Base Language - Bug #4741

RUN statement with a path to a procedure that does not have any extension fails to be found

07/03/2020 05:45 PM - Roger Borrello

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Roger Borrello	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 07/03/2020 05:49 PM - Roger Borrello

There is an issue about finding an external procedure that is not passed to RUN with an extension. I added this testcase:

uast/run/run-noext-driver.p

```
# This shows that FWD cannot find an external procedure when the extension is not included

# This should run fine (provided you included it in your build)
run run/run-noext-victim.p.

pause.

# This cannot be found in FWD, but is fine in 4GL
run run/run-noext-victim.

message "Done.".
```

uast/run/run-noext-victim.p

```
# This is an innocent victim of the run-noext-driver.p.
# Make sure you have your legacy propath correct (".:testcase/uast" or similar).
message "Hello dummy!".
```

Constantin's notes:
There is this TODO in SourceNameMapper.convertName:

```
      // TODO: RUN can target r-code even if extension is missing; i.e. RUN foo. will execute
      // foo.r, if it exists
```

I think you need to change the lookupDirectName and lookupAliasedDirectName, to work without the extension.

#2 - 07/03/2020 05:52 PM - Roger Borrello

Constantin asked:

Roger, rename run-noext-victim.p to run-noext-victim.k.p and do a RUN run-noext-victim.k. Does this work in 4GL? Also run run/run-noext-victim.k

No, neither worked in 4GL.

Depending on the answer, lookupDirectName may need to fallback on the 'no extension' only in certain cases.

OK, so make the change to lookupDirectName to fallback searching the p2jNonExtMap map, and see what happens. Keep the same warning as we have in lookupCompiledRcode, if multiple files match the same name, without extension.

#4 - 07/03/2020 05:54 PM - Roger Borrello

He was talking about SourceNameMapper.convertName at line 1395.

The usage needs to be added to lookupDirectName function.

#5 - 07/04/2020 01:00 PM - Roger Borrello

I forgot to use this task for this...

I asked about the existing code making an assumption that it is rcode. However, we know that 4GL can perform compilation "on-the-fly" by just passing a ".p" without the extension. What set of extensions do we need to check for? ".r", ".p", and ".w"?

Constantin replied, We don't know what the real extension is, at conversion time - that's why the runtime needs to look into the filenames without the extension. There may be a PROPATH lookup involved, too, but for now, if an application is able to uniquely resolve the program (from the filename without the extension), this change is enough; and add these details (about PROPATH lookup) to the SourceNameMapper code and #4741

#6 - 07/06/2020 09:06 AM - Roger Borrello

In the debugger I noticed that the procedure name came into the convertName method first, without a path in front of it, then again with the first path in the PROPATH. Perhaps the PROPATH is already taken into account?

My modifications are in 3821c-11385 for review. I included checks for .w and .p. Are there any other extensions to look for?

#7 - 07/06/2020 09:16 AM - Constantin Asofiei

Roger Borrello wrote:

> In the debugger I noticed that the procedure name came into the convertName method first, without a path in front of it, then again with the first path in the PROPATH. Perhaps the PROPATH is already taken into account?
>
> My modifications are in 3821c-11385 for review. I included checks for .w and .p. Are there any other extensions to look for?

Roger, only the .r extension is something which is hard-coded in 4GL, AFAIK. You can't decide to remove the extension and look for the 'extensionless' name.

Instead, you need to do this:

1. if the exact match of pname is found, use that.
2. otherwise, remove its extension (if it has one) and look in the p2jNonExtMap map.

#8 - 07/06/2020 09:49 AM - Roger Borrello

Constantin Asofiei wrote:

> Instead, you need to do this:
>
> 1. if the exact match of pname is found, use that.
> 2. otherwise, remove its extension (if it has one) and look in the p2jNonExtMap map.

How does this look?

```
      lookupDirectName = (pname ->
      {
         ExternalProgram extProg = null;

         extProg = p2jMap.get(pname);  // Check for direct name before looking for noExt name

         if (extProg == null)
         {
            pname = (noExt) ? pname : pname.substring(0, pname.length() - ".p".length());

            // we could use a try, but keeping a match of names without extension to the full
            // program names (with extension) is easier
            List<ExternalProgram> progs = p2jNonExtMap.get(pname);

            if (progs != null && !progs.isEmpty())
            {
               extProg = progs.get(0);

               if (LOG.isLoggable(Level.WARNING))
               {
                  for (int i = 1; i < progs.size(); i++)
                  {
                     LOG.log(Level.WARNING, String.format(msg,
                                                          extProg.pname,
                                                          legacyProgName,
                                                          progs.get(i).pname));
                  }
               }
            }
         }

         return extProg;
      });
```

#9 - 07/06/2020 09:51 AM - Constantin Asofiei

Change the test to this: if (extProg == null && pname.indexOf('.') < 0).

#10 - 07/06/2020 10:04 AM - Roger Borrello

Is it possible to have . in your path (or filename) that might fail that test?

#11 - 07/06/2020 08:27 PM - Roger Borrello

Checked in updates to 3821c-11389 that don't have the pname.indexOf('.') < 0) check. I am open to changing, just needed to understand about whether or not that might fail for paths with '.' within them.

#12 - 07/07/2020 05:57 AM - Greg Shah

Relative paths ./ and ../../something/else/../blah/ are possible. It is also legal to include . in random parts of the paths like this/is.a/do.t/l.a.d.e.n/path/program.

#13 - 07/07/2020 06:18 AM - Constantin Asofiei

Greg, you are correct. We need more testcases to find every case.

For now, can it be enough to check if the last index of '.' is the second-to-last char in the string? (i.e. the program has a single char extension)?

#14 - 07/07/2020 09:02 AM - Roger Borrello

Constantin Asofiei wrote:

> Greg, you are correct. We need more testcases to find every case.
>
> For now, can it be enough to check if the last index of '.' is the second-to-last char in the string? (i.e. the program has a single char extension)?

What are we protecting against by doing that? In 4GL, you can run run/run-noext-victim.pee and as long as the procedure exists, it's fine.

There are 2 cases to cover:

1. Explicit filename passed (with various path options)
2. Filename passed without an extension

We have to limit our set of extensions. We don't really care if there is a ".r", but if there is a ".w" and ".p", we may have to enforce a precedence determined by testcases.

Right now, I am having trouble with the testcases, because even though I have completely removed the -victim files, the -driver still finds some internally compiled procedure to run. I thought by deleting everything in C:\TEMP in my VM environment it would clean that up, but it still finds something to run.

#15 - 07/07/2020 09:25 AM - Constantin Asofiei

Roger Borrello wrote:

> ... but if there is a ".w" and ".p", we may have to enforce a precedence determined by testcases.

It can be a .x, .k, .tt

> Right now, I am having trouble with the testcases, because even though I have completely removed the -victim files, the -driver still finds some internally compiled procedure to run. I thought by deleting everything in C:\TEMP in my VM environment it would clean that up, but it still finds something to run. 4GL doesn't care about the extension, RUN will work! Only .r is special in terms that if the extension is missing, it looks for .r.

#16 - 07/07/2020 09:26 AM - Constantin Asofiei

> It can be a .x, .k, .tt

And 4GL doesn't care about the extension, RUN will work - only .r is assumed if the extension is missing. But FWD doesn't have a physical .r file - so we need to check for the program name without its extension, if it matches.

#17 - 07/27/2020 03:40 PM - Roger Borrello

I'm looking again at this task due to what is going on with #4208. Specifically this code in convertName:

```
      Function<String, Boolean> isRcode =
         (pname -> pname.endsWith(".r") || (!caseSens && pname.endsWith(".R")));
```

The lookupCompiledRCode code is:

```
      lookupCompiledRcode = !legacyRcode ? null : (pname ->
      {
.
.
.
```

Something doesn't seem correct regarding isRcode, and my concern is because I leveraged the same methodology for the expanded

lookupDirectName function.

If the purpose is to determine whether or not the passed in legacyProgName is r-code (or p-code, with respect to my additions), what effect does the case-sensitivity have on that determination? I'm not sure including caseSens is even necessary to see if the legacyProgName ends in an 'r' or 'R' (or 'p' or 'P').

#18 - 07/27/2020 03:50 PM - Constantin Asofiei

Roger, do a Windows OS test with a compiled file named some-program.r and try RUN some-program.r and RUN some-program.R. After that, rename some-program.r to some-program.R and see if RUN some-program.r and RUN some-program.R will work in 4GL.

Again, AFAIK in 4GL only the r-code extension is hard-coded and meaningful for the 4GL interpreter, as a RUN some-program will still find the some-program.r. But a RUN other-program **will not find** a other-program.p or other-program.w.

In FWD, we don't work with the r-code (.r file), so when we have a RUN some-program or RUN some-program.r, we must look for the program file name which was included in the conversion (be it .p, .w, or whatever extension or even with no extension, as this is possible in 4GL).

#19 - 07/27/2020 10:27 PM - Roger Borrello

4GL on Windows was fine with the below, no matter what I named the run-wrongcase-victim.r.

```
run run/run-WrongCase-victim.p.
run run/run-WrongCase-victim.r.
run run/run-WrongCase-victim.R.
run run/run-WrongCase-victim.
```

My question was more around just trying to include caseSens in the check for whether or not a procedure is rCode or pCode. Wouldn't this achieve the same results:

```
      Function<String, Boolean> isRcode = (pname ->
            pname.endsWith(".R") || pname.endsWith(".r")
      );
      boolean legacyRcode = isRcode.apply(legacyProgName);
      Function<String, Boolean> isPcode = (pname ->
            pname.endsWith(".P") || pname.endsWith(".p") || pname.endsWith(".W") || pname.endsWith(".w")
      );
      boolean legacyPcode = isPcode.apply(legacyProgName);
```

The reason I am around looking into this, is that my search for a procedure is going through here, and not matching up on the ExternalProgram extProg = p2jMap.get(pname); check, even though it should. I am brought to the conclusion that the p2jMap can not account for both case-sensitive and case-insensitive. We probably should keep both hash-maps.

#20 - 07/28/2020 04:08 AM - Constantin Asofiei

Roger, again, FWD **must not** match the extension for a non-compiled 4GL program. This is because the extension can be anything, 4GL does not limit the external program extensions to .p or .w - that's just a convention. Only the r-code (.r or .R) must be checked, as this seems to be something hard-coded in the 4GL interpreter.

About p2jMap - you are correct, the key is dependent on case-sensitivity:

```
               // remove case if required
               if (!caseSens)
               {
                  pname = pname.toLowerCase();
               }
```

I don't want to duplicate this map in every context. Instead, build two maps - one with the original key and one with the lowercased key. Same for p2jNonExtMap. And when a FWD context wants to access this map, it uses either the case-sensitive or case-insensitive one, based on the caseSens context-local flag.

#21 - 07/28/2020 08:42 AM - Roger Borrello

Moved these to the WorkArea. Can they still be static?

```
      /**
       * Name element separator used for the Progress file name data.  At runtime this is OS
       * independent value used internally in name_map.xml file to resolve procedure names.
       */
      private String fileSep = "/";

      /** Path separator used for the Progress file name data. */
      private String pathSep = null;

      /** Case sensitivity flag for the Progress file name data. */
      private Boolean caseSens = null;
```

I have added helpers:

```
boolean caseSens = getLegacyCaseSensitive();
String pathSep   = getLegacyPathSeparator();
String fileSep   = getLegacyFileSeparator();
```

Each of these is similar to the below, although checking for fileSep = null is useless, since it is initialized.

```
   private static String getLegacyFileSeparator()
   {
```

```
      WorkArea wa = local.get();

      if (wa.fileSep == null)
      {
          wa.fileSep = EnvironmentOps.getLegacyFileSeparator();
      }

      return wa.fileSep;
   }
```

In initMappingData, I put the caseSens, pathSep, and fileSep at the top, before the p2jMap != null check.

I added 2 new HashMaps:

```
            p2jMap_ci = new HashMap<>(1024);
            p2jNonExtMap_ci = new HashMap<>(1024)
```

I also create a 2nd pname (pname_ci) in the loop through classMap, and instead of performing lowercase conditionally, I do it explicitly on the new String.

```
                // remove case
                pname_ci = pname_ci.toLowerCase();
```

Storing the bidirectional mappings is where I am getting lost (if all the above is OK)... this is what I've added:

```
                // store bidirectional mappings for the progress relative
                // filename to/from the java relative file name mapping
                ExternalProgram ep = buildExternalProgram(pname, jname, classMap);
                ExternalProgram ep_ci = buildExternalProgram(pname_ci, jname, classMap);
                ep.withServices = "true".equalsIgnoreCase(classMap.getAttribute(WITH_SERVICES));
                ep_ci.withServices = "true".equalsIgnoreCase(classMap.getAttribute(WITH_SERVICES));

                p2jMap.put(ep.pname, ep);
                p2jMap_ci.put(ep_ci.pname, ep_ci);
                j2pMap.put(ep.jname, ep);

                String nonExtName = pname;
                String nonExtName_ci = pname_ci;
                int lastDotIdx = nonExtName.lastIndexOf('.');
                if (lastDotIdx > 0)
                {
                   nonExtName = nonExtName.substring(0, lastDotIdx);
                   nonExtName_ci = nonExtName_ci.substring(0, lastDotIdx);
                }

                List<ExternalProgram> progs = p2jNonExtMap.get(nonExtName);
                List<ExternalProgram> progs_ci = p2jNonExtMap_ci.get(nonExtName_ci);
                if (progs == null)
                {
                   p2jNonExtMap.put(nonExtName, progs = new ArrayList<>(1));
                   p2jNonExtMap_ci.put(nonExtName_ci, progs_ci = new ArrayList<>(1));
                }

                progs.add(ep);
                progs_ci.add(ep_ci);
```

In the methods that are performing lookups into the map, I am using:

```
      initMappingData();
      boolean caseSens = getLegacyCaseSensitive();
```

```
    ExternalProgram extProg = caseSens ? p2jMap.get(pname) : p2jMap_ci.get(pname.toLowerCase());
```

What is the result is that I am now not finding procedures that I had no issues with in the past. So something is quite wrong.

#22 - 07/28/2020 08:58 AM - Greg Shah

> Moved these to the WorkArea. Can they still be static?

No. The idea of the WorkArea is that it is context-local. This means it is a per-session thing. Static means JVM-wide which defeats the purpose. That is what we are trying to avoid.

I don't see an obvious problem with the rest.

#23 - 07/28/2020 09:15 AM - Roger Borrello

Greg Shah wrote:

> > Moved these to the WorkArea. Can they still be static?
>
> No. The idea of the WorkArea is that it is context-local. This means it is a per-session thing. Static means JVM-wide which defeats the purpose. That is what we are trying to avoid.

OK. I had taken that declaration out.

> I don't see an obvious problem with the rest.

Is there some thing with what context would get construct which WorkArea? The server token is associated with the AppServer, and the client token is associated with the gui. Could those values be pulled from the directory every time, instead of being cached locally?

#24 - 07/28/2020 09:19 AM - Constantin Asofiei

Roger, just the key needs to be different. ExternalProgram.pname needs to keep the original name (it was wrong to lowercase pname for Windows OS), not the lowercased one. You can reuse the ExternalProgram value in the two maps.

#25 - 07/28/2020 09:40 AM - Roger Borrello

Constantin Asofiei wrote:

> Roger, just the key needs to be different. ExternalProgram.pname needs to keep the original name (it was wrong to lowercase pname for Windows OS), not the lowercased one. You can reuse the ExternalProgram value in the two maps.

I think this is it...

```
            for (int i = 0; i < len; i++)
            {
               Element classMap = (Element) cmaps.item(i);

               String pname = classMap.getAttribute(ATTR_PNAME);
               String pname_ci = classMap.getAttribute(ATTR_PNAME);
               String jname = classMap.getAttribute(ATTR_JNAME);

               if (pname == null || pname.length() == 0 || jname == null || jname.length() == 0)
               {
                  String errmsg = "Malformed class mapping entry for " + classMap.toString();
                  throw new IllegalStateException(errmsg);
               }

               // remove case
               pname_ci = pname_ci.toLowerCase();

               // store bidirectional mappings for the progress relative
               // filename to/from the java relative file name mapping
               ExternalProgram ep = buildExternalProgram(pname, jname, classMap);
               ep.withServices = "true".equalsIgnoreCase(classMap.getAttribute(WITH_SERVICES));

               p2jMap.put(ep.pname, ep);
               p2jMap_ci.put(ep.pname_ci, ep);
               j2pMap.put(ep.jname, ep);

               String nonExtName = pname;
               String nonExtName_ci = pname_ci;
               int lastDotIdx = nonExtName.lastIndexOf('.');
               if (lastDotIdx > 0)
               {
                  nonExtName = nonExtName.substring(0, lastDotIdx);
                  nonExtName_ci = nonExtName_ci.substring(0, lastDotIdx);
               }

               List<ExternalProgram> progs = p2jNonExtMap.get(nonExtName);
               List<ExternalProgram> progs_ci = p2jNonExtMap_ci.get(nonExtName_ci);
               if (progs == null)
               {
                  p2jNonExtMap.put(nonExtName, progs = new ArrayList<>(1));
                  p2jNonExtMap_ci.put(nonExtName_ci, progs_ci = new ArrayList<>(1));
               }

               progs.add(ep);
            }
```

#26 - 07/28/2020 09:49 AM - Greg Shah

> FWD must not match the extension for a non-compiled 4GL program. This is because the extension can be anything, 4GL does not limit the external program extensions to .p or .w - that's just a convention. Only the r-code (.r or .R) must be checked, as this seems to be something hard-coded in the 4GL interpreter.

I think I understand your point, but we have a mismatch that needs to be resolved. Most applications that we convert are compiled in the 4GL. This means that all .p, .w and even .whatever files will be present ONLY as a .r in the runtime system. Although the 4GL may not have special extension processing in the interpreter, I think it can execute the resulting .r by passing either the original extension or by passing no extension, right?

In FWD, this no extension case is broken since we never have a .r there. Isn't Roger's fallback code needed to match this case?

#27 - 07/28/2020 10:01 AM - Roger Borrello

Not related to Greg's question, but my implementation...

This (obviously) doesn't compile, because there isn't a ep.pname_ci. I need to know why we are using the ExternalProgram.pname instead of just pname when we add that key. I'm planning on changing the 2nd line to p2jMap_ci.put(pname_ci, ep); but that got me thinking about the 1st line. Are they the same (ep.pname and pname)?

```
                p2jMap.put(ep.pname, ep);
                p2jMap_ci.put(ep.pname_ci, ep);
                j2pMap.put(ep.jname, ep);
```

#28 - 07/28/2020 10:33 AM - Constantin Asofiei

Greg Shah wrote:

> This means that all .p, .w and even .whatever files will be present ONLY as a .r in the runtime system. Although the 4GL may not have special extension processing in the interpreter, I think it can execute the resulting .r by passing either the original extension or by passing no extension, right?

Actually, is even weirder than this. I've compiled a program to some-prog.r, removed the original some-prog.p and all these work:

```
run some-prog.p.
run some-prog.r.
run some-prog.whatever.
run some-prog.
```

And looks like the .r has precedence even it actually there is a some-prog.whatever file in the same folder.

Going back to FWD - we convert the source-code files, so we have their original names (with extension). But the runtime can execute them by either specifying a (wrong) extension or no extension at all (depending on if the program was originally deployed as r-code or not). If there is a .r for this filename, then any or no extension will find it. I think the FWD runtime needs to know which programs had a r-code and which did not... as another attribute to name_map.xml - and this would mean we need to include in the abl/ folder the r-code and the source-code, too.

> In FWD, this no extension case is broken since we never have a .r there. Isn't Roger's fallback code needed to match this case?

Yes, when there is no extension specified, it can mean that either:

- the program exists 'as specified' in the PROPATH
- the program's source-code exists with another extension (which can be anything) and the runtime is supposed to use the .r for its r-code.

But, this works only and only if the filenames (without extensions) are unique in the entire codeset. I think this needs to be reworked and do a proper PROPATH lookup for the .r program first (which looks only for programs with the rcode attribute in name_map.xml annotation) and if not found, then look for the original name match.

#29 - 07/28/2020 10:43 AM - Greg Shah

> I think the FWD runtime needs to know which programs had a r-code and which did not... as another attribute to name_map.xml - and this would mean we need to include in the abl/ folder the r-code and the source-code, too.
> ...
> do a proper PROPATH lookup for the .r program first (which looks only for programs with the rcode attribute in name_map.xml annotation) and if not found, then look for the original name match.

I'm OK with making a flag, but the flag should be global. It can be in the directory. I have never seen an application which is run both compiled and from source code. I've only seen all compiled or all from source code. Until we get an application that is different, we don't need to track this per-program.

Most apps are compiled so we should default the flag to compiled mode.

#30 - 07/28/2020 02:18 PM - Roger Borrello

Is this another hash-map that has implications of not matching up with the case-sensitivity associated with the project-token?

```
   /** Map of aliases (r-code folders) to their source folders (conversion folders). */
   private static Map<String, String> pathAliases = null;
```

It was using caseSens at the time it was populated, and is within the "protection" of:

```
      if (p2jMap != null)
      {
         return;
      }
```

#31 - 07/28/2020 02:56 PM - Greg Shah

Roger Borrello wrote:

> Is this another hash-map that has implications of not matching up with the case-sensitivity associated with the project-token?
>
> [...]
>
> It was using caseSens at the time it was populated, and is within the "protection" of:
> [...]

Did you just answer your own question? :)

I think any of the data structures which have keys (or values or whatever) modified in initMappingData() based on caseSens would be something you would have to resolve.

#32 - 07/28/2020 03:28 PM - Roger Borrello

It was more like wishful thinking. There are so many issues getting through startup with this version, I must be missing another big-un.

The changes...

```
   /** Map of aliases (r-code folders) to their source folders (conversion folders). - case-insensitive */
   private static Map<String, String> pathAliases_ci = null;
```

```
   private static String[] resolvePathAliases(String pname)
   {
         Map<String,String> pA = getLegacyCaseSensitive() ? pathAliases : pathAliases_ci;
      return resolvePathAliases(pA, pname);
   }
```

```
   private static void loadPathAliases()
   {
.
.
.
         pathAliases = new HashMap<>();
         pathAliases_ci = new HashMap<>();
         for (String aliasPath : aliasPaths)
         {
            aliasPath = path + "/" + aliasPath;
            String source = ds.getNodeString(aliasPath, "source");
            String alias = ds.getNodeString(aliasPath, "alias");
            String source_ci;
            String alias_ci;

            source_ci = source.toLowerCase();
            alias_ci = alias.toLowerCase();

            pathAliases.put(alias, source);
            pathAliases_ci.put(alias_ci, source_ci);
         }
      }
      finally
      {
         ds.unbind();
      }
   }
```

It's can't be that straightforward, as we are still not getting through the client initialization.

#33 - 07/28/2020 05:04 PM - Roger Borrello

I am not sure why, but these changes to create the multiple maps for case-sensitive and case-insensitve in SourceNameMapper seem to have the side effect of breaking the ServerImpl construction that creates new requestInfo for the AppServer.

The requestInfo.ref is null, throwing NPE. This is the first connection made to the AppServer.

```
public ServerImpl()
{
   requestInfo.assign(ObjectOps.newInstance(OerequestInfo.class));

   requestInfo.ref.setClientContextId(new character(AppServerManager.nextContextId(false)));
}
```

Any thoughts on whether this was because we only had case-sensitive searches before my modifications?

#34 - 07/29/2020 05:55 PM - Roger Borrello

In trying to figure out why OO initialization is involved, it looks like the changes I have made are causing ObjectOps from being able to initialize some references. In ServerImpl constructor, requestInfo is left without a valid ref, and the setClientContextId method call causes a NPE.

When ControlFlowOps.initializeLegacyObject processes Progress.Lang.OERequestInfo, it eventually invokes the constructor (__lang_OerequestInfo_constructor__), which throws the exception when the code block is entered.

I am now keeping pathAliases and p2jMap in separate lists, but not jp2Map and classMap. I would think that the maps that take the legacy name in to lookup the Java names would need to have a case-insensitive variety, or should they **only** have a case-insensitive mapping?

#35 - 07/29/2020 06:37 PM - Roger Borrello

Traced this down to SourceNameMapper.getInternalEntry where we have null passed in as the pname. I cannot toLowerCase it... more to come after I fix that.

#36 - 07/30/2020 06:24 AM - Constantin Asofiei

Roger Borrello wrote:

> I am now keeping pathAliases and p2jMap in separate lists, but not jp2Map and classMap.

The key for j2pMap is case sensitive, as this is a Java class name. classMap doesn't have a String key, so no need to change that.

> Traced this down to SourceNameMapper.getInternalEntry where we have null passed in as the pname. I cannot toLowerCase it... more to come after I fix that.

What's the stacktrace for this SourceNameMapper.getInternalEntry call?

#37 - 07/30/2020 06:30 AM - Constantin Asofiei

Roger, a change which affects the lookup in p2jMap is that ExternalProgram.pname is no longer the key in the p2jMap. Any p2jMap lookup which gets an external pname (key) must prepare the key to be case (in)sensitive, before accessing the map.

#38 - 07/30/2020 11:31 AM - Roger Borrello

Constantin Asofiei wrote:

> Roger, a change which affects the lookup in p2jMap is that ExternalProgram.pname is no longer the key in the p2jMap. Any p2jMap lookup which gets an external pname (key) must prepare the key to be case (in)sensitive, before accessing the map.

Is that a change that is planned? Because the existing code uses:

```
                String pname = classMap.getAttribute(ATTR_PNAME);
                String jname = classMap.getAttribute(ATTR_JNAME);

                if (pname == null || pname.length() == 0 || jname == null || jname.length() == 0)
                {
                   String errmsg = "Malformed class mapping entry for " + classMap.toString();
                   throw new IllegalStateException(errmsg);
                }

                // remove case if required
                if (!caseSens)
                {
                   pname = pname.toLowerCase();
                }

                // store bidirectional mappings for the progress relative
                // filename to/from the java relative file name mapping
                ExternalProgram ep = buildExternalProgram(pname, jname, classMap);
                ep.withServices = "true".equalsIgnoreCase(classMap.getAttribute(WITH_SERVICES));

                p2jMap.put(ep.pname, ep);
                j2pMap.put(ep.jname, ep);
```

My planned change to this is:

```
                String pname = classMap.getAttribute(ATTR_PNAME);
                String pname_ci = pname.toLowerCase();
                String jname = classMap.getAttribute(ATTR_JNAME);

                if (pname == null || pname.length() == 0 || jname == null || jname.length() == 0)
                {
                   String errmsg = "Malformed class mapping entry for " + classMap.toString();
                   throw new IllegalStateException(errmsg);
                }

                // store bidirectional mappings for the progress relative
                // filename to/from the java relative file name mapping
                ExternalProgram ep = buildExternalProgram(pname, jname, classMap);
                ep.withServices = "true".equalsIgnoreCase(classMap.getAttribute(WITH_SERVICES));

                p2jMap.put(ep.pname, ep);
                p2jMap_ci.put(pname_ci, ep);
                j2pMap.put(ep.jname, ep);
```

I am going to check my changes in. Do you want to inspect before or after?

#39 - 07/30/2020 01:31 PM - Roger Borrello

Changes checked into 3821c-11426.

#40 - 07/30/2020 01:32 PM - Roger Borrello

- Status changed from New to WIP

- Assignee set to Roger Borrello

- % Done changed from 0 to 100

#41 - 04/20/2021 12:04 PM - Roger Borrello

- Status changed from WIP to Review

I cannot see any reason to leave this task open.

#42 - 04/20/2021 12:34 PM - Greg Shah

- Status changed from Review to Closed