Application isolation for Java in IBM Integration Bus V10 (and V9.0.0.3!)

Simon Stone

Published on June 23, 2015 / Updated on March 22, 2016

14

Introduction

Previous releases of IBM Integration Bus, starting with IBM WebSphere Message Broker V8, do not provide Java class loader isolation for Java classes deployed within applications. This can cause unexpected behaviour during deployment or runtime when Java classes are deployed within multiple applications to a single integration server.

IBM Integration Bus V10 introduces application isolation for Java, and enables this support by default for all new applications. This support has also been back ported and released as part of fix pack 3 for V9 (V9.0.0.3), but in V9 the support is disabled by default to avoid changing behaviour as part of a fix pack upgrade.

It is possible that this new functionality may effect existing integration solutions which have been built to rely on the lack of Java class loader isolation. Users with existing integration solutions that make use of Java classes should review this article for details on these changes and potential migration issues before migrating those solutions to V10.

Java in integration solutions

IBM Integration Bus allows you to build transformation logic in Java code. Most Java code is used by JavaCompute nodes in a message flow, however Java code can also be invoked from ESQL code and from a graphical map (by using custom Java transformations).

Java code can either be provided in the form of source code that is developed using the toolkit, or as precompiled Java code that is packaged into JAR files. Additionally, JAR files may be provided to provide additional functionality – we see many customers using third party Java libraries that provide additional functionality. An example of such a Java library is log4j, a popular Java logging framework.

Java classes and JAR files can be placed into several types of container:

Applications, integration services, and REST APIs

Applications are a container for all of the artefacts that make up an integration solution. Integration services and REST APIs are two specialised types of application, which provide a container for SOAP based or REST based services respectively. Applications were also designed to provide isolation from other integration solutions – artefacts within applications should not be visible outside of the scope of that application.

Static libraries

Libraries were introduced in V8, and can be used to store reusable artefacts that can be used across multiple integration solutions. In V10, these libraries have been renamed to static libraries. Applications can include static libraries, and make use of those reusable artefacts. When such an application is packaged into a BAR file, the application is packaged up with a private copy of those static libraries and the resources within that static library. This means that multiple copies of the same static library will be deployed when multiple applications include it. Because of this, it is easy to get into a situation where different versions of the same static library are deployed within multiple applications.

Static libraries can also be referenced by artefacts in integration projects. In this case, a single copy of the static library is deployed directly to the integration server.

Shared libraries

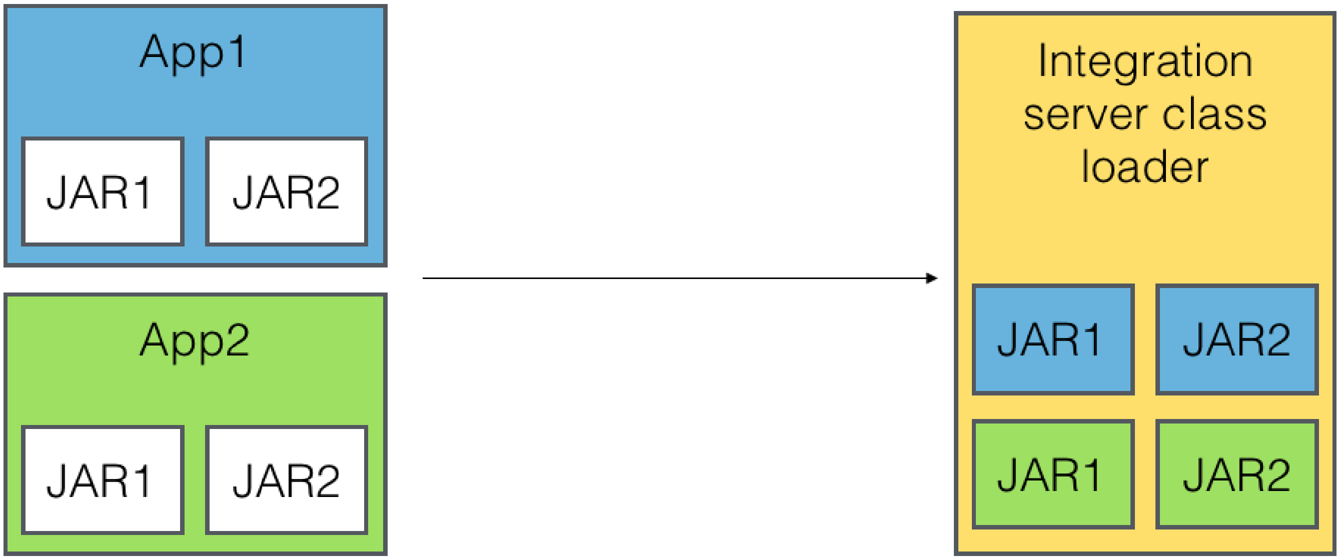
Shared libraries were introduced in V10. Shared libraries also provide a container that can be used to store reusable artefacts that can be used across multiple integration solutions. Applications can reference shared libraries, and make use of these reusable artefacts. Shared libraries must be deployed directly to the integration server. Unlike static libraries, the shared library is not copied into the application when it is packaged into a BAR file – instead, all applications referencing that shared library refer to the same deployed copy of that shared library.

Integration projects

Integration projects contain artefacts that are deployed directly to the integration server. All integration solutions that were developed before V7, or without using applications, will be stored in integration projects.

Previous class loading behaviour

Java classes and JAR files are usually packaged into a BAR file, which is then deployed to the integration server. When an integration server receives a deployment containing Java classes and JAR files, it loads all of those Java classes and JAR files into a single integration server wide class loader. That single class loader is used to access all deployed Java classes, regardless of whether or not those Java classes has been deployed as part of an application:



This has some obvious pitfalls. If App1 and App2 contain two copies of a given Java class, then both Java classes will be loaded into that single class loader. The order that those classes is loaded into that single class loader is non-deterministic, and is affected by multiple factors, including project names and time of deployment. The order may even be modified by just restarting the integration server.

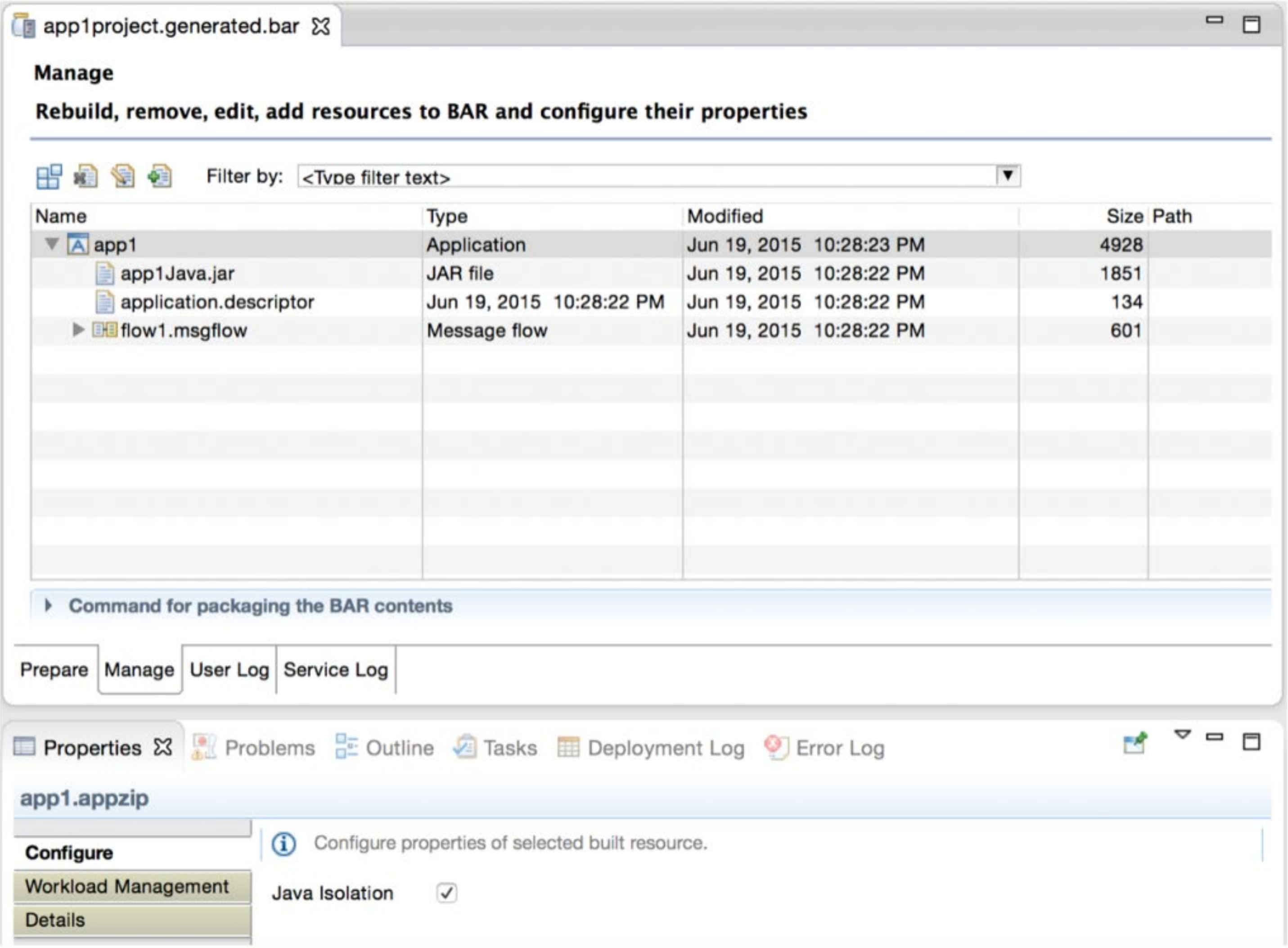
If version 1 of a Java class is deployed inside App1, and version 2 of the same Java class is deployed inside App2, then both App1 and App2 will end up using the same version of that Java class. This may cause incorrect behaviour during runtime of one of the applications when it makes a call to a version of the Java class that it was not packaged with.

Additionally, any static variables (such as singletons) are visible at the class loader level. Because there is only a single class loader, all static variables are visible to all applications. This may cause undesired behaviour in those applications – for example, one application could clear a static map that is being used to store important state by another application.

In IBM Integration Bus V10, we provide true Java isolation for Java classes deployed within an application. These changes have also been back ported and released as part of fix pack 3 for V9 (V9.0.0.3).

New Java isolation option for applications

Applications have now been extended to have a new boolean attribute, called javaIsolation. The new attribute is visible from the BAR editor in the Integration Toolkit, once an application has been packaged into a BAR file:



Manage

Rebuild, remove, edit, add resources to BAR and configure their properties

Filter by: <Type filter text>

Name	Type	Modified	Size	Path
app1	Application	Jun 19, 2015 10:28:23 PM	4928	
app1Java.jar	JAR file	Jun 19, 2015 10:28:22 PM	1851	
application.descriptor	Jun 19, 2015 10:28:22 PM	Jun 19, 2015 10:28:22 PM	134	
flow1.msgflow	Message flow	Jun 19, 2015 10:28:22 PM	601	

Command for packaging the BAR contents

app1.appzip

Configure

Workload Management

Details

Configure properties of selected built resource.

Java Isolation ☑

By default, any applications packaged with V10 will have this attribute set to true. This applies to applications packaged with the Integration Toolkit. It also applies to applications packaged with the command line applications mqsicreatebar and mqsipackagebar. When you move to V10 or later, if you require the old behaviour, you must manually set the attribute to false after the application is packaged into a BAR file.

Applications packaged with V9.0.0.3 will have this attribute set to false by default. This decision was made so that the behaviour of existing applications would not be modified just by installing fix pack 3. Users wishing to exploit Java isolation in V9 must install fix pack 3 or later, and explicitly opt-in to Java isolation by manually setting the attribute to true.

Applications packaged before V9.0.0.3 – BAR files built using V9 to V9.0.0.2, or V8 – will not have a javaIsolation attribute present. When the attribute is not present, the runtime defaults to the old behaviour, as if the javaIsolation attribute was present and set to false. This way, any existing BAR files will continue to work in exactly the same way as they did before.

The current value of the javaIsolation attribute for an application inside a BAR file can be examined with the BAR editor in the Integration Toolkit, as well as the mqsireadbar command:

Displaying the current Java isolation setting for an application packaged in a BAR file using the command line

```
$ mqsireadbar -b app1project.generated.bar -r
BIP1052I: Reading Bar file using runtime mqsireadbar...
app1project.generated.bar:
  app1.appzip (19/06/15 16:57):
    application.descriptor (19/06/15 16:57):
    app1Java.jar (19/06/15 16:57):
    Deployment descriptor:
      startMode
      javaIsolation
```

Because this BAR file was packaged with V10, the javaIsolation flag is set to the default value of true. Because it is the default value, and it has not been overridden, the string true does not appear in the output of mqsireadbar.

In addition to using the BAR editor in the Integration Toolkit, the value of the javaIsolation attribute can be modified using the mqsiapplybaroverrides command:

Enabling Java isolation for an application packaged in a BAR file using the command line

```
$ mqsiapplybaroverride -b app1project.generated.bar -m javaIsolation=true -k app1
BIP1138I: Applying overrides using runtime mqsiapplybaroverride...
BIP1140I: Overriding property javaIsolation with 'true' in 'app1.appzip/META-INF/broker.xml' ...
BIP1143I: Saving Bar file app1project.generated.bar...

BIP8071I: Successful command completion.

$ mqsireadbar -b app1project.generated.bar -r
BIP1052I: Reading Bar file using runtime mqsireadbar...
app1project.generated.bar:
  app1.appzip (19/06/15 20:48):
    application.descriptor (19/06/15 20:48):
    app1Java.jar (19/06/15 20:48):
    Deployment descriptor:
      startMode
      javaIsolation = true
```

Disabling Java isolation for an application packaged in a BAR file using the command line

```
$ mqsiapplybaroverride -b app1project.generated.bar -m javaIsolation=false -k app1
BIP1138I: Applying overrides using runtime mqsiapplybaroverride...
BIP1140I: Overriding property javaIsolation with 'false' in 'app1.appzip/META-INF/broker.xml' ...
BIP1143I: Saving Bar file app1project.generated.bar...

BIP8071I: Successful command completion.

$ mqsireadbar -b app1project.generated.bar -r
BIP1052I: Reading Bar file using runtime mqsireadbar...
app1project.generated.bar:
  app1.appzip (19/06/15 20:48):
    application.descriptor (19/06/15 20:48):
    app1Java.jar (19/06/15 20:48):
    Deployment descriptor:
      startMode
      javaIsolation = false
```

Once the BAR file has been deployed, the value of the javaIsolation attribute for the deployed application can be examined using the command line and the administrative web user interface:

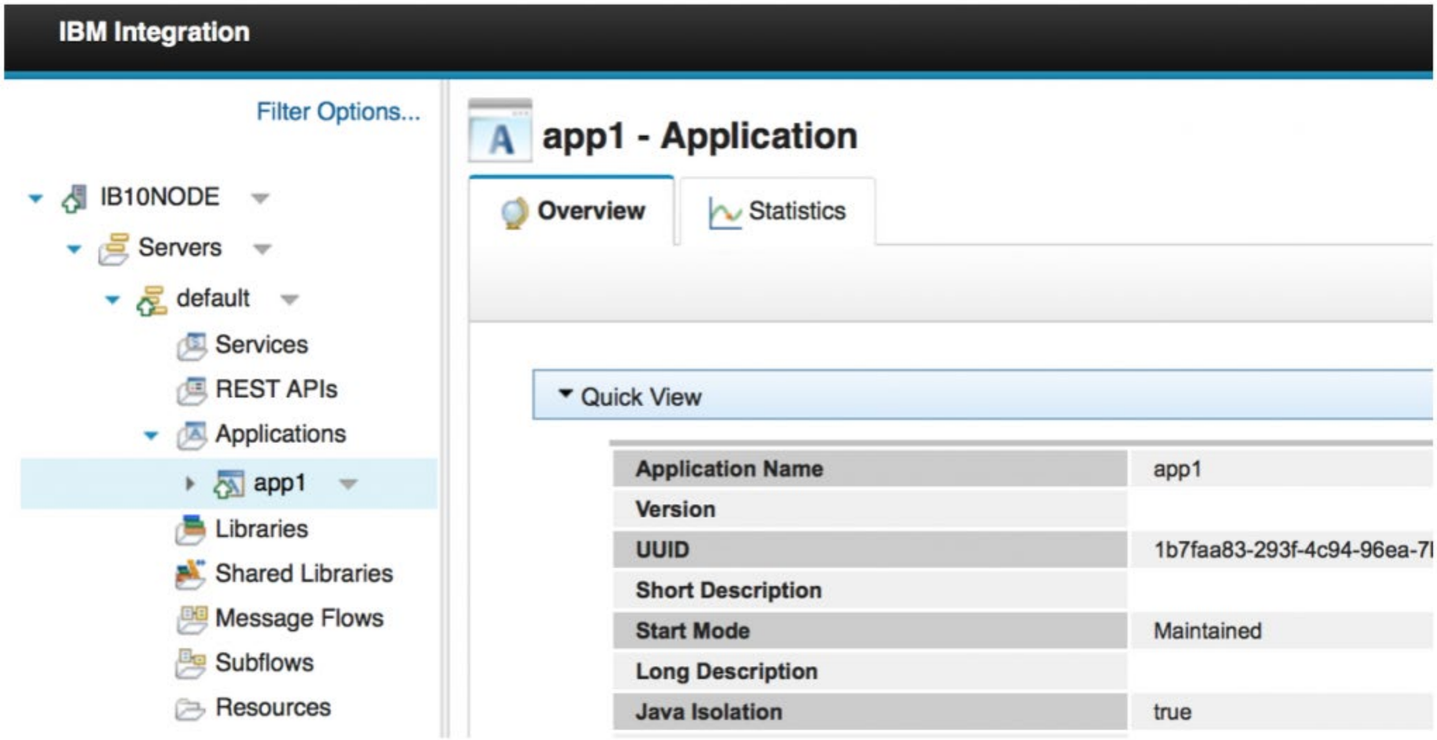
Displaying the current Java isolation setting for an application deployed to an integration server using the command line

```
$ mqsilist IB10NODE -e default -d2

--------
BIP1275I: Application 'app1' on integration server 'default' is running.

Deployed: '19/06/15 16:57' in Bar file '/Users/sstone1/runtime-EclipseApplication/GeneratedBarFiles/app1pr
oject.generated.bar'
Last edited: '19/06/15 16:57'
UUID '1b7faa83-293f-4c94-96ea-7bbf05922705'
Start mode: 'Maintained'
Java isolation: 'true'
Long description: ''
Keywords:
  'VERSION' = '
BIP8071I: Successful command completion.
```

Displaying the current Java isolation setting for an application deployed to an integration server using the administrative web user interface

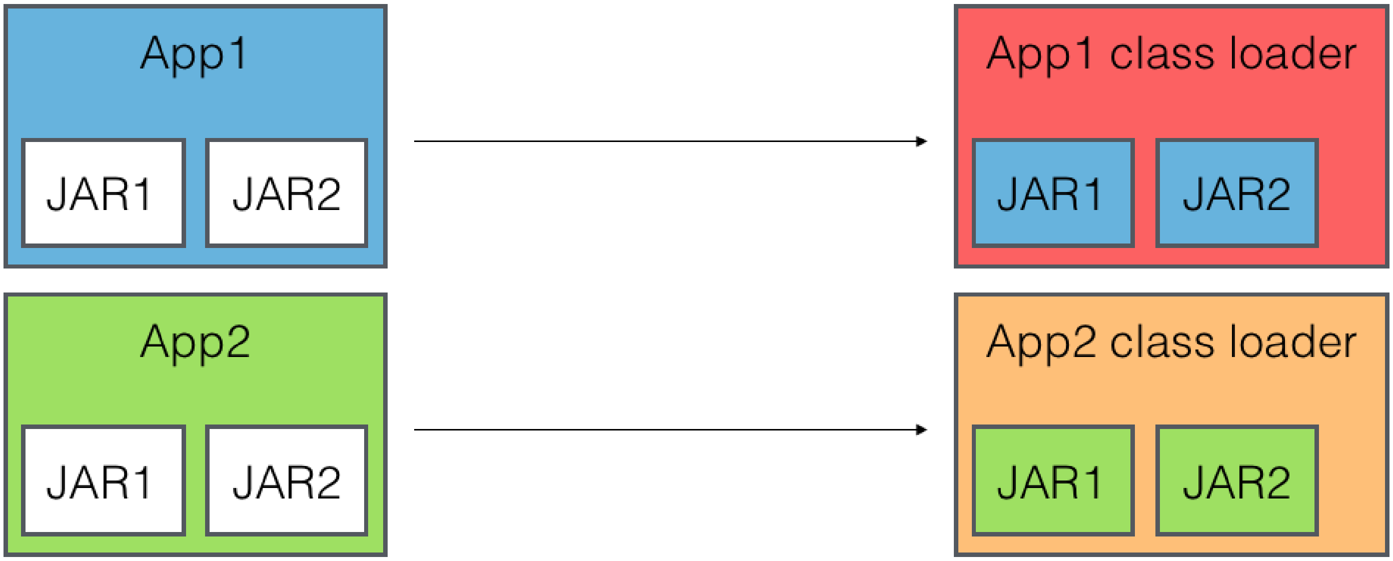


Application Name	app1
Version	
UUID	1b7faa83-293f-4c94-96ea-7
Short Description	
Start Mode	Maintained
Long Description	
Java Isolation	true

New class loading behaviour

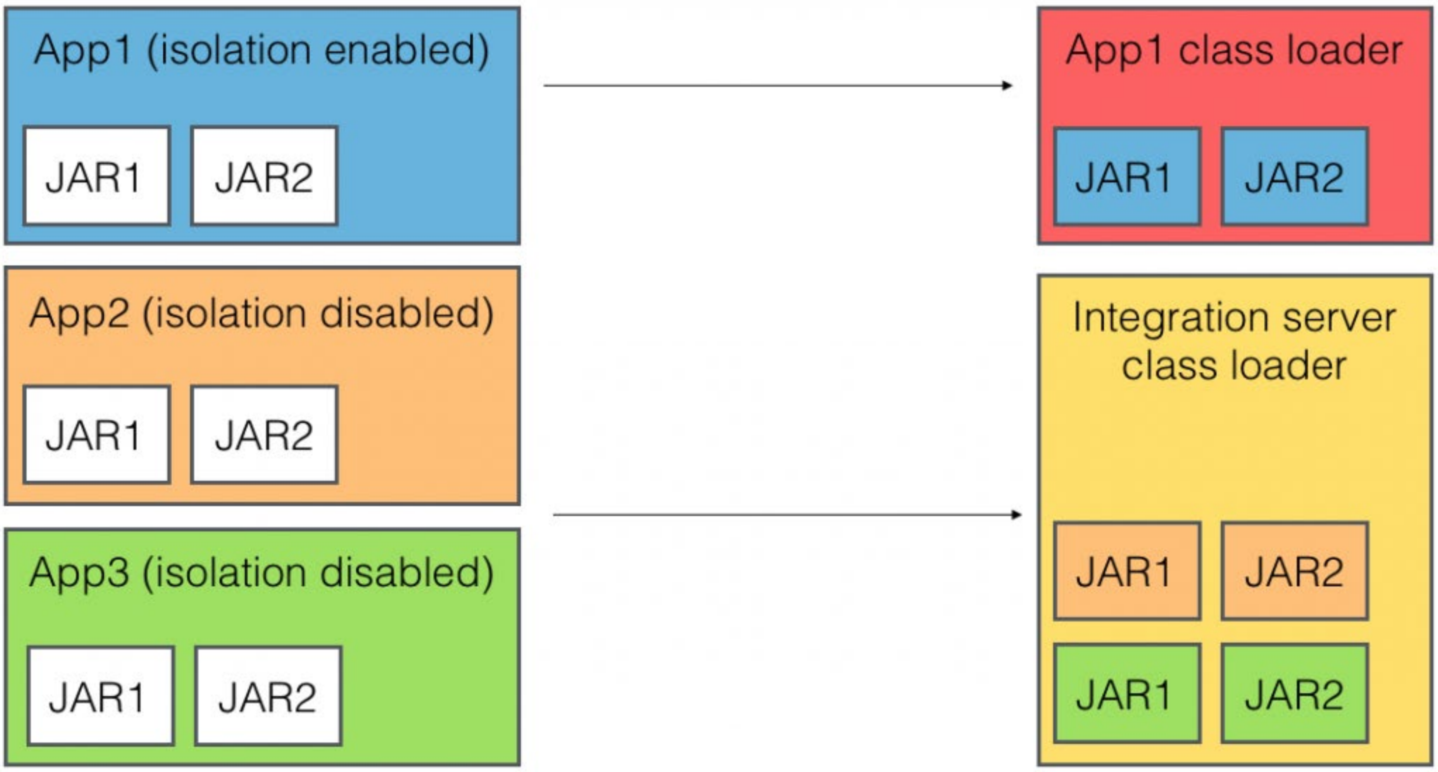
Applications, integration services, and REST APIs

Now when an integration server receives a deployment containing Java classes and JAR files inside of an application, it examines the value of the javaIsolation flag. If it is present, and left unset or explicitly set to true, then Java isolation is enabled for that application. If Java isolation is enabled for an application, then those Java classes and JAR files are loaded into a class loader exclusively for that application. For example, if both App1 and App2 are deployed with Java isolation enabled:



Since both applications have Java isolation enabled, Java code in App1 cannot see the Java code in App2, and vice versa. If App1 and App2 have different versions of the same Java code, then both applications will always use the correct version of the Java code – the version that they were packaged with.

The class loader for the integration server continues to exist. When Java isolation is enabled for an application, the Java classes from that application are not loaded into the integration server class loader, and are no longer visible to any Java code outside of that application. If Java isolation is disabled for an application, then those Java classes and JAR files continue to be loaded into the class loader for the integration server. It is possible to deploy a mix of applications – some with Java isolation enabled, and some without:



Static libraries

When an application includes a static library, and that application has Java isolation enabled, the Java classes and JAR files within that application and all of its included static libraries are loaded into a class loader for that application. If Java isolation is disabled, then those Java classes and JAR files are loaded into the class loader for the integration server.

Java classes and JAR files within static libraries deployed directly to an integration server, outside of an application, are always loaded into the class loader for the integration server.

Shared libraries

Java classes and JAR files within a shared library are always loaded into a class loader for that shared library. Java classes and JAR files within a shared library are never loaded into the class loader for the integration server.

Integration projects

Java classes and JAR files within integration projects – including all artefacts developed before applications were introduced in V8 – will maintain the original behaviour. Java classes and JAR files will always be loaded into the class loader for the integration server.

Potential migration issues

When an existing application is repackaged using V10, then Java isolation will be automatically enabled for that application. You may find that the behaviour of the existing application is modified, if that application was relying on the lack of Java isolation. In this case, we would recommend that as part of your migration to V10 you refactor your application to work with Java isolation enabled. However, we appreciate that this may not be feasible due to project commitments or timelines, and so we have made it possible to explicitly disable Java isolation for an application once that application has been packaged into a BAR file.

If an existing application is repackaged using V9.0.0.3, then Java isolation will not be automatically enabled for that application. Java isolation must be explicitly enabled for an application once that application has been packaged into a BAR file.

Here's a couple of examples of integration solutions that rely on the lack of Java isolation, which will no longer function correctly if Java isolation is enabled:

Applications use Java code from a static library that is deployed to the integration server

A static library contains some common Java code, such as log4j, that is used by multiple applications. Those applications contain ESQL code that makes calls to static Java methods in the static library. Because Java references in ESQL are not validated by the toolkit, the applications do not have references to the static library. Instead, the static library is deployed directly to the integration server. When the applications are also deployed to the integration server, they are able to see the Java code in the static library, even though it is outside of the scope of that application.

When Java isolation is enabled for those applications, those applications are unable to be deployed or started as the Java code in the static library will no longer be visible. The ESQL code will no longer be able to access the static Java methods.

Applications share data in a Java singleton

An application contains a set of Java code that creates a Java singleton. That Java singleton is used to cache results from database queries into a Java singleton, or store some temporary integration state. That same Java code is placed into multiple applications, possibly through the use of a static library. Only a single copy of that Java code is used by all of the applications that contain that Java code (because it has the same class name), and so the same instance of the Java singleton is shared by all of those applications.

When Java isolation is enabled for those applications, those applications can be deployed without errors. However, each application now has its own copy of the Java singleton. In the database cache case, this may lead to increased Java heap usage if each application now maintains its own cache of database query results. In the state store case, if multiple applications need to access the same stored state, then those applications may not function correctly because they are no longer able to share state.

Solutions

In both scenarios, when using IBM Integration Bus V10, it is recommended to migrate the Java code into a shared library which is then referenced by the applications. Because the Java code is in a shared library, and not a static library, the Java code is not duplicated across all of the referencing applications. The applications can then all reference the same copy of the Java code by using the class loader for that shared library.

Alternatively, the Java code can be removed from the static library, and instead packaged into a JAR file that is placed into the shared-classes directory on the file system of the system running IBM Integration Bus. Java classes within JAR files that are placed into one of the shared-classes directories are visible to all deployed applications, regardless of whether or not those applications have Java isolation enabled. The downside to this approach is that the Java code is not deployed, and requires an integration server restart to pick up changes to the JAR file.

TAGS IIB, JAVA, DEVELOP-INTEGRATION-SOLUTION, INTEGRATION-SOLUTION, APPLICATION

Simon Stone

---

14 comments on"Application isolation for Java in IBM Integration Bus V10 (and V9.0.0.3!)"

Uday Prasanna January 10, 2018

Hi simon,
I want the code for accesing the .broker file details present in different servers?
Is there any api for this?

Reply (Edit)

Partha_wipro February 03, 2017

Hi Simon,
I want to call some java methods from my ESQL code. The ESQL code are containing with in a Shared library.
But I did not find any option in the toolkit to create a shared library containing java files.

Reply (Edit)

FurqanBaqai July 14, 2016

Hi Simon,

Thanks a lot for explaining it. We have noticed this behavior with version 9003 and was looking explanation.

Thanks again.

Reply (Edit)

jino james March 31, 2016

Hi,

Is there any way to reset the static values in shared classes class loader without restarting the Broker?

Reply (Edit)

Simon Stone April 04, 2016

Hi Jino,

As you know, you can reload classes loaded from the shared-classes directory by restarting the execution group (you don't have to restart the whole broker). This will reload the Java classes from the JAR files, resulting in all of the static variables being reset to their default values.

As an alternative to restarting the execution group, you can create an administrative message flow that resets the static variables. For example, you might have an HTTPInput -> JavaCompute -> HTTPReply flow that when called (possibly automatically via scripting) executes some Java code that resets the static variables.

Regards, Simon.

Reply (Edit)

Yogesh Bhardwaj September 20, 2016

Hi Simon,

Appears I have to made some correction in my Queries/Comment dated January 5, 2016 at 7:02 am. Actually it is clear to me now what is visibility and what is delegation.

Would you please confirm me on below notes, Is my understanding around Shared library is correct..?

Note1: There is no delegation between shared library and referenced shared library class loaders. & if you include a JavaCompute node in a subflow in a shared library, that node can access the Java classes in that shared library and any referenced shared libraries.(Only visibility)

Note 2: Static libraries and integration projects cannot reference shared libraries. Java classes in static libraries or independent projects cannot access Java classes in shared libraries.

Note 3: Java classes in applications cannot access Java classes in shared libraries but a JavaCompute node in a flow or subflow in an application, can access the Java classes in a shared library and any shared libraries that are referenced by that shared library. & there is no delegation between application and referenced shared library class loaders.

Note 4: ESQL code in applications or shared libraries can also call static Java methods in referenced shared libraries by using the shared library qualifier in the CLASSLOADER clause.

(Integration server class loader) -> shared class loader -> system class loader -> Bootstrap class loader

(Class loader specified by configurable service) -> shared class loader -> system class loader -> Bootstrap class loader

(Application class loader with Java isolation enabled) -> shared class loader -> system class loader -> Bootstrap class loader

(A shared library class loader) -> shared class loader -> system class loader -> Bootstrap class loader

Regards,
Yogesh Bhardwaj

Reply (Edit)

Yogesh Bhardwaj January 05, 2016

Hi ya,

I have read this blog twice with my queries which were answered by you....Hope this will be the last set of queries i have in this regards..

I well understood the problem mentioned under “Applications share data in a Java singleton”.... and one part of the solution which is “Alternatively, the Java code can be removed from the static library, and instead packaged into a JAR file that is placed into the shared-classes directory on the file system of the system running IBM Integration Bus.” Q1:- will Shared lib class loader will be able to access the class/jars under shared class loader.(for eg log4j.)...? you have not mentioned any delegation in between these two and this is feasible design approach.

Now alternative solution which you specified is “In both scenarios, when using IBM Integration Bus V10, it is recommended to migrate the Java code into a shared library which is then referenced by the applications.”
On same time you mentioned “There is no delegation between application and referenced shared library class loaders,and there is no delegation between shared library and referenced shared library class loaders”..

Which means if a shared library(A)(having jarA) and shared library(B)(having jarB) reference other shared library(C) (having jarC).
“A and B” having its own copy of jarC in memory at run time. I believe this is true As IBM mentioned it “Shared libraries and Java files”

Same fashion, a Application(App1)(having jcn) and Application(App2)(having jcn) reference to same shared library(Lib) (having jar2).
“App1 and App2” having its own copy of jar2 in memory at run time. Is this correct understanding..? As both having different class loader and as you mentioned no delegation. if their is no delegation how come this be true, “The applications can then all reference the same copy of the Java code by using the class loader for that shared library.”..?

Looking forward for your inputs...

Reply (Edit)

Yogesh Bhardwaj December 31, 2015

Let me put my query in another way....please let me know if you can put some light in this area.... I need this information As soon as possible..Hope you understand my concern.

Integration server: 1 JVM and Multiple class loaders..

Different class loaders in IIB:
Class loader for each Application(isolation mode enable)
Class loader for each Shared Library
Class loader for Integration server.
Class loader(shared class loader) for jar placed in “shared-classes”
Class loader specified by configurable service(JCN by default use Integration server class loader, but can use configurable service to make use of another class loader.)

The Javadoc of java.lang.ClassLoader specifies that any class loader must first delegate the loading of a class to the parent, and only if this fails does it try to load the class itself.

So only 2 queries..?
1) Application containing JCN in respective Message flow still using Integration server class loader with isolation mode enable..?
2)Class Loader Hierarchy in IIB...?

I know some part of it(which is mentioned below), need to know from Application and Shared Library perspective.

Delegation direction:
(Integration server class loader) -> shared class loader -> system class loader -> Bootstrap class loader

(Class loader specified by configurable service) -> shared class loader -> system class loader -> Bootstrap class loader

Reply (Edit)

Simon Stone January 04, 2016

"1) Application containing JCN in respective Message flow still using Integration server class loader with isolation mode enable..?"

No, an application with Java isolation enabled will never delegate to the integration server wide class loader.

"2)Class Loader Hierarchy in IIB…?
I know some part of it(which is mentioned below), need to know from Application and Shared Library perspective.
Delegation direction:
(Integration server class loader) -> shared class loader -> system class loader -> Bootstrap class loader
(Class loader specified by configurable service) -> shared class loader -> system class loader -> Bootstrap class loader"

Application class loader with Java isolation enabled will delegate to the shared class loader (containing the shared-classes JARs), then to the system class loader, then to the bootstrap class loader.
A shared library class loader always delegates to the shared class loader.
There is no delegation between application and referenced shared library class loaders, and there is no delegation between shared library and referenced shared library class loaders.

Reply (Edit)

Yogesh Bhardwaj January 04, 2016

Thank you Simon. it make sense.

Thanks a lot!!

Reply (Edit)

Yogesh Bhardwaj December 22, 2015

Hi Simon, I would like to thank you for this post. Here I am adding some more pointers and queries. Would you please advise.
In context to "Integration projects"
1) "Integration projects" can only reference to "Statics Libraries" , it can't reference to "shared library".
2) "Statics Libraries" act as a shared library for "Integration projects" when it is deployed standalone and referenced by an "Integration projects".
3) When "Statics Libraries" are deployed standalone , it will use Integration server classloader as mentioned by you , "Java classes and JAR files within static libraries deployed directly to an integration server, outside of an application, are always loaded into the class loader for the integration server."

The solution you proposed sound logical to me but seems a problem for my case, let me put an example.
The first part of solution ,"migrate the Java code into a shared library which is then referenced by the applications". My application and Integration Project using ESQL to call "static log4J API java method" and I can't rely on "shared library" as I can't reference them from an "Integration Project".
One possibility is that I run application(static library bundled along with) with java isolation mode false, and used "static library" as standalone deployment , but with this I can't achieve true isolation and problem will remain same "which version is loaded"

Q1: Now as we know static lib can reference other Static lib only, and both will be using Integration server wide class loader hence no issue. What about "Shared library" reference by other "shared library " each is using own class loader. Eg:- shared1(jar1) , shared2(jar2) and shared1 reference to shared2, Will shread1 load the jar2 as well …? Based on definition I guess answer is Yes.

Q2: static lib can use "shared-class" path to load the reference jar, I am expecting it will not be a problem as both will user integration wide class Loader. What about "Shared library" which can use "shared-class" path to load the reference jar. For e.g. :- sharedLib1(jar1)(own class loader) and shared-class(jar2) (integration class loader) and sharedLib1(jar1) need to load shared-class(jar2) will shread1 load the jar2 as well …? I believer Answer is yes, what is your opinion on this..?

Now back to problem, if I think of unbundle my java classes available in Jar to place lo4j specific jar in shared-class path, could you please explain how sharred-class getting loaded(which loader is getting used), at what time and any "IBM JVM" specific consideration regarding this…?

Senario 2: I have a custom java class(YogiJavaComputeNode) which extend "MbJavaComputeNode" and implements MbJavaComputeNode's abstract method. It having one more abstract method declared so I need to declare as superclass in JCN (java class* -> browse). My all application containing JCN using this, what will the class loader consideration for this. NOTE:- I want my application to be java isolated for some other java module.

Reply (Edit)

Sadhana September 22, 2015

Hi Simon –

We have recently moved to version 10.0 of IIB and seem to have run into a weird issue of class loaders. We have App1 , App2 ,referring to a shared library , Lib1 that contains a few JCNs that access JAVA classes from Lib1Java project .

We have deployed Lib1 in a single Integration server where both App1 and App2 are also deployed. When we run tests on App1 and App2 , we have the following issue :

```
****************************************
BIP4394E: Java exception: ‘com.ibm.broker.plugin.MbUserException’;
thrown from class name: ‘com.XX.XX.common’, method name: ‘evaluate()’, file:
‘XXX.java’, line: ’59’; trace text: ‘com.ibm.broker.
config.proxy.ConfigManagerProxyLoggedException: mqsiprofile has not
been run in this environment: LocalIAPIConnect (Library is already
loaded in another ClassLoader)’; resource bundle:
‘XXXX.java’; key: ‘Error in getting XXXX
details’; inserts(optional): ‘{8}’, ‘{9}’, ‘{10}’, ‘{11}’, ‘{12}’,
‘{13}’, ‘{14}’, ‘{15}’, ‘{16}’, ‘{17}’

***************************
```

Does this mean that the JAVA isolation is not turned on for the Apps ?
This error occurs only on our Production Environment

Reply (Edit)

Simon Stone September 25, 2015

Hi Sadhana,

It looks you are deploying IntegrationAPI.jar as part of your applications? Am I correct? In V10, we added some JNI code to IntegrationAPI.jar which is used when connecting to local brokers. Because we use IntegrationAPI.jar internally, the JNI code is causing a clash with your deployed version, leading to the error.

If you remove the deployed copy of IntegrationAPI.jar, then the problem should go away. You do not need to deploy this JAR, as we already put it on the Java class path for you. I appreciate that you need to add IntegrationAPI.jar to the build path in the Integration Toolkit, as we don’t make it available by default. Unfortunately in most cases this also causes the JAR file to be packaged into the BAR file.

We’d like to make some changes in this area; I see you have a PMR open, and I’ll look at getting it routed to the L3 team to progress this issue further.

Regards, Simon.

Reply (Edit)

Sadhana September 25, 2015

Simon – This would be the first time someone has actually identified the root cause . Wow !!! Bravo !!
Thanks a million for having taken the time to take a look at the PMR. We have been having this issue in Production and have tried to scrap the integration server and go with a new one completely as a temporary workaround .

Could I get a hold of your IBM email Id ? Please ?

Reply (Edit)