Great find—yes, the artifact [wmq.jakarta.jmsra](https://wmq.jakarta.jmsra) version **9.3.5.1** is the Jakarta Messaging 3.0-compliant resource adapter from IBM MQ. However, its compatibility with **JBoss EAP 8.x** depends on whether your JBoss runtime supports **Jakarta EE 9+**, which uses the jakarta.\* namespace instead of javax.\*.

**✅ When to Use** [**wmq.jakarta.jmsra**](https://wmq.jakarta.jmsra)

Use this RAR if:

* Your JBoss 8.x instance is configured for **Jakarta EE 9 or higher**
* Your application uses [**jakarta.jms**](https://jakarta.jms)**.\*** packages instead of [javax.jms](https://javax.jms).\*

**⚠️ Compatibility Caveats**

* JBoss EAP 8.x is designed to support Jakarta EE 10, so it **should** be compatible with Jakarta Messaging 3.0.
* You **cannot mix** [javax.jms](https://javax.jms) and [jakarta.jms](https://jakarta.jms) in the same application.
* If your legacy application still uses [javax.jms](https://javax.jms), you should stick with the older [wmq.jmsra](https://wmq.jmsra) RAR (e.g., version 9.2.2.0).

**🔍 Reference**

You can explore the artifact on [Maven Central](https://central.sonatype.com/artifact/com.ibm.mq/wmq.jakarta.jmsra) and read IBM’s guidance on [Jakarta Messaging support](https://www.ibm.com/docs/en/ibm-mq/9.3.x?topic=applications-using-mq-classes-jmsjakarta-messaging).

Would you like help inspecting your application’s JMS imports to confirm which RAR is appropriate?

Perfect! To confirm which IBM MQ resource adapter (RAR) is appropriate for your application running on JBoss EAP 8.x, we’ll want to inspect your JMS imports. Here’s a quick way to do that:

**🧪 Step-by-Step: Identify JMS Namespace**

1. **Search Your Codebase** Look for JMS-related imports in your application. You can use grep or a code editor’s search function:
2. grep -r "import javax.jms" /path/to/your/codebase

grep -r "import jakarta.jms" /path/to/your/codebase

1. **Interpret the Results**
   * If you see import [javax.jms](https://javax.jms).\*, your app is using the **older JMS API**, and you should stick with [wmq.jmsra](https://wmq.jmsra) (e.g., version 9.2.2.0).
   * If you see import [jakarta.jms](https://jakarta.jms).\*, your app is using **Jakarta Messaging 3.0**, and [wmq.jakarta.jmsra](https://wmq.jakarta.jmsra) (version 9.3.5.1) is the right choice.
2. **Double-Check Dependencies** You can also inspect your [pom.xml](https://pom.xml) or [build.gradle](https://build.gradle) for JMS dependencies:
   * Look for artifacts like jakarta.jms:jakarta.jms-api or javax.jms:jms.

Would you like help writing a script to automate this check across multiple modules or repositories?

Ah, that’s a crucial observation—and it aligns with how JBoss EAP 8.x is architected. Since JBoss 8.x is built for **Jakarta EE 10**, it expects all EE APIs, including JMS, to use the jakarta.\* namespace. Here’s what’s likely happening under the hood:

**🧩 Why** [**javax.jms**](https://javax.jms) **RAR Fails on JBoss EAP 8.x**

* **Namespace Mismatch**: JBoss EAP 8.x modules internally route to Jakarta EE APIs ([jakarta.jms](https://jakarta.jms).\*). When you deploy a RAR that uses [javax.jms](https://javax.jms).\*, the classloader can’t resolve those types correctly, leading to deployment failure.
* **Module Resolution Conflict**: Even if you try to manually add [javax.jms](https://javax.jms) modules, JBoss’s internal wiring favors [jakarta.jms](https://jakarta.jms), causing linkage errors or ClassNotFoundException.
* **No Backward Compatibility Layer**: Unlike some app servers that offer compatibility bridges, JBoss EAP 8.x does **not** support automatic mapping between javax.\* and jakarta.\*.

**✅ Recommended Path Forward**

Since your JBoss 8.x instance is locked into Jakarta EE 10:

* Use [**wmq.jakarta.jmsra**](https://wmq.jakarta.jmsra) **version 9.3.5.1** from IBM MQ.
* Ensure your application code and dependencies are fully migrated to [**jakarta.jms**](https://jakarta.jms)**.\***.
* If you’re using legacy code, consider refactoring the JMS portions or using bytecode transformation tools (though that’s rarely ideal).

**📝 Migration Checklist**

1. **Audit Your Codebase**
   * Search for all JMS-related imports using grep or your IDE.
   * Identify any usage of [javax.jms](https://javax.jms) and plan for migration to [jakarta.jms](https://jakarta.jms).
2. **Update Dependencies**
   * Replace javax.jms:jms with jakarta.jms:jakarta.jms-api in your [pom.xml](https://pom.xml) or [build.gradle](https://build.gradle).
3. **Refactor Code**
   * Update all [javax.jms](https://javax.jms) imports to [jakarta.jms](https://jakarta.jms).
   * Test each refactored module to ensure compatibility.
4. **Validate RAR Deployment**
   * Deploy the [wmq.jakarta.jmsra](https://wmq.jakarta.jmsra) RAR independently on JBoss EAP 8.x.
   * Check server logs for any deployment errors.
5. **Test Application Integration**
   * Deploy your application alongside the RAR.
   * Run integration tests to verify JMS functionality.
6. **Monitor Performance**
   * Use JBoss monitoring tools to ensure the RAR and application are performing as expected.

**🛠️ Validation Script**

Here’s a simple script to automate the namespace check:

#!/bin/bash

# Path to your codebase

CODEBASE\_PATH="/path/to/your/codebase"

# Search for javax.jms imports

echo "Checking for javax.jms imports..."

grep -r "import javax.jms" $CODEBASE\_PATH

# Search for jakarta.jms imports

echo "Checking for jakarta.jms imports..."

grep -r "import jakarta.jms" $CODEBASE\_PATH

# Output results

echo "Namespace check complete."

Would you like help customizing this checklist or script for your specific environment?

**🛠️ Migration Resource Adapter Configuration**

To migrate the resource adapter configuration from [standalone-ha.xml](https://standalone-ha.xml) in JBoss 7.3 to JBoss 8.x, follow these steps:

1. **Backup Configuration Files**
   * Create backups of both [standalone-ha.xml](https://standalone-ha.xml) files from JBoss 7.3 and JBoss 8.x.
2. **Compare Configuration Structure**
   * Use a diff tool to identify differences between the two files.
   * Pay attention to changes in namespace, module definitions, and subsystem configurations.
3. **Update Resource Adapter Definitions**
   * Locate the <resource-adapter> section in the JBoss 7.3 [standalone-ha.xml](https://standalone-ha.xml).
   * Update the namespace to jakarta.\* if required.
   * Ensure compatibility with Jakarta EE 10 APIs.
4. **Validate Module Dependencies**
   * Check for any module dependencies referenced in the resource adapter configuration.
   * Update module paths and names to match JBoss 8.x conventions.
5. **Test Deployment**
   * Deploy the updated [standalone-ha.xml](https://standalone-ha.xml) on JBoss 8.x.
   * Monitor server logs for errors or warnings related to the resource adapter.
6. **Iterative Testing**
   * Test the application functionality with the migrated resource adapter.
   * Resolve any issues related to classloading or subsystem integration.

Would you like help automating the migration process or validating the updated configuration?

**🛠️ Configuration Validation Script**

Here’s a script to validate the resource adapter configuration:

#!/bin/bash

# Path to configuration files

JBOSS7\_CONFIG="/path/to/jboss7/standalone-ha.xml"

JBOSS8\_CONFIG="/path/to/jboss8/standalone-ha.xml"

# Backup original files

echo "Creating backups..."

cp $JBOSS7\_CONFIG $JBOSS7\_CONFIG.bak

cp $JBOSS8\_CONFIG $JBOSS8\_CONFIG.bak

# Compare configurations

echo "Comparing configurations..."

diff $JBOSS7\_CONFIG $JBOSS8\_CONFIG

# Validate resource adapter section

echo "Validating resource adapter configuration..."

grep -E "<resource-adapter>|<module-name>|<connection-definition>" $JBOSS8\_CONFIG

# Output results

echo "Validation complete."

Would you like help customizing this script for your environment?

Would you like help customizing this checklist or script for your specific environment?