Base Language - Feature #6407

name_map.xml improvements

05/25/2022 11:08 AM - Greg Shah

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Galya B	**% Done:**	20%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #6649: improve the performance or SourceNa...	**Test**

History

#1 - 05/25/2022 11:18 AM - Greg Shah

We have applications which support 4GL code extensions that are not part of the core application itself. For example, customer/partner-written OO 4GL classes which extend application classes/implement application interfaces and which need to be loaded/accessed at runtime.

Although we do allow the propath to be assigned dynamically, we don't have any provision for the name_map.xml to be "sharded" (split across multiple files). Doing so would allow extension jars to be added on a per-installation basis and the name map entries for that extension jar would be loaded/added from the shard that resides in that extension jar.

Another improvement is to move to a load-time scanning approach where we build some (or all?) of the name mappings from annotations when the application code is loaded. If this does not generate too much of a performance hit to implement, then it is the preferred approach. In a perfect world, we would have no name_map.xml at all and there would be little cost in server startup time. If we go with this approach, then we should ensure that the extension jar case is handled as well.

Does each session need its own mappings based on what is in the propath? The application described above does not do that, but technically it would be possible in the 4GL.

#2 - 06/06/2022 04:11 AM - Constantin Asofiei

6129a/13926 adds lazy loading of the legacy converted classes, to improve the FWD server startup time and also not load the entire .class for the legacy converted classes into the JVM, at once.

A summary of the current approach is this:

- at startup, SourceNameMapper loads the entire name_map.xml. For legacy converted classes, this is a registry of all converted classes (mapping of their Java name to legacy name), plus any virtual functions defined in the legacy class.
- for legacy converted classes, they structure is built when they are loaded by the application code (this is the 'lazy loading' approach).
- for external programs, although the LegacySignature and others are emitted in the converted code, we still rely on name_map.xml to build the entire structure, at server startup.

What improvements can be made:

- refactor name_map.xml to use the same approach as for the legacy classes (emit only the virtual functions/internal procedures and other metadata which can't exist in the converted Java code)
- refactor SourceNameMapper to use a lazy-loading of the external program, and rely on the LegacySignature annotations
- another improvement would be to emit some kind of special constructs for the virtual functions/internal procedures directly in the converted code,

> and leave name_map.xml only as a registry for mapping web services and converted code to their Java counterpart.

#4 - 03/12/2024 05:41 PM - Greg Shah

We need to implement something soon due to some customer deadlines. I don't think we have time to implement the full annotations-based approach right now. Instead, I propose that we implement a refactored version of our loading process. The idea is we should support more than one name_map.xml found in the jars, with the loading process resulting in the same in-memory representation we have today, but just having been loaded from multiple files.

#5 - 03/12/2024 05:45 PM - Greg Shah

- Related to Feature #6649: improve the performance or SourceNameMapper runtime added

#6 - 03/28/2024 03:47 PM - Greg Shah

- Assignee set to Galya B

#7 - 03/28/2024 03:50 PM - Greg Shah

> another improvement would be to emit some kind of special constructs for the virtual functions/internal procedures directly in the converted code, and leave name_map.xml only as a registry for mapping web services and converted code to their Java counterpart.

This is the right thing to do. Even for the web services, can't we include enough annotations so that we can build the "registry" by scanning? In a perfect world, there would be no name map at all and so long as our scanning process works across multiple jars, there is no issue with splitting conversion into many pieces.

#8 - 03/28/2024 03:58 PM - Constantin Asofiei

Greg Shah wrote:

> > another improvement would be to emit some kind of special constructs for the virtual functions/internal procedures directly in the converted code, and leave name_map.xml only as a registry for mapping web services and converted code to their Java counterpart.
>
> This is the right thing to do. Even for the web services, can't we include enough annotations so that we can build the "registry" by scanning? In a perfect world, there would be no name map at all and so long as our scanning process works across multiple jars, there is no issue with splitting conversion into many pieces.

We had this discussion before - moving **everything** to annotations will mean the entire converted .class code needs to be scanned and loaded into the JVM.

So, if we still don't want to load the entire app into the JVM at server startup, the goal is for name_map.xml be just a registry/mapping of converted program files/classes, and every other detail is in annotations.

#9 - 03/28/2024 05:09 PM - Greg Shah

> We had this discussion before - moving **everything** to annotations will mean the entire converted .class code needs to be scanned and loaded into the JVM.

A good point. I don't remember this discussion but certainly we have to load the classes if we use runtime annotations. We are planning to pre-load a non-trivial percentage of the code via appcds, but we don't want to preload everything.

> So, if we still don't want to load the entire app into the JVM at server startup, the goal is for name_map.xml be just a registry/mapping of converted program files/classes, and every other detail is in annotations.

I guess so.

#10 - 03/29/2024 07:08 AM - Galya B

MultiClassLoader has a list of all jars in the classpath. It can look for name_map.xml in each jar (with jar tvf filename.jar) and store the packages with name_map.xml in a list. Then use the packages in SourceNameMapper.initMappingData to load data in p2jMap, etc. Also search through them on each request for resolving something. ~~The external uses of SourceNameMapper.getPackageRoot() will have to be reworked (ControlFlowOps, SoapHandler)~~. If there are any clashes pkgroot will take precedence.

Probably some details will come up with implementation, but am I on the right track?

#11 - 03/29/2024 08:31 AM - Greg Shah

> MultiClassLoader has a list of all jars in the classpath. It can look for name_map.xml in each jar (with jar tvf filename.jar) and store the packages with name_map.xml in a list.

We don't need to use a child process for this search. We can search in the jars directly. We have code in Utils for searching/reading resources out of jar files.

> Probably some details will come up with implementation, but am I on the right track?

Yes

#12 - 03/29/2024 08:41 AM - Constantin Asofiei

Galya, what package would you use for looking into multiple jars? Look everywhere? The problem with name_map.xml is that it can appear in multiple jars, under the same pkgroot. When app-by-app conversion is done (and after that we ran these in a single JVM), we don't want each sub-app to have its own pkgroot.

I would expect for the approach to look like this:

- look under pkgroot inside all jars
- load them into SourceNameMapper, one by one, and address conflicts - if there are conflicts, and the details in a name_map.xml are different than what was already loaded, log a warning.

Also, SourceNameMapper will not load the .class and annotations when the entry in name_map.xml is processed - this needs to be done lazily, when that converted program is first used.

#13 - 03/29/2024 09:27 AM - Galya B

Constantin Asofiei wrote:

> Galya, what package would you use for looking into multiple jars? Look everywhere? The problem with name_map.xml is that it can appear in multiple jars, under the same pkgroot. When app-by-app conversion is done (and after that we ran these in a single JVM), we don't want each sub-app to have its own pkgroot.

Well, looking for one path in multiple jars is more problematic. Then I can't read the file from the classpath, but the jar needs to be unarchived and the xml file found on the file system and read.

#14 - 03/29/2024 09:29 AM - Galya B

That is of course if the check for the file listed by jar tvf filename.jar gives positive.

#15 - 03/29/2024 09:37 AM - Greg Shah

Why can't we use the same technique from Utils.searchResourceJars()? We don't need the propath part but we definitely don't want to use jar as a child process or unarchive things.

#16 - 03/29/2024 09:38 AM - Constantin Asofiei

You don't need to unzip the jar - Java has tools to read an archive. This just means we can't rely on Class.getResource, and instead we need to check if jar if it has a zip-entry with this exact path and name (pkgroot/name_map.xml)

#17 - 03/29/2024 09:45 AM - Galya B

Constantin Asofiei wrote:

> Java has tools to read an archive.

If you mean the custom FWD's JarClassLoader, then it should work.

#18 - 03/29/2024 09:46 AM - Galya B

Constantin Asofiei wrote:

> Also, SourceNameMapper will not load the .class and annotations when the entry in name_map.xml is processed - this needs to be done lazily, when that converted program is first used.

Which is 'that converted program'? We don't know what's inside the name map before it's read. Since all name maps will be in the same package, they need to be read at the same time, otherwise you won't know for sure what the classloader has loaded.

#19 - 03/29/2024 10:03 AM - Galya B

SourceNameMapper.initMappingData seems to be executed only once for the lifespan of the runtime machine with a server start hook, so that's where the checks will be performed over all the maps at the same time. Am I missing something?

#20 - 03/29/2024 10:20 AM - Galya B

- Status changed from New to WIP

6407a created from trunk r15101.

#21 - 03/29/2024 10:57 AM - Constantin Asofiei

Galya B wrote:

> SourceNameMapper.initMappingData seems to be executed only once for the lifespan of the runtime machine with a server start hook, so that's where the checks will be performed over all the maps at the same time. Am I missing something?

Correct.

> Which is 'that converted program'? We don't know what's inside the name map before it's read. Since all name maps will be in the same package, they need to be read at the same time, otherwise you won't know for sure what the classloader has loaded.

I mean we don't load Class.forName (to look at annotations) until that converted program is targeted by a RUN statement.

#22 - 04/03/2024 03:55 AM - Galya B

Constantin Asofiei wrote:

- load them into SourceNameMapper, one by one, and address conflicts - if there are conflicts, and the details in a name_map.xml are different than what was already loaded, log a warning.

I'm not sure what the criteria for equality is. For example servicePrograms are of type ExternalProgram and have a lot of properties. Do you expect the comparison to go to the lowest level or just log a warning on attempt to add a second service program with the same pname that may or may not be exactly the same?

#23 - 04/03/2024 05:22 AM - Constantin Asofiei

I don't mean the annotations in the .class - I mean the state from name_map.xml must match if there is an existing entry from a previous name_map.xml

In the end, name_map.xml must end up only with a very short registry of just legacy name to Java names.

#24 - 04/03/2024 05:33 AM - Galya B

Constantin Asofiei wrote:

> I don't mean the annotations in the .class - I mean the state from name_map.xml must match if there is an existing entry from a previous name_map.xml

servicePrograms gets populated by reading name_map.xml. What annotations? buildExternalProgram is done during reading the map and populates pname, jname, ooname, publishedEvents, ieMap, p2jf, j2pf, main that is InternalEntry with its own parameters populated.

So all of this should be matching?

#25 - 04/03/2024 05:43 AM - Constantin Asofiei

The point was to move to annotations/other structures inside the .java, here I mean all info from name_map.xml; ideally, we would be left just with:

```
  <class-mapping jname="Test" pname="test.p"/>
```

Until this is done, the entire class-mapping node must match - the key to match should be the same Java converted class name.

#26 - 04/03/2024 05:49 AM - Galya B

Constantin Asofiei wrote:

> The point was to move to annotations/other structures inside the .java

Constantin, maybe you've missed why I'm assigned to the task, that is in #6407-4.

Greg Shah wrote:

> We need to implement something soon due to some customer deadlines. I don't think we have time to implement the full annotations-based approach right now. Instead, I propose that we implement a refactored version of our loading process. The idea is we should support more than one name_map.xml found in the jars, with the loading process resulting in the same in-memory representation we have today, but just having been loaded from multiple files.

#27 - 04/03/2024 05:49 AM - Constantin Asofiei

Galya B wrote:

> Constantin, maybe you've missed why I'm assigned to the task, that is in #6407-4.

OK, thanks for the remainder. Then the entire class-mapping node needs to match if it exists in multiple name_map.xml.

#28 - 04/03/2024 05:53 AM - Galya B

Constantin Asofiei wrote:

> Galya B wrote:
>
> > Constantin, maybe you've missed why I'm assigned to the task, that is in #6407-4.
>
> OK, thanks for the remainder. Then the entire class-mapping node needs to match if it exists in multiple name_map.xml.

And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.

#29 - 04/03/2024 05:57 AM - Constantin Asofiei

Galya B wrote:

> And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.

If the entries are different, then there is something very bad with the conversion. This is not a warning, is a fatal error.

#30 - 04/03/2024 05:57 AM - Galya B

Constantin Asofiei wrote:

> Galya B wrote:
>
> > And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.
>
> If the entries are different, then there is something very bad with the conversion. This is not a warning, is a fatal error.

So the server should exit in this case?

#31 - 04/03/2024 06:00 AM - Constantin Asofiei

Yes.

#32 - 04/03/2024 06:06 AM - Galya B

OK, then I need one more clarification. Since there is a separate container rest-service, where programs get resolved by adding postfix -fwd-rest-service.p to pnames, do we consider these different from the regular class-mapping with the same pname or if the pname is the same without the postfix this is a clash?

#33 - 04/03/2024 06:16 AM - Constantin Asofiei

> REST services written directly in Java can be configured via the 'rest-service' element.

So these nodes are not from conversion, but added via name_map_merge.xml - they can be ignored for this match.

#34 - 04/03/2024 10:23 AM - Greg Shah

> And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.

If the entries are different, then there is something very bad with the conversion. This is not a warning, is a fatal error.

Won't this happen anytime there is an overlap of the same relative program names in 2 different apps?

#35 - 04/03/2024 10:25 AM - Galya B

Greg Shah wrote:

> > And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.
>
> If the entries are different, then there is something very bad with the conversion. This is not a warning, is a fatal error.
>
> Won't this happen anytime there is an overlap of the same relative program names in 2 different apps?

If the class info is the same, there will be no clash, no warning and exit.

#36 - 04/03/2024 10:26 AM - Galya B

I don't have an actual example though. Not sure if the path is supposed to be the same in two different jars.

#37 - 04/03/2024 10:30 AM - Constantin Asofiei

Greg Shah wrote:

> And all of this just for a warning log? I would say it's more reasonable to log the warning always when a second entry with the same pname is found.

If the entries are different, then there is something very bad with the conversion. This is not a warning, is a fatal error.

Won't this happen anytime there is an overlap of the same relative program names in 2 different apps?

The 'pname' in class-mapping is the full path relative to the abl/ folder - we don't use relative legacy names in name_map.xml, they are all 'full' relative to abl/.

More:

- if the same .p/.cls (as full name relative to abl/) is converted in multiple modules, then is assumed that **is the same physical file** in all modules. I don't know how we can enforce this.
- all modules need to convert with the same root package name, as 'jname' is relative to the root Java package.

#38 - 04/03/2024 10:45 AM - Greg Shah

> The 'pname' in class-mapping is the full path relative to the abl/ folder

Yes, this is why I called them relative paths.

> if the same .p/.cls (as full name relative to abl/) is converted in multiple modules, then is assumed that **is the same physical file** in all modules. I don't know how we can enforce this.

Let's consider this from the perspective of how it works for this multi-app case in OE. If both are compiled separately and then included in the same runtime installation, each .r would actually be in a different path right? Thus the runtime propath would be able to differentiate them. I suspect we need to make these unique by adding a per-app directory.

> all modules need to convert with the same root package name, as 'jname' is relative to the root Java package.

I wonder if this is right. Similar to the pname case, perhaps we should be having a unique root package (app-level sub-packages)?

#39 - 04/03/2024 10:55 AM - Galya B

Originally (currently) if multiple paths are found for the same filename when resolving the name map, the paths end up in MultiPathLookup, that returns null for clashes on lookup:

```
      public String lookupPath(String legacyProgName, String[] propath, boolean caseSens)
      {
         String result = null;
         for (SinglePathLookup spl : references)
         {
            String possibleResult = spl.lookupPath(legacyProgName, propath, caseSens);
            if (result == null && possibleResult != null)
            {
               result = possibleResult;
               continue;
            }

            // The result is already set, so any possible result is a conflict and should return null
            // in order to fallback to the old implementation
            if (possibleResult != null)
            {
               return null;
            }
         }
         return result;
      }
```

#40 - 04/03/2024 11:03 AM - Constantin Asofiei

Greg Shah wrote:

> Let's consider this from the perspective of how it works for this multi-app case in OE. If both are compiled separately and then included in the same runtime installation, each .r would actually be in a different path right? Thus the runtime propath would be able to differentiate them. I suspect we need to make these unique by adding a per-app directory.

I don't understand. Are you talking about duck-type-ing? If a .r for file F is needed by two modules, and is compiled in module M1 and module M2, then F.r must be in the same path, even if is included in more than one .pl.

If you are saying that module M1 has a program foo/bar/F.p and module M2 has a program foo/bar/F.p. which are completely different programs and unrelated to each other, then these need to be differentiated via specific folders in abl/. And the PROPATH (static or at runtime) is responsible for making sure the correct one is picked up.

> > all modules need to convert with the same root package name, as 'jname' is relative to the root Java package.

> I wonder if this is right. Similar to the pname case, perhaps we should be having a unique root package (app-level sub-packages)?

What about the API (interfaces) which are common for 2 or more modules, which are converted in each module? We can't change the Java package for these.

#41 - 04/03/2024 11:12 AM - Galya B

This is actually more complicated than it looked at first. We're trying to combine different apps coming possibly from different propaths in OE and having diverging files included into one FWD instance. In java this is the same as having multiple versions of the same dependency in the classpath. Constantin seems to be against the idea of supporting this and I can understand why, thinking of all the issues we have with this in java. But on the other hand how can we make sure the customer is not compiling different apps with different versions of the same file.

#42 - 04/03/2024 11:14 AM - Greg Shah

> If you are saying that module M1 has a program foo/bar/F.p and module M2 has a program foo/bar/F.p. which are completely different programs and unrelated to each other, then these need to be differentiated via specific folders in abl/. And the PROPATH (static or at runtime) is responsible for making sure the correct one is picked up.

Exactly. But in OE, they don't have to do that **if they install the compiled results in different directories**. So: if in OE, the separate modules are installed in different directories, then we need to add those directories at conversion time. If we add that requirement, it will work in FWD.

On the other hand, it might be that the customer builds multiple apps separately in OE but then installs them all in the same location. In such a case, the conflicting .r files would overwrite each other, with the last one installed being the one that "wins". In that case, we can implement a precedence approach as well, with the last one loaded winning. As long as the loading order in FWD is the same as the installation order in OE, it will work the same way.

Either way, we can (and should) resolve this without abending the server.

> What about the API (interfaces) which are common for 2 or more modules, which are converted in each module? We can't change the Java package for these.

A good point. Especially because of the OO4GL stuff but even for procedures, any truly common code must have the same paths in any app being converted. This makes it more important to ensure that we have a clean way to handle conflicts as noted above.

#43 - 04/03/2024 11:20 AM - Greg Shah

> In java this is the same as having multiple versions of the same dependency in the classpath.

Not exactly. In Java, the same exact class name can't be loaded twice in the same JVM. That is different from OE. The Java classnames are fully qualified and only one will win.

There are 2 cases in OE. Case 1 is the "installed in different locations" and that is perfectly fine in OE. You can in fact run both versions if you setup your propath differently for each RUN statement. Case 2 is the "overwriting compiled results" case and in that case, there is only 1 version of that program at runtime so it wins.

> Constantin seems to be against the idea of supporting this and I can understand why, thinking of all the issues we have with this in java. But on the other hand how can we make sure the customer is not compiling different apps with different versions of the same file.

I think we can resolve both cases in a manner that is roughly equivalent to the OE result.

#44 - 04/03/2024 11:32 AM - Galya B

Case 2 is the same as java and can cause malfunctioning programs like in java, this is what I meant.
Case 1 is fine in FWD too, the only requirement is to have in the jar the same directory for name_map.xml as declared in pkgroot. Other than that the jar can have a completely different package for the actual classes.

#45 - 04/03/2024 11:34 AM - Galya B

If we need to load the jars in a specific order, we'll need explicit instructions, probably new config in directory.

#46 - 04/03/2024 11:36 AM - Galya B

Galya B wrote:

> Other than that the jar can have a completely different package for the actual classes.

Actually not quite, since we support only one pkgroot, but anyways the new app can be nested, it's not relevant to where the name_map.xml is.

#47 - 04/03/2024 12:58 PM - Galya B

One pname corresponds to only one jname, because that's how conversion seems to work. The question is if the other attributes of the mapping are the same or different between the two places where the mappings are configured. In any case only one java class will be loaded in the runtime classpath, so only one of those configs will be valid. This is not something we can decide even with configs and Constantin was right that it is a major issue.

As of how the same file is loaded from different paths, I think this is already handled by the lookup in MultiPathLookup.

#48 - 04/03/2024 01:02 PM - Greg Shah

> This is not something we can decide even with configs and Constantin was right that it is a major issue.

Case 1 can be handled at conversion time by adding an app-specific path segment to make the filenames unique. There is no change needed for runtime support.

Case 2 can be handled by implementing a precedence order to honor the first loaded or the last loaded.

#49 - 04/03/2024 01:08 PM - Galya B

I speak about case 3: different maps for different files with different attributes, but same path. Obviously if you load the wrong file it won't match the map.

#50 - 04/03/2024 01:09 PM - Galya B

Galya B wrote:

> I speak about case 3: different maps for different files with different attributes. Obviously if you load the wrong file it won't match the map.

and one of the two apps won't work, even if we implement precedence and load one

#51 - 04/03/2024 01:28 PM - Galya B

To solve case 2. we need the precedence order to be fed to the jmv via startup args, so that the classloader can take it in consideration in findClass and loadClass.

To solve case 3. probably best to abend or one of the apps will malfunction.

#52 - 04/03/2024 02:26 PM - Constantin Asofiei

We had cases of the same OE qualified class name used in different modules. The qualified legacy name can't be changed by our conversion, no matter if we place it in a different Java package. The solution for this was for the customer to change the legacy 4GL code, so that there are no collisions in qualified legacy class names.

To be more specific, the collision was for a .cls in module M3 and M2, which both got bundled separately as (M1, M3) and (M2, M3) when deployed in OpenEdge, but in FWD we bundled them all as (M1, M2, M3).

Otherwise, to detail more my concerns:

- we have M1 dependent on I2 (API interfaces) for module M2
- in OpenEdge, M1 and I2 are used for a build (not deploy) and (M2, I2) for the other build.
- if a customer requires M1, the .pl for M1 just gets released alongside the .pl for (M2, I2) (the .pl is like a .jar in Java)
- in FWD, we want to convert (M1, I2) and (M2, I2) separately
- however we place the I2 classes in the abl/ folder, this must be the same for both cases, the conversion for (M1, I2) and the conversion for (M2, I2) must produce the same Java qualified class names for the code in I2.
- I don't know if this works or not, but it's worth noting: we could convert first just I2, add this jar to the conversion classpath, and FWD should be able to convert just M1 without the I2 legacy code. But I don't know if this would work or not (haven't tested).
- the conclusion: when legacy code for inter-module dependencies are converted via multiple modules, the generated Java qualified class name (and its actual code actually) must match
  - for .cls is obvious, we break the Java code if we use different Java packages for the same interface in different modules
  - for .p is a little more subtle: you will end up with duplicated Java code if the same .p is converted in module M1 under abl/module1/foo/bar.p and under M2 under abl/module2/foo/bar.p

So, my problem is not what package root we use for the converted code which is not shared between modules - we use this paradigm today, we create different abl/ ~~folders~~ sub-folders and place the .cls and .p under that. My problem is if we want to automate this, what happens (and how we distinguish) that some .cls/.p is shared code or not.

Also, about the .r code - I agree with Greg that 'is possible' for a deploy which just uses .r code and copies the files (in a sequence) to some folder (overwriting files maybe), and the result is the application. But, in practice, I think .pl archives are used more - and depending on the PROPATH, runtime could use a .p from different .pl files, even if the full path matches.

#53 - 04/03/2024 02:29 PM - Greg Shah

> But, in practice, I think .pl archives are used more - and depending on the PROPATH, runtime could use a .p from different .pl files, even if the full path matches.

Yes, the .pl scenario is "case 1" and "just works" in OE.

#54 - 04/03/2024 02:34 PM - Constantin Asofiei

Greg Shah wrote:

> > But, in practice, I think .pl archives are used more - and depending on the PROPATH, runtime could use a .p from different .pl files, even if the full path matches.
>
> Yes, the .pl scenario is "case 1" and "just works" in OE.

And in FWD the abl/ sub-folders where we place each module is the equivalent of the .pl 'namespace'.

#55 - 04/03/2024 03:08 PM - Constantin Asofiei

And I think I understand now Greg's point about the .r and folder deploy in OpenEdge:

- in FWD, you would convert the module code without abl/ sub-folders, and whatever mapping in name_map.xml is first for that pname (or maybe some rules), that would be 'it'; a warning can be logged for this. But, even if we have different Java packages for the sub-modules, the converted .class will still exist for all the pnames which match. And if we don't different Java packages for sub-modules, then is a matter of the Java classpath to place the 'proper' jar in the correct position in the classpath, otherwise JVM will load the wrong .class.
- but, my original concern is for the case when shared code is being converted in multiple modules; in this case, if there is a mismatch, then that is bad, and a warning I don't think is enough. You could end up with the same Java qualified class name for two different legacy classes.

For classes, if the mapping of qualified Java name and legacy name are the same, then considering that we can't check all the annotations in the class (or its actual content), a warning I think it should be logged, also (maybe on a lower level like FINE).

For external programs, on a second thought, we will end up with all internal entries configurations at annotations/in the converted .class, so I think we can keep the same, look at only the class-mapping attributes.

#56 - 04/04/2024 04:36 AM - Galya B

Constantin Asofiei wrote:

> And I think I understand now Greg's point about the .r and folder deploy in OpenEdge:
>
> - in FWD, you would convert the module code without abl/ sub-folders, and whatever mapping in name_map.xml is first for that pname (or maybe some rules), that would be 'it'; a warning can be logged for this. But, even if we have different Java packages for the sub-modules, the converted .class will still exist for all the pnames which match. And if we don't different Java packages for sub-modules, then is a matter of the Java classpath to place the 'proper' jar in the correct position in the classpath, otherwise JVM will load the wrong .class.

case 2: the precedence order to be fed to the jmv via startup args, so that the classloader can take it in consideration in findClass and loadClass.

- but, my original concern is for the case when shared code is being converted in multiple modules; in this case, if there is a mismatch, then that is bad, and a warning I don't think is enough. You could end up with the same Java qualified class name for two different legacy classes.

case 3: probably best to abend or one of the apps will malfunction.

> For classes, if the mapping of qualified Java name and legacy name are the same, then considering that we can't check all the annotations in the class (or its actual content), a warning I think it should be logged, also (maybe on a lower level like FINE).
>
> For external programs, on a second thought, we will end up with all internal entries configurations at annotations/in the converted .class, so I think we can keep the same, look at only the class-mapping attributes.

One java name can have only one mapping. And legacy names are resolved by the java name. So a critical mismatch leading to server exit is the same jname, but different ooname or pname. A duplicate entry is the same jname with the same ooname and pname and results in a FINE log and we expect the issue to be solved by the jar precedence in the classloader. Is this right?

#57 - 04/04/2024 04:39 AM - Galya B

P.S. The previous comment got updated.

#58 - 04/04/2024 05:45 AM - Constantin Asofiei

From what you posted in #6407-56, only one concern: the granularity for the precedence order I think needs to be "this .class from this .jar first", not "this .jar first".

#59 - 04/04/2024 09:21 AM - Galya B

Constantin Asofiei wrote:

> From what you posted in #6407-56, only one concern: the granularity for the precedence order I think needs to be "this .class from this .jar first", not "this .jar first".

Something like this would be very verbose if a clash in a whole module needs to be resolved:
-Dclassloader.order=com.example.Class1:first.jar,second.jar;com.example.Class2:first.jar,second.jar.

#60 - 04/04/2024 09:28 AM - Greg Shah

Do we really need an explicit list of precedence order? Why not implicitly take it from the classpath and honor the first found?

#61 - 04/04/2024 09:29 AM - Galya B

Greg Shah wrote:

> Do we really need an explicit list of precedence order? Why not implicitly take it from the classpath and honor the first found?

The classpath has the jars loaded randomly. What is there to do then? Just end up with whatever.

#62 - 04/04/2024 09:37 AM - Constantin Asofiei

Galya B wrote:

> Greg Shah wrote:
>
> > Do we really need an explicit list of precedence order? Why not implicitly take it from the classpath and honor the first found?
>
> The classpath has the jars loaded randomly. What is there to do then? Just end up with whatever.

My understanding is that JVM keeps the same order of jars as defined in classpath (it does not use a random order). I've been adding bin/ to the start of propath before p2j.jar since forever for my local testing and it always worked.

If our classloaders are doing something different, then that may be a problem at some point.

#63 - 04/04/2024 09:39 AM - Galya B

Constantin Asofiei wrote:

> My understanding is that JVM keeps the same order of jars as defined in classpath (it does not use a random order). I've been adding bin/ to the start of propath before p2j.jar since forever for my local testing and it always worked.
>
> If our classloaders are doing something different, then that may be a problem at some point.

MultiClassLoader finds the class by iterating HashSet. You can't rely on a specific order of iteration, even less to be similar to the java implementation. I wouldn't bet even on all the clashing classes to be loaded from the same jar.

#64 - 04/04/2024 09:42 AM - Galya B

We can rework it to iterate jars alphabetically.

#65 - 04/04/2024 09:57 AM - Galya B

Galya B wrote:

> MultiClassLoader finds the class by iterating HashSet.

Actually it's a collection, backed by the HashMap, so the order is still decided by the internal order of a HashMap. So what I'm saying is still valid.

#66 - 04/04/2024 11:20 AM - Galya B

- Status changed from WIP to Review

- % Done changed from 0 to 20

6407a r15107 based on trunk r15101 ready for review: adds support for reading mappings from multiple name_map.xml files found in the pkgroot in any jar in the classpath.

#67 - 04/04/2024 11:32 AM - Greg Shah

> > MultiClassLoader finds the class by iterating HashSet.
>
> Actually it's a collection, backed by the HashMap, so the order is still decided by the internal order of a HashMap. So what I'm saying is still valid.

If we only add one match to the collection for each fully qualified class, then there is no issue. We just have to ensure we add in the classpath order.

#68 - 04/04/2024 11:39 AM - Galya B

Greg Shah wrote:

> MultiClassLoader finds the class by iterating HashSet.

> Actually it's a collection, backed by the HashMap, so the order is still decided by the internal order of a HashMap. So what I'm saying is still valid.

> If we only add one match to the collection for each fully qualified class, then there is no issue. We just have to ensure we add in the classpath order.

It's a HashMap of jars, the loader iterates over the values, that are jar classloaders. The first jar having the class wins. So it's random.

#69 - 04/04/2024 11:43 AM - Galya B

So what I'm saying is that there is no classpath order in my humble opinion. Iterating HashMap is not ordered.

#70 - 04/04/2024 11:45 AM - Galya B

MultiClassLoader:

```
/** Map containing as key the jar name and as value its associated class loader. */
private final Map<String, JarClassLoader> loaders = new HashMap<>();

/** A set containing all jars added to the classpath when the server was started. */
private static final Set<String> classpathJars = JarUtil.resolveClasspathJars(null);
```

Do you want to replace these with lists?

#71 - 04/04/2024 12:12 PM - Galya B

I did replace the Set with List in r15108.

Tomorrow I'm testing again in testcases. A good list of different procedures is converted. Then I put a breakpoint in SourceNameMapper on init complete and compare what's loaded in the maps with:

- running the original code
- the new branch - the map is split in two and a new jar added. Then try to run a few different procs from both maps and evaluate their success.

#72 - 04/15/2024 02:19 AM - Galya B

Reminder: The branch needs a review (at r15110). It also has minor changes for #8166-145 and #8166-150.

#73 - 04/15/2024 07:03 AM - Galya B

6407a rebased on trunk r15146.
New r15156 with a minor fix for #8166-151.

#74 - 04/25/2024 04:01 AM - Constantin Asofiei

Review for 6407a rev 15157:

- WebServer - please add a history entry
- Utils.packageToPath - this replaces all dots with File.separatorChar - on Windows, this is '\'. Is this valid when working with jar resources? My understanding is that jar resources are always using '/' linux-style separator.
- SourceNameMapper - convertJavaProg is used during conversion via NameMappingWorker.classNameExists, to check for collisions. This requires to read the name_map.xml file from the file system, and not jar. This needs to be fixed.

Otherwise, we need to do some conversion testing with these changes. And some small runtime tests.

#75 - 04/25/2024 05:13 AM - Galya B

Constantin Asofiei wrote:

> Review for 6407a rev 15157:
>
> - WebServer - please add a history entry

Fixed.

> - Utils.packageToPath - this replaces all dots with File.separatorChar - on Windows, this is '\'. Is this valid when working with jar resources? My understanding is that jar resources are always using '/' linux-style separator.

The reason I had to make Utils.packageToPath to be OS specific is JarClassLoader.containsResource check in SourceNameMapper.initMappingData that compares the concatenated path for NAME_MAP_FILE with the resource paths in JarClassLoader.resources coming from the classpath that has OS specific file dividers.

Now that I look closer, I actually need to change how the first char is stripped off the resource name in JarClassLoader.containsResource and getResourceName to work for Windows.

> - SourceNameMapper - convertJavaProg is used during conversion via NameMappingWorker.classNameExists, to check for collisions. This requires to read the name_map.xml file from the file system, and not jar. This needs to be fixed.

I'm not sure what the change is here?

> Otherwise, we need to do some conversion testing with these changes. And some small runtime tests.

I did manually conversions of a few procedures and it works. What do we look for? I'm not sure what impact is expected on conversion.

#76 - 04/25/2024 05:32 AM - Constantin Asofiei

Galya B wrote:

- SourceNameMapper - convertJavaProg is used during conversion via NameMappingWorker.classNameExists, to check for collisions. This requires to read the name_map.xml file from the file system, and not jar. This needs to be fixed.

> I'm not sure what the change is here?

If there is a name_map.xml on the file system, use that, too. My understanding from the code is that only jars are used for name_map.xml

> I did manually conversions of a few procedures and it works. What do we look for? I'm not sure what impact is expected on conversion.

We need to test full conversion of some customer projects.

#77 - 04/25/2024 06:16 AM - Galya B

Constantin Asofiei wrote:

> Galya B wrote:
>
> - SourceNameMapper - convertJavaProg is used during conversion via NameMappingWorker.classNameExists, to check for collisions. This requires to read the name_map.xml file from the file system, and not jar. This needs to be fixed.
>
> I'm not sure what the change is here?
>
> If there is a name_map.xml on the file system, use that, too. My understanding from the code is that only jars are used for name_map.xml

Where is that? Not in the original initMappingData as far as I can see. NameMappingWorker provides only two args pkgroot and name that is The class name we want to check for existence.. initMappingData only works with one path, that is URL url = ClassLoader.getSystemResource(sb.toString());, so the resource is always read from the ClassLoader and that is the classpath, i.e. the jars.

#78 - 04/25/2024 07:22 AM - Constantin Asofiei

You are correct, at this point, unless the current folder is in the classpath, ClassLoader.getSystemResource(sb.toString()); will not find a name_map.xml from file-system. The goal of classNameExists was to ensure that we can emit unqualified FWD classes in conversion safely (this is used just for Action at this point); I think it may have been broken from the beginning, testing just from Eclipse where bin/ is in the classpath may have been the 'wrong indication' that it was working.

#79 - 04/25/2024 07:51 AM - Constantin Asofiei

With the classNameExists in mind, these changes don't affect conversion.

#80 - 04/25/2024 07:53 AM - Galya B

Constantin Asofiei wrote:

> You are correct, at this point, unless the current folder is in the classpath, ClassLoader.getSystemResource(sb.toString()); will not find a name_map.xml from file-system. The goal of classNameExists was to ensure that we can emit unqualified FWD classes in conversion safely (this is used just for Action at this point); I think it may have been broken from the beginning, testing just from Eclipse where bin/ is in the classpath may have been the 'wrong indication' that it was working.

Actually if a dir with name map file is in the classpath the name map should have been found in the original code in the dir under the custom package root (NameMappingWorker.pkgroot defaults to empty if not set in Configuration), while currently it only iterates the jars. If such scenario exists, then I can add back a simple ClassLoader.getSystemResource, but only in case of no name maps found in jars, otherwise it will get complicated. How does it sound?

#81 - 04/25/2024 02:24 PM - Constantin Asofiei

Galya B wrote:

> Actually if a dir with name map file is in the classpath the name map should have been found in the original code in the dir under the custom package root (NameMappingWorker.pkgroot defaults to empty if not set in Configuration), while currently it only iterates the jars. If such scenario exists, then I can add back a simple ClassLoader.getSystemResource, but only in case of no name maps found in jars, otherwise it will get complicated. How does it sound?

Yes, but I don't think we ever set . or other dir in the classpath, in the build scripts we have for conversion.

Try two programs named action.p and action2.p with just this content:

```
block-level on error undo, throw.

message "x".
```

I suspect with or without 6407a, this will not emit BlockManager.Action.THROW, but only Action.THROW.

#82 - 04/26/2024 02:34 AM - Galya B

Constantin Asofiei wrote:

> Galya B wrote:
>
> > Actually if a dir with name map file is in the classpath the name map should have been found in the original code in the dir under the custom package root (NameMappingWorker.pkgroot defaults to empty if not set in Configuration), while currently it only iterates the jars. If such scenario exists, then I can add back a simple ClassLoader.getSystemResource, but only in case of no name maps found in jars, otherwise it will get complicated. How does it sound?
>
> Yes, but I don't think we ever set . or other dir in the classpath, in the build scripts we have for conversion.
>
> Try two programs named action.p and action2.p with just this content:
> [...]
>
> I suspect with or without 6407a, this will not emit BlockManager.Action.THROW, but only Action.THROW.

With or without 6407a I get:

```
Action.java:25: error: cannot find symbol
    [javac]          onBlockLevel(Condition.ERROR, Action.THROW);
```

Not sure what it means.

Is there something else I need to do for this task?

#83 - 04/26/2024 02:37 AM - Constantin Asofiei

Galya B wrote:

> With or without 6407a I get:
> [...]
>
> Not sure what it means.

There is ambiguity between the Action.java for converted code and BlockManager.Action unqualified inner class name.

> Is there something else I need to do for this task?

I don't think so.

Regarding the Utils.packageToPath change - please check runtime on Windows for i.e. Hotel GUI.

#84 - 04/26/2024 06:25 AM - Galya B

Constantin Asofiei wrote:

> Regarding the Utils.packageToPath change - please check runtime on Windows for i.e. Hotel GUI.

I can't test hotel_gui (or any web ui) in a VM, because the web server gets the url from the ip of the network interface, that is something 10.xx.xx.xx and then the certificate doesn't match and the browser gives ERR_SSL_PROTOCOL_ERROR that can't be avoided.
I've tested to convert one procedure with testcases and it works. Not sure what else to test in Win.

#85 - 04/26/2024 06:26 AM - Constantin Asofiei

Galya B wrote:

> Not sure what else to test in Win.

Try the Swing client for Hotel GUI on Windows.

#86 - 04/26/2024 08:00 AM - Galya B

There is an issue with the file separators indeed. It's \ on Windows for the jars on the file system, but the resources in the jars always use Linux style separators. Fixed in r15159.

I think this is the only issue related to the task. I couldn't start the hotel_gui swing client because there is something going on with ant import.db, but it's not related:

```
C:\code\hotel_gui\build_db.xml:42: java.io.IOException: Cannot run program "C:\JavaCoretto\jdk1.8.0_402\jre\bi
n\java.exe" (in directory "C:\code\hotel_gui"): CreateProcess error=206, The filename or extension is too long
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:1048)
```

And it's not the java path. Not sure what is misconfigured. I haven't run it before on Windows.

#87 - 04/26/2024 08:29 AM - Galya B

OK, the java command for the task create.db.h2 exceeds 8191 chars (the max length of commands in windows) due to the classpath:

```
   <!-- path used when running application related tasks-->
   <path id="app.classpath">
      <fileset dir="${fwd.lib.home}"    includes="*.jar"/>
      <fileset dir="${deploy.home}/lib" includes="*.jar"/>
   </path>
```

When one of the two dirs is removed, the task gets through.

#88 - 04/26/2024 10:38 AM - Galya B

I don't know what's going on now, but the client doesn't even spawn and I can't stop it in the cmd prompt. No errors in logs. jstack suggests a warning msg is to be displayed, but no signs of any window. I think the warning should be the one I've just described in #8689:

```
"main" #1 prio=5 os_prio=0 tid=0x000002cc5f647800 nid=0xf54 in Object.wait() [0x000000ec93bfe000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    at com.goldencode.p2j.ui.client.TypeAhead.getKeystroke(TypeAhead.java:447)
    - locked <0x0000000714c3a088> (a com.goldencode.p2j.ui.client.TypeAhead)
    at com.goldencode.p2j.ui.chui.ThinClient.modalEventLoopWorker(ThinClient.java:10324)
    at com.goldencode.p2j.ui.chui.ThinClient.modalEventLoop(ThinClient.java:10199)
    at com.goldencode.p2j.ui.chui.ThinClient.lambda$messageBox$36(ThinClient.java:10136)
    at com.goldencode.p2j.ui.chui.ThinClient$$Lambda$386/1544518128.run(Unknown Source)
    at com.goldencode.p2j.ui.chui.ThinClient.withIndependentEventList(ThinClient.java:10250)
    at com.goldencode.p2j.ui.chui.ThinClient.messageBox(ThinClient.java:10106)
    at com.goldencode.p2j.ui.chui.ThinClient.messageBox(ThinClient.java:9951)
    at com.goldencode.p2j.ui.chui.ThinClient.messageBox(ThinClient.java:9882)
```

```
    at com.goldencode.p2j.ui.chui.ThinClient.displayWarningMessage(ThinClient.java:9811)
    at com.goldencode.p2j.ui.ErrorWriterInteractive.displayWarning(ErrorWriterInteractive.java:138)
    at com.goldencode.p2j.util.ErrorManager.displayWarning(ErrorManager.java:3079)
    at com.goldencode.p2j.ui.chui.ThinClient.displayWarning(ThinClient.java:9575)
    at com.goldencode.p2j.ui.ClientExportsMethodAccess.invoke(Unknown Source)
    at com.goldencode.p2j.util.MethodInvoker.invoke(MethodInvoker.java:156)
    at com.goldencode.p2j.util.TraceHelper.trace(TraceHelper.java:145)
    at com.goldencode.p2j.net.Dispatcher.trace(Dispatcher.java:1083)
    at com.goldencode.p2j.net.Dispatcher.processInbound(Dispatcher.java:787)
    at com.goldencode.p2j.net.Conversation.block(Conversation.java:422)
    at com.goldencode.p2j.net.Conversation.waitMessage(Conversation.java:348)
    at com.goldencode.p2j.net.Queue.transactImpl(Queue.java:1221)
    at com.goldencode.p2j.net.Queue.transact(Queue.java:682)
    at com.goldencode.p2j.net.BaseSession.transact(BaseSession.java:273)
    at com.goldencode.p2j.net.HighLevelObject.transact(HighLevelObject.java:221)
    at com.goldencode.p2j.net.RemoteObject$RemoteAccess.invokeCore(RemoteObject.java:1466)
    at com.goldencode.p2j.net.InvocationStub.invoke(InvocationStub.java:144)
    at com.sun.proxy.$Proxy3.standardEntry(Unknown Source)
    at com.goldencode.p2j.main.ClientCore.start(ClientCore.java:495)
    at com.goldencode.p2j.main.ClientDriver.start(ClientDriver.java:284)
    at com.goldencode.p2j.main.CommonDriver.process(CommonDriver.java:593)
    at com.goldencode.p2j.main.ClientDriver.process(ClientDriver.java:378)
    at com.goldencode.p2j.main.ClientDriver.main(ClientDriver.java:425)
```

Constantin, do I need to make hotel_gui or testcases run on Windows as part of this task, because this is far off as of now?

#89 - 04/26/2024 11:17 AM - Greg Shah

- Status changed from Review to Internal Test

> do I need to make hotel_gui or testcases run on Windows as part of this task

No. The Windows situation will not be resolved here. Windows is in a bad state for Hotel due to a lack of attention AND to the fact that some things are just nasty on Windows. But mostly because of lack of attention.

As you've been doing, create regression testing bug reports for anything you've found. We'll defer that work.

Constantin: Are we good to go?

#90 - 04/26/2024 11:27 AM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Are we good to go?

My main concern with Windows was that jar file separator which needs to be linux-style, and this was solved. Otherwise, if some testing with customers app works, I'm OK with the changes.

#91 - 04/29/2024 04:56 AM - Galya B

6407a rebased on trunk r15171.

Constantin Asofiei wrote:

> Otherwise, if some testing with customers app works, I'm OK with the changes.

I've run the code with the m project and the app loads fine. I'm not sure what else to look for.

#92 - 04/30/2024 08:12 AM - Greg Shah

Constantin: Are there any reasons why this can't be merged to trunk?

#93 - 04/30/2024 08:12 AM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Are there any reasons why this can't be merged to trunk?

No, it can be merged.

#94 - 04/30/2024 08:27 AM - Greg Shah

- Status changed from Internal Test to Merge Pending

6407a can be merged after 8486a.

#95 - 04/30/2024 08:45 AM - Galya B

- Status changed from Merge Pending to Test

6407a was merged to trunk as rev. 15180 and archived.

Support for reading mappings from multiple name_map.xml files in classpath jars.