JaasIdentityResolver. The class to get the client identity. |
| openejb.client.connection.pool.timeout or openejb.client.connectionpool.timeout | int (ms) | the timeout of the client |
| openejb.client.connection.pool.size or openejb.client.connectionpool.size | int | size of the socket pool |
| openejb.client.keepalive | int (ms) | the keepalive duration |
| openejb.client.protocol.version | string | Optional legacy server protocol compatibility level. Allows 4.6.x clients to potentially communicate with older servers. OpenEJB 4.5.2 and older use version "3.1", and 4.6.x currently uses version "4.6" (Default). This does not allow old clients to communicate with new servers prior to 4.6.0 |
| tomee.serialization.class.blacklist | string | default list of packages/classnames excluded for EJBd deserialization (needs to be set on server and client sides). Please see the description of Ejbd Transport for details. |
| tomee.serialization.class.whitelist | string | default list of packages/classnames allowed for EJBd deserialization (blacklist wins over whitelist, needs to be set on server and client sides). Please see the description of Ejbd Transport for details. |

# Resources

In TomEE resources are mainly "singleton" (understood as defined once per server or application). Technically it can be anything but you will probably meet more Datasources than other type of resources.

Most resources will be created automatically if there is no matching resources - by name and type - when an injection will be found. To avoid that use openejb.offline property and set it to true.

## Definition a resource: how does it work?

Before all let see how properties syntax is equivalent to XML one (system.properties and tomee.xml typically).

Properties syntax uses dot notation to represent setters/properties which are plain properties in XML syntax and a URL syntax with query parameters to define the resource where it is directly the resource and tag attributes in XML. Finally the id is an attribute in XML and the key of the resource definition in properties.

Let see it with a sample, both delcarations are the same:

myDataSource = new://Resource?type=DataSource

myDataSource.JdbcUrl = jdbc:hsqldb:mem:site

myDataSource.UserName = sa

<**Resource** id=**"myDataSource"** type=**"DataSource"**>

JdbcUrl = jdbc:hsqldb:mem:site

UserName = sa

</**Resource**>

One started you can get injected any resource using @Resource:

@Resource(name = **"myDataSource"**)

**private** DataSource dataSource;

## Factory syntax

Here are the attributes of a resource:

| **Name** | **Optional** | **Description** |
| --- | --- | --- |
| id | false | name of the resource, will match openejb:Resource/id in JNDI tree. |
| provider | true | define a default resource definition using service-jar.xml |
| class-name | true | specify which class to instantiate |
| factory-name | true | specify which method to invoke on the class-name when specified to create the resource |
| properties-provider | true | a class responsible to provide to tomee the properties to use, it can have a property serviceId to know which resource it is. |
| classpath | true | a classpath to use to create the resource. Note: if not implementing an interface the resource will be isolated from the applications. |
| aliases | true | other names for the resource, allows for instance to share the same pool for a datasource used with multiple names in applications. |
| post-construct/pre-destroy | true | methods called when creating/destroying the resources. |
| Lazy | true | for resources set them to be created when first accessed and not when deployed |

TomEE supports some implicit properties for resources but sometimes you just want to fully control the resource and not use implicit properties which can be affected to a property which doesn’t expect such a value (typically the case if you create a custom Oracle datasource). For such case you can set SkipImplicitAttributes property to true and your resource will ignore implicit properties.

Implicit properties are:

| **Name** | **Description** |
| --- | --- |
| transactionManager | The JTA transaction manager |
| ServiceId | the "id" of the resource (its name) |

In the same spirit you can skip properties fallback using SkipPropertiesFallback and setting it to true. It typically avoids to fallback all unset properties (no matching property found) to a Properties instance and set it if one matching property is found. In Oracle case for instance it matches the connection properties which can have side effects.

### **Value ciphering**

The propertie values support ciphering using the syntax cipher:{algorithm}:{cipheredValue}, for instance cipher:Static3DES:xMH5uM1V9vQzVUv5LG7YLA== will be read as Passw0rd. Ciphers can be computed using tomee.sh script: ${tomee.home}/bin/tomee.sh cipher Passw0rd.

## Common Resources

### **DataSources**

DataSources have defaults for all values and a default datasource can be provided automatically but if you want to configure it here are the common properties:

You can set the boolean JtaManaged to false if you don’t want your datasource to be using JTA - if you manage transactions yourself.

Then other configurations are linked the pool the datasource is using. By default TomEE uses tomcat-jdbc but we also provide commons-dbcp (2 for TomEE 7.x and 1 for TomEE 1.x). The properties are then the related configurations with these particular entries we try to keep in sync for both:

| **Name** | **Description** |
| --- | --- |
| JdbcDriver | the jdbc driver of the datasource |
| JdbcUrl | the jdbc url of the datasource |
| Username | the user to use |
| Password | the password of the user |

#### **Password and ciphering**

DataSource were the first resource to support password ciphering. Originally it was another property which is still supported. It is called PasswordCipher. Its value is the ciphering algorithm and it affects the password value. However cipher:xxx is still supported on Password value. Default PasswordCipher being PlainText it behaves as no ciphering is in place by default.

Sample:

ds = new://Resource?type=javax.sql.DataSource

# our password is "Passw0rd"

ds.Password = xMH5uM1V9vQzVUv5LG7YLA==

ds.PasswordCipher = Static3DES

#### **Advanced DataSource configuration**

TomEE also provides few utilities you can add in DataSource properties:

| **Name** | **Description** |
| --- | --- |
| LogSql | Should SQL be logged (using TomEE logger) |
| LogSqlPackages | if set the logging will show the matching packages (separated by comma) inline when logging the query, allows to know where a query comes from |
| Flushable | if true the datasource can be casted as a Flushable to recreate the pool |
| ResetOnError | if a SQLException happens the pool is automatically recreated. Configuration is either "true" to do it each time an exception occurs, x or retry(x) to do it and retry until maximum x times |
| ResetOnErrorMethods | which methods are handled by ResetOnError |
| TomEEProxyHandler | Custom InvocationHandler wrapping the datasource calls |
| DataSourceCreator | which pool to use, dbcp, tomcat, dbcp-alternative (DBCP and TomEE proxying instead of DBCP JTA integration), simple (no pooling) |

#### **DataSource and JTA**

JtaManaged determines wether or not this data source should be JTA managed or user managed. If set to 'true' it will automatically be enrolled in any ongoing transactions. Calling begin/commit/rollback or setAutoCommit on the datasource or connection will not be allowed. If you need to perform these functions yourself, set JtaManaged to false

#### **DataSource and JPA**

In terms of JPA persistence.xml:

* JtaManaged=true can be used as a 'jta-data-source'
* JtaManaged=false can be used as a 'non-jta-data-source'

## ActiveMQResourceAdapter

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ActiveMQResourceAdapter"**>

BrokerXmlConfig = broker:(tcp://localhost:61616)?useJmx=false

ServerUrl = vm://localhost?waitForStart=20000&async=true

DataSource = Default Unmanaged JDBC Database

StartupTimeout = 10 seconds

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=ActiveMQResourceAdapter

Foo.BrokerXmlConfig = broker:(tcp://localhost:61616)?useJmx=false

Foo.ServerUrl = vm://localhost?waitForStart=20000&async=true

Foo.DataSource = Default Unmanaged JDBC Database

Foo.StartupTimeout = 10 seconds

### **Configuration**

#### **BrokerXmlConfig**

Broker configuration URI as defined by ActiveMQ see <http://activemq.apache.org/broker-configuration-uri.html> BrokerXmlConfig xbean:file:conf/activemq.xml - Requires xbean-spring.jar and dependencies

#### **ServerUrl**

Broker address

#### **DataSource**

DataSource for persistence messages

#### **StartupTimeout**

How long to wait for broker startup

## jakarta.jms.ConnectionFactory

An ActiveMQ (JMS) connection factory.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"jakarta.jms.ConnectionFactory"**>

ResourceAdapter = Default JMS Resource Adapter

TransactionSupport = xa

PoolMaxSize = 10

PoolMinSize = 0

ConnectionMaxWaitTime = 5 seconds

ConnectionMaxIdleTime = 15 Minutes

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=jakarta.jms.ConnectionFactory

Foo.ResourceAdapter = Default JMS Resource Adapter

Foo.TransactionSupport = xa

Foo.PoolMaxSize = 10

Foo.PoolMinSize = 0

Foo.ConnectionMaxWaitTime = 5 seconds

Foo.ConnectionMaxIdleTime = 15 Minutes

### **Configuration**

#### **ResourceAdapter**

An ActiveMQ (JMS) resource adapter.

#### **TransactionSupport**

Specifies if the connection is enrolled in global transaction allowed values: xa, local or none. Default to xa.

#### **PoolMaxSize**

Maximum number of physical connection to the ActiveMQ broker.

#### **PoolMinSize**

Minimum number of physical connection to the ActiveMQ broker.

#### **ConnectionMaxWaitTime**

Maximum amount of time to wait for a connection.

#### **ConnectionMaxIdleTime**

Maximum amount of time a connection can be idle before being reclaimed.

## jakarta.jms.Queue

An ActiveMQ (JMS) queue.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"jakarta.jms.Queue"**>

# not set means id

destination =

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=jakarta.jms.Queue

# not set means id

Foo.destination =

### **Configuration**

#### **destination**

Specifies the name of the queue

## jakarta.jms.Topic

An ActiveMQ (JMS) topic.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"jakarta.jms.Topic"**>

# not set means id

destination =

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=jakarta.jms.Topic

# not set means id

Foo.destination =

### **Configuration**

#### **destination**

Specifies the name of the topic

## org.omg.CORBA.ORB

|  |  |
| --- | --- |
|  | to use it you need to add an implementation of corba. |

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"org.omg.CORBA.ORB"** />

Declarable in properties via

Foo = new://Resource?type=org.omg.CORBA.ORB

## jakarta.mail.Session

A mail session.

Declarable in tomee.xml via

<**Resource** id=**"mail/mysession"** type=**"jakarta.mail.Session"**>

mail.transport.protocol = smtp

mail.smtp.host = smtp.provider.com

mail.smtp.auth = true

mail.smtp.starttls.enable = true

mail.smtp.port = 587

mail.smtp.user = user@provider.com

password = abcdefghij

</**Resource**>

Declarable in properties via

mail/mysession = new://Resource?type=jakarta.mail.Session

mail/mysession.mail.transport.protocol = smtp

mail/mysession.mail.smtp.host = smtp.provider.com

mail/mysession.mail.smtp.auth = true

mail/mysession.mail.smtp.starttls.enable = true

mail/mysession.mail.smtp.port = 587

mail/mysession.mail.smtp.user = user@provider.com

mail/mysession.password = abcdefghij

The properties are jakarta.mail.Session ones with the addition of useDefault which specifies if getDefaultInstance() or getInstance is used to create the session. getDefaultInstance() will ensure that several calls are done with the same configuration and return the same instance. For tomee it is likely better to rely on getInstance()(ie keep useDefault to false) and use aliases option of the resource to define an alias if you need to share the same instance accross multiple names.

## ManagedExecutorService

A concurrency utility for EE executor service.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ManagedExecutorService"**>

Core = 5

Max = 25

KeepAlive = 5 s

Queue = 15

ThreadFactory = org.apache.openejb.threads.impl.ManagedThreadFactoryImpl

Lazy = true

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=ManagedExecutorService

Foo.Core = 5

Foo.Max = 25

Foo.KeepAlive = 5 s

Foo.Queue = 15

Foo.ThreadFactory = org.apache.openejb.threads.impl.ManagedThreadFactoryImpl

Foo.Lazy = true

### **Configuration**

#### **Core**

The pool core size.

#### **Max**

The pool max size.

#### **KeepAlive**

The thread keep alive time (in duration format)

#### **Queue**

The queue type size.

#### **ThreadFactory**

The thread factory implementation class.

#### **Lazy**

If set to true the pool is created when first accessed otherwise it is created at startup.

## ManagedScheduledExecutorService

Inherit from ManagedExecutorService and adds scheduling abilities.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ManagedScheduledExecutorService"**>

Core = 5

ThreadFactory = org.apache.openejb.threads.impl.ManagedThreadFactoryImpl

Lazy = true

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=ManagedScheduledExecutorService

Foo.Core = 5

Foo.ThreadFactory = org.apache.openejb.threads.impl.ManagedThreadFactoryImpl

Foo.Lazy = true

### **Configuration**

See ManagedExecutorService.

## ManagedThreadFactory

A thread factory for a ManagedExecutorService.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ManagedThreadFactory"**>

Prefix = openejb-managed-thread-

Lazy = true

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=ManagedThreadFactory

Foo.Prefix = openejb-managed-thread-

Foo.Lazy = true

### **Configuration**

#### **Prefix**

The thread prefix (suffixed with thread id).

## ContextService

A concurrency utilities for JavaEE context service. It allows to create contextual proxies (inheriting from security, classloader…​contexts).

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ContextService"** />

Declarable in properties via

Foo = new://Resource?type=ContextService

## JndiProvider: inject remote clients

A thread factory for a ManagedExecutorService. Default implementation is org.apache.openejb.threads.impl.ManagedThreadFactoryImpl.

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ManagedThreadFactory"**>

Prefix = openejb-managed-thread-

Lazy = true

</**Resource**>

Declarable in properties via

Foo = new://Resource?type=ManagedThreadFactory

Foo.Prefix = openejb-managed-thread-

Foo.Lazy = true

### **Configuration**

#### **Prefix**

The thread prefix (suffixed with thread id).

## ContextService

A concurrency utilities for JavaEE context service. It allows to create contextual proxies (inheriting from security, classloader…​contexts).

Declarable in tomee.xml via

<**Resource** id=**"Foo"** type=**"ContextService"** />

Declarable in properties via

Foo = new://Resource?type=ContextService

# Resources

All containers will be created automatically - which means you don’t need to define them if you don’t need to tune their configuration - when a bean of their type if found.

To avoid that use openejb.offline property and set it to true.

## @Stateless

A @Stateless container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"STATELESS"**>

AccessTimeout = 30 seconds

MaxSize = 10

MinSize = 0

StrictPooling = true

MaxAge = 0 hours

ReplaceAged = true

ReplaceFlushed = false

MaxAgeOffset = -1

IdleTimeout = 0 minutes

GarbageCollection = false

SweepInterval = 5 minutes

CallbackThreads = 5

CloseTimeout = 5 minutes

UseOneSchedulerThreadByBean = false

EvictionThreads = 1

</**Container**>

Declarable in properties via

Foo = new://Container?type=STATELESS

Foo.AccessTimeout = 30 seconds

Foo.MaxSize = 10

Foo.MinSize = 0

Foo.StrictPooling = true

Foo.MaxAge = 0 hours

Foo.ReplaceAged = true

Foo.ReplaceFlushed = false

Foo.MaxAgeOffset = -1

Foo.IdleTimeout = 0 minutes

Foo.GarbageCollection = false

Foo.SweepInterval = 5 minutes

Foo.CallbackThreads = 5

Foo.CloseTimeout = 5 minutes

Foo.UseOneSchedulerThreadByBean = false

Foo.EvictionThreads = 1

### **Configuration**

#### **AccessTimeout**

Specifies the time an invokation should wait for an instance of the pool to become available.

After the timeout is reached, if an instance in the pool cannot be obtained, the method invocation will fail.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

Any usage of the jakarta.ejb.AccessTimeout annotation will override this setting for the bean or method where the annotation is used.

#### **MaxSize**

Specifies the size of the bean pools for this stateless SessionBean container. If StrictPooling is not used, instances will still be created beyond this number if there is demand, but they will not be returned to the pool and instead will be immediately destroyed.

#### **MinSize**

Specifies the minimum number of bean instances that should be in the pool for each bean. Pools are prefilled to the minimum on startup. Note this will create start order dependencies between other beans that also eagerly start, such as other @Stateless beans with a minimum or @Singleton beans using @Startup. The start order.

The minimum pool size is rigidly maintained. Instances in the minimum side of the pool are not eligible for IdleTimeout or GarbageCollection, but are subject to MaxAge and flushing.

If the pool is flushed it is immediately refilled to the minimum size with MaxAgeOffset applied. If an instance from the minimum side of the pool reaches its MaxAge, it is also immediately replaced. Replacement is done in a background queue using the number of threads specified by CallbackThreads.

#### **StrictPooling**

StrictPooling tells the container what to do when the pool reaches it’s maximum size and there are incoming requests that need instances.

With strict pooling, requests will have to wait for instances to become available. The pool size will never grow beyond the the set MaxSize value. The maximum amount of time a request should wait is specified via the AccessTimeout setting.

Without strict pooling, the container will create temporary instances to meet demand. The instances will last for just one method invocation and then are removed.

Setting StrictPooling to false and MaxSize to 0 will result in no pooling. Instead instances will be created on demand and live for exactly one method call before being removed.

#### **MaxAge**

Specifies the maximum time that an instance should live before it should be retired and removed from use. This will happen gracefully. Useful for situations where bean instances are designed to hold potentially expensive resources such as memory or file handles and need to be periodically cleared out.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as 1 hour and 27 minutes and 10 seconds

#### **ReplaceAged**

When ReplaceAged is enabled, any instances in the pool that expire due to reaching their MaxAge will be replaced immediately so that the pool will remain at its current size. Replacement is done in a background queue so that incoming threads will not have to wait for instance creation.

The aim of his option is to prevent user requests from paying the instance creation cost as MaxAge is enforced, potentially while under heavy load at peak hours.

Instances from the minimum side of the pool are always replaced when they reach their MaxAge, this setting dictates the treatment of non-minimum instances.

#### **ReplaceFlushed**

When ReplaceFlushed is enabled, any instances in the pool that are flushed will be replaced immediately so that the pool will remain at its current size. Replacement is done in a background queue so that incoming threads will not have to wait for instance creation.

The aim of his option is to prevent user requests from paying the instance creation cost if a flush performed while under heavy load at peak hours.

Instances from the minimum side of the pool are always replaced when they are flushed, this setting dictates the treatment of non-minimum instances.

A bean may flush its pool by casting the SessionContext to Flushable and calling flush(). See SweepInterval for details on how flush is performed.

**import** jakarta.annotation.Resource;

**import** jakarta.ejb.SessionContext;

**import** jakarta.ejb.Stateless;

**import** java.io.Flushable;

**import** java.io.IOException;

**public** **class** MyBean {

**private** SessionContext sessionContext;

**public** **void** flush() **throws** IOException {

((Flushable) sessionContext).flush();

}

}

#### **MaxAgeOffset**

Applies to MaxAge usage and would rarely be changed, but is a nice feature to understand.

When the container first starts and the pool is filled to the minimum size, all those "minimum" instances will have the same creation time and therefore all expire at the same time dictated by the MaxAge setting. To protect against this sudden drop scenario and provide a more gradual expiration from the start the container will spread out the age of the instances that fill the pool to the minimum using an offset.

The MaxAgeOffset is not the final value of the offset, but rather it is used in creating the offset and allows the spreading to push the initial ages into the future or into the past. The pool is filled at startup as follows:

**for** (**int** i = 0; i < poolMin; i++) {

**long** ageOffset = (maxAge / poolMin \* i \* maxAgeOffset) % maxAge;

pool.add(**new** Bean(), ageOffset));

}

The default MaxAgeOffset is -1 which causes the initial instances in the pool to live a bit longer before expiring. As a concrete example, let’s say the MinSize is 4 and the MaxAge is 100 years. The generated offsets for the four instances created at startup would be 0, -25, -50, -75. So the first instance would be "born" at age 0, die at 100, living 100 years. The second instance would be born at -25, die at 100, living a total of 125 years. The third would live 150 years. The fourth 175 years.

A MaxAgeOffset of 1 would cause instances to be "born" older and therefore die sooner. Using the same example MinSize of 4 and MaxAge of 100 years, the life spans of these initial four instances would be 100, 75, 50, and 25 years respectively.

A MaxAgeOffset of 0 will cause no "spreading" of the age of the first instances used to fill the pool to the minimum and these instances will of course reach their MaxAge at the same time. It is possible to set to decimal values such as -0.5, 0.5, -1.2, or 1.2.

#### **IdleTimeout**

Specifies the maximum time that an instance should be allowed to sit idly in the pool without use before it should be retired and removed.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

#### **GarbageCollection**

Allows Garbage Collection to be used as a mechanism for shrinking the pool. When set to true all instances in the pool, excluding the minimum, are eligible for garbage collection by the virtual machine as per the rules of java.lang.ref.SoftReference and can be claimed by the JVM to free memory. Instances garbage collected will have their @PreDestroy methods called during finalization.

In the OpenJDK VM the -XX:SoftRefLRUPolicyMSPerMB flag can adjust how aggressively SoftReferences are collected. The default OpenJDK setting is 1000, resulting in inactive pooled instances living one second of lifetime per free megabyte in the heap, which is very aggressive. The setting should be increased to get the most out of the GarbageCollection feature of the pool. Much higher settings are safe. Even a setting as high as 3600000 (1 hour per free MB in the heap) does not affect the ability for the VM to garbage collect SoftReferences in the event that memory is needed to avoid an OutOfMemoryException.

#### **SweepInterval**

The frequency in which the container will sweep the pool and evict expired instances. Eviction is how the IdleTimeout, MaxAge, and pool "flush" functionality is enforced. Higher intervals are better.

Instances in use are excluded from sweeping. Should an instance expire while in use it will be evicted immediately upon return to the pool. Effectively MaxAge and flushes will be enforced as a part of normal activity or sweeping, while IdleTimeout is only enforcable via sweeping. This makes aggressive sweeping less important for a pool under moderate load.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as 1 hour and 27 minutes and 10 seconds

#### **CallbackThreads**

When sweeping the pool for expired instances a thread pool is used to process calling @PreDestroy on expired instances as well as creating new instances as might be required to fill the pool to the minimum after a Flush or MaxAge expiration. The CallbackThreads setting dictates the size of the thread pool and is shared by all beans deployed in the container.

#### **CloseTimeout**

PostConstruct methods are invoked on all instances in the pool when the bean is undeployed and its pool is closed. The CloseTimeout specifies the maximum time to wait for the pool to close and PostConstruct methods to be invoked.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as 1 hour and 27 minutes and 10 seconds

#### **UseOneSchedulerThreadByBean**

back to previous behavior (TomEE 1.x) where 1 scheduler thread was used for stateless eviction by bean (ie for 500 stateless beans you get 500 eviction threads)

#### **EvictionThreads**

number of threads to associate to eviction threads (1 is not bad for most applications)

## @Stateful

A @Stateful container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"STATEFUL"**>

AccessTimeout = 30 seconds

Cache = org.apache.openejb.core.stateful.SimpleCache

Passivator = org.apache.openejb.core.stateful.SimplePassivater

TimeOut = 20

Frequency = 60

Capacity = 1000

BulkPassivate = 100

</**Container**>

Declarable in properties via

Foo = new://Container?type=STATEFUL

Foo.AccessTimeout = 30 seconds

Foo.Cache = org.apache.openejb.core.stateful.SimpleCache

Foo.Passivator = org.apache.openejb.core.stateful.SimplePassivater

Foo.TimeOut = 20

Foo.Frequency = 60

Foo.Capacity = 1000

Foo.BulkPassivate = 100

### **Configuration**

#### **AccessTimeout**

Specifies the maximum time an invocation could wait for the @Stateful bean instance to become available before giving up.

After the timeout is reached a jakarta.ejb.ConcurrentAccessTimeoutException will be thrown.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as "1 hour and 27 minutes and 10 seconds"

Any usage of the jakarta.ejb.AccessTimeout annotation will override this setting for the bean or method where the annotation is used.

#### **Cache**

The cache is responsible for managing stateful bean instances. The cache can page instances to disk as memory is filled and can destroy abandoned instances. A different cache implementation can be used by setting this property to the fully qualified class name of the Cache implementation.

#### **Passivator**

The passivator is responsible for writing beans to disk at passivation time. Different passivators can be used by setting this property to the fully qualified class name of the PassivationStrategy implementation. The passivator is not responsible for invoking any callbacks or other processing, its only responsibly is to write the bean state to disk.

Known implementations:

* org.apache.openejb.core.stateful.RAFPassivater
* org.apache.openejb.core.stateful.SimplePassivater

#### **TimeOut**

Specifies the time a bean can be idle before it is removed by the container.

This value is measured in minutes. A value of 5 would result in a time-out of 5 minutes between invocations. A value of -1 would mean no timeout. A value of 0 would mean a bean can be immediately removed by the container.

Any usage of the jakarta.ejb.StatefulTimeout annotation will override this setting for the bean where the annotation is used.

#### **Frequency**

Specifies the frequency (in seconds) at which the bean cache is checked for idle beans.

#### **Capacity**

Specifies the size of the bean pools for this stateful SessionBean container.

#### **BulkPassivate**

Property name that specifies the number of instances to passivate at one time when doing bulk passivation.

## @Singleton

A @Singleton container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"SINGLETON"**>

AccessTimeout = 30 seconds

</**Container**>

Declarable in properties via

Foo = new://Container?type=SINGLETON

Foo.AccessTimeout = 30 seconds

### **Configuration**

#### **AccessTimeout**

Specifies the maximum time an invocation could wait for the @Singleton bean instance to become available before giving up.

After the timeout is reached a jakarta.ejb.ConcurrentAccessTimeoutException will be thrown.

Usable time units: nanoseconds, microsecons, milliseconds, seconds, minutes, hours, days. Or any combination such as 1 hour and 27 minutes and 10 seconds

Any usage of the jakarta.ejb.AccessTimeout annotation will override this setting for the bean or method where the annotation is used.

## @MessageDriven

A MDB container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"MESSAGE"**>

ResourceAdapter = Default JMS Resource Adapter

MessageListenerInterface = jakarta.jms.MessageListener

ActivationSpecClass = org.apache.activemq.ra.ActiveMQActivationSpec

InstanceLimit = 10

FailOnUnknowActivationSpec = true

</**Container**>

Declarable in properties via

Foo = new://Container?type=MESSAGE

Foo.ResourceAdapter = Default JMS Resource Adapter

Foo.MessageListenerInterface = jakarta.jms.MessageListener

Foo.ActivationSpecClass = org.apache.activemq.ra.ActiveMQActivationSpec

Foo.InstanceLimit = 10

Foo.FailOnUnknowActivationSpec = true

### **Configuration**

#### **ResourceAdapter**

The resource adapter delivers messages to the container

#### **MessageListenerInterface**

Specifies the message listener interface handled by this container

#### **ActivationSpecClass**

Specifies the activation spec class

#### **InstanceLimit**

Specifies the maximum number of bean instances that are allowed to exist for each MDB deployment.

#### **FailOnUnknowActivationSpec**

Log a warning if true or throw an exception if false is an activation spec can’t be respected

## @Managed

A managed bean container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"MANAGED"** />

Declarable in properties via

Foo = new://Container?type=MANAGED

## CMP entity

A CMP bean container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"CMP\_ENTITY"**>

CmpEngineFactory = org.apache.openejb.core.cmp.jpa.JpaCmpEngineFactory

</**Container**>

Declarable in properties via

Foo = new://Container?type=CMP\_ENTITY

Foo.CmpEngineFactory = org.apache.openejb.core.cmp.jpa.JpaCmpEngineFactory

### **Configuration**

#### **CmpEngineFactory**

The engine to use for this container. By default TomEE only provides the JPA implementation.

## BMP entity

A BMP entity container.

Declarable in tomee.xml via

<**Container** id=**"Foo"** type=**"BMP\_ENTITY"**>

PoolSize = 10

</**Container**>

Declarable in properties via

Foo = new://Container?type=BMP\_ENTITY

Foo.PoolSize = 10

### **Configuration**

#### **PoolSize**

Specifies the size of the bean pools for this bmp entity container.

# Log4j2 Configuration with TomEE

Out of the box, TomEE is uses a Java-Util-Logging (JUL) based logging system, which is configured using conf/logging.properties.

Occasionally, users may wish to swap over to using Log4j2. These instructions detail how to do this with the latest TomEE versions. These instructions have been tested with TomEE 7.x and TomEE 8.x (master).

## Setup

You’ll need to obtain the following jars: log4j-core, log4j-api and log4j-jul. These instructions were tested with the 2.17.1 versions:

<https://repo1.maven.org/maven2/org/apache/logging/log4j/log4j-core/2.17.1/log4j-core-2.17.1.jar> <https://repo1.maven.org/maven2/org/apache/logging/log4j/log4j-api/2.17.1/log4j-api-2.17.1.jar> <https://repo1.maven.org/maven2/org/apache/logging/log4j/log4j-jul/2.17.1/log4j-jul-2.17.1.jar>

Add these to the TomEE bin directory. Add the following to setenv.sh on \*nix:

JAVA\_OPTS=**"$JAVA\_OPTS -Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager"**

LOGGING\_CONFIG=**"-DnoOp"**

LOGGING\_MANAGER=**"-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager"**

CLASSPATH=**".:$CATALINA\_BASE/bin:$CATALINA\_BASE/bin/log4j-core-2.17.1.jar:$CATALINA\_BASE/bin/log4j-api-2.17.1.jar:$CATALINA\_BASE/bin/log4j-jul-2.17.1.jar"**

or add the following to setenv.bat on Windows:

@echo off

set **"JAVA\_OPTS=%JAVA\_OPTS% -Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager**

**set LOGGING\_CONFIG=-DnoOpp**

**set LOGGING\_MANAGER=-Djava.util.logging.manager=org.apache.logging.log4j.jul.LogManager**

**set "**CLASSPATH=.;%CATALINA\_BASE%\bin;%CATALINA\_BASE%\bin\log4j-core-2.17.1.jar;%CATALINA\_BASE%\bin\log4j-api-2.17.1.jar;%CATALINA\_BASE%\bin\log4j-jul-2.17.1.jar**"**

If you are using the TomEE Maven Plugin, it can be configured as follows:

<**plugin**>

<**groupId**>org.apache.tomee.maven</**groupId**>

<**artifactId**>tomee-maven-plugin</**artifactId**>

<**version**>${tomee.plugin.version}</**version**>

<**configuration**>

<**tomeeVersion**>${tomee.version}</**tomeeVersion**>

<**tomeeClassifier**>plus</**tomeeClassifier**>

<**classpaths**>

<**classpath**>org.apache.logging.log4j:log4j-api:2.17.1</**classpath**>

<**classpath**>org.apache.logging.log4j:log4j-jul:2.17.1</**classpath**>

<**classpath**>org.apache.logging.log4j:log4j-core:2.17.1</**classpath**>

</**classpaths**>

<**systemVariables**>

<**log4j.configurationFile**>${project.basedir}/src/test/tomee/conf/log4j2.xml</**log4j.configurationFile**>

<**java.util.logging.manager**>org.apache.logging.log4j.jul.LogManager</**java.util.logging.manager**>

</**systemVariables**>

</**configuration**>

</**plugin**>

Take care to match the jar filenames if you have downloaded jars for a slightly different version of log4j2.

## Configuration

Add your log4j2.xml config in the bin directory. Here’s a simple config you can use to help you get started:

<?xml version=**"1.0"** encoding=**"UTF-8"** ?>

<**Configuration** status=**"warn"** name=**"catalina"** packages=**""**>

<**Appenders**>

<**Console** name=**"console"** target=**"SYSTEM\_OUT"**>

<**PatternLayout** pattern=**"%d %p %c{1.} [%t] %m%n"** />

</**Console**>

<**File** name=**"catalina"** fileName=**"${sys:catalina.base}/logs/catalina.log"**>

<**PatternLayout**>

<**Pattern**>%d %p %c{1.} [%t] %m%n</**Pattern**>

</**PatternLayout**>

</**File**>

<**Async** name=**"Async"**>

<**AppenderRef** ref=**"catalina"** />

</**Async**>

</**Appenders**>

<**Loggers**>

<**Root** level=**"info"**>

<**AppenderRef** ref=**"Async"** />

<**AppenderRef** ref=**"console"** />

</**Root**>

</**Loggers**>

</**Configuration**>

## Application

Some settings can be specific to applications, these ones are also properties based and are read in WEB-INF/application.properties. When you can’t use tomee.xml to configure resources you can use WEB-INF/resources.xml which inherit from tomee.xml its syntax but binds the resources to the application and reuses the application classloader.

More about [Container Configuration](https://tomee.apache.org/latest/docs/admin/configuration/application.html).