Base Language - Feature #6649

improve the performance or SourceNameMapper runtime

08/01/2022 10:21 AM - Constantin Asofiei

Status:	Test	**Start date:**	
Priority:	Urgent	**Due date:**	
Assignee:	Dănuț Filimon	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #6407: name_map.xml improvements	**Test**

History

#1 - 08/01/2022 10:25 AM - Constantin Asofiei

SourceNameMapper APIs like getExternalProgram or getInternalEntry get called each time something dynamic is executed (like RUN, dynamic-function, new object, etc).

Find what can be improved in this class, or in the callers (like ControlFlowOps and ObjectOps - some caching may help there).

#2 - 01/04/2023 10:25 AM - Greg Shah

- Assignee set to Hynek Cihlar

#3 - 01/19/2023 11:21 AM - Hynek Cihlar

- Status changed from New to WIP

#4 - 01/20/2023 07:05 AM - Hynek Cihlar

6129b revision 14376 reimplements the case insensitive maps in SourceNameMapper with CaseInsensitiveHashMap.

#5 - 01/20/2023 09:47 AM - Hynek Cihlar

- % Done changed from 0 to 20

#6 - 01/23/2023 12:02 PM - Hynek Cihlar

Constantin, I didn't find any major performance issues profiling the big customer application. Can you please share more details about your findings (please send any sensitive info in the email)?

#7 - 01/24/2023 02:24 AM - Constantin Asofiei

- File getInternalEntry.png added

Bellow are the backtraces for SourceNameMapper.getInternalEntry - these are reduced after the InvokeConfig call-site caching, but please look into it if there is a way to reduce these calls in some other ways.

#8 - 01/24/2023 02:37 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Bellow are the backtraces for SourceNameMapper.getInternalEntry - these are reduced after the InvokeConfig call-site caching, but please look into it if there is a way to reduce these calls in some other ways.

OK, I see know what you meant. I can add a cache of the resolved internal entries in InternalEntryCaller. This seems to be safe as the internal entries don't seem to be changed during the app life time.

#9 - 01/24/2023 02:38 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin Asofiei wrote:
>
> > Bellow are the backtraces for SourceNameMapper.getInternalEntry - these are reduced after the InvokeConfig call-site caching, but please look into it if there is a way to reduce these calls in some other ways.
>
> OK, I see know what you meant. I can add a cache of the resolved internal entries in InternalEntryCaller. This seems to be safe as the internal entries don't seem to be changed during the app life time.

Any caching needs to be aware if the PROPATH has changed, because same relative path for a program name may resolve to another program, after PROPATH has changed.

#10 - 01/24/2023 02:53 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > Constantin Asofiei wrote:
> >
> > > Bellow are the backtraces for SourceNameMapper.getInternalEntry - these are reduced after the InvokeConfig call-site caching, but please look into it if there is a way to reduce these calls in some other ways.
> >
> > OK, I see know what you meant. I can add a cache of the resolved internal entries in InternalEntryCaller. This seems to be safe as the internal entries don't seem to be changed during the app life time.
>
> Any caching needs to be aware if the PROPATH has changed, because same relative path for a program name may resolve to another program, after PROPATH has changed.

Yes, the idea was to cache the absolute paths. I will see if it also makes sense to cache the relative paths with some kind of invalidation you mentioned.

#11 - 01/25/2023 03:48 PM - Hynek Cihlar

- % Done changed from 20 to 50

3821c revision 14382 introduces caching of resolved procedure internal entries. It speeds up tests in a large customer application by about 40 ms. Please review.

#12 - 01/25/2023 04:08 PM - Greg Shah

3821c?

#13 - 01/25/2023 04:13 PM - Hynek Cihlar

Greg Shah wrote:

> 3821c?

I meant 6129b. 3821c has burned into my cerebral cortex after the years working with the branch :-).

#14 - 01/30/2023 10:37 AM - Hynek Cihlar

I checked in various performance changes in 6129b revision 14394. Please review.

#15 - 02/01/2023 02:34 PM - Greg Shah

Code Review Task Branch 6129b Revision 14394

The changes look good.

#16 - 02/01/2023 02:34 PM - Greg Shah

Does SourceNameMapper.convertName() often get called with the same legacyProgName and lookup function? If so, we can implement caching to avoid the costly processing (all the path searching) of this method. We would have to invalidate the cache when the propath gets assigned but otherwise it seems like a potential win. The ControlFlowOps changes from #1970 would have reduced this greatly, but that doesn't mean that it eliminated these cases. Please instrument this method to check if this is the case or not.

#17 - 05/02/2023 10:31 AM - Greg Shah

It seems to me that nearly all the time in all projects the unqualified filename has very few conflicts in the overall project. Contrast this with the fact that most project directories have many files in them (hundreds and even thousands, especially when you consider subdirs). This suggests that a legacy name lookup based on first looking for the unqualified filename and then disambiguating any conflicts (the same file in multuple paths) would usually involve much less work than the current approach of searching through all paths (left to right) for the match.

#18 - 05/02/2023 10:36 AM - Hynek Cihlar

Greg Shah wrote:

> It seems to me that nearly all the time in all projects the unqualified filename has very few conflicts in the overall project. Contrast this with the fact that most project directories have many files in them (hundreds and even thousands, especially when you consider subdirs). This suggests that a legacy name lookup based on first looking for the unqualified filename and then disambiguating any conflicts (the same file in multuple paths) would usually involve much less work than the current approach of searching through all paths (left to right) for the match.

This sounds reasonable.

#19 - 05/03/2023 05:23 PM - Greg Shah

- Assignee changed from Hynek Cihlar to Joe Davis

#20 - 05/04/2023 09:42 AM - Hynek Cihlar

- File cache_internal_entries.diff added

Joe, I suggest you go with Greg's idea in #6649-17 first. This is relatively simple change and will be a good opportunity to get to know SourceNameMapper better.

I'm attaching a set of changes which were meant to cache resolved internal entries. This is a WIP and will need some profiling to see how much (if any) CPU time this will save.

Also I suggest you look at the trunk revision 14484 to get an idea what sort of performance changes have been done in this area.

For the rest you will need to fire up the profiler and focus on the hotspots of SourceNameMapper and the related classes (like ControlFlowOps).

#21 - 05/22/2023 07:01 PM - Joe Davis

- Status changed from WIP to Review

#22 - 05/23/2023 11:14 AM - Hynek Cihlar

Joe, I see you changed the status of this issue to Review. Are the changes checked in anywhere?

#23 - 05/23/2023 05:59 PM - Joe Davis

Hynek Cihlar wrote:

> Joe, I see you changed the status of this issue to Review. Are the changes checked in anywhere?

They should be checked into 6649a? Unless there's a problem with bzr on my side.

#24 - 05/24/2023 06:05 AM - Hynek Cihlar

Joe Davis wrote:

> Hynek Cihlar wrote:
>
> > Joe, I see you changed the status of this issue to Review. Are the changes checked in anywhere?
>
> They should be checked into 6649a? Unless there's a problem with bzr on my side.

The last revision I see in the branch is 14560, a merge from trunk. Did you check in your changes? If your branch is unbound, you will have to push them to the push branch.

#25 - 05/25/2023 11:45 AM - Greg Shah

Joe: Where does this work stand? I'd like to get the early review done as well as get an update on your subsequent optimizations.

#26 - 05/26/2023 11:57 AM - Joe Davis

Greg Shah wrote:

> Joe: Where does this work stand? I'd like to get the early review done as well as get an update on your subsequent optimizations.

The current status is that it's ready for review, but I'm trying to figure out what's wrong with my local bzr repo at present, as it's evidently not pushing on commit.

#27 - 05/26/2023 12:08 PM - Hynek Cihlar

Joe Davis wrote:

> Greg Shah wrote:
>
> > Joe: Where does this work stand? I'd like to get the early review done as well as get an update on your subsequent optimizations.
>
> The current status is that it's ready for review, but I'm trying to figure out what's wrong with my local bzr repo at present, as it's evidently not pushing on commit.

What do you get with bzr info?

#28 - 06/05/2023 06:04 AM - Hynek Cihlar

Hynek Cihlar wrote:

> Joe Davis wrote:
>
> > The current status is that it's ready for review, but I'm trying to figure out what's wrong with my local bzr repo at present, as it's evidently not pushing on commit.
>
> What do you get with bzr info?

Joe, any news on this? Did you manage to fix the branch?

#29 - 09/13/2023 08:16 AM - Greg Shah

- % Done changed from 50 to 0

- Status changed from Review to WIP

- Assignee deleted (Joe Davis)

#30 - 09/14/2023 08:17 AM - Dănuț Filimon

I did a test on a large customer application for #6649-16 where I got a total of **42846** calls to SourceNameMapper.convertName and investigated the type of functions and the legacyProgName used:

- A total of **1620** different legacyProgName values were passed as an argument;
- From the **1620** entries which I will refer from now on, **430** were called **5 or more** times;
- All entries used a single function when searching the file **(!)**;
- convertNames returned **null** 16982 times, and returned successfully **25864** times when using lookupDirectName. lookupAliasesDirectName and lookupAliasesCOmpiledRcode did not return anything;
- Looking at the legacyProgName, there are path like **test/test1.p**, but also **test\testfile1.p**;
- The top 3 most used legacyProgName values were called **5486**, **2957** and **1203** times, and took **72**, **62** and **22** milliseconds to complete;

#31 - 09/15/2023 08:56 AM - Dănuț Filimon

- Assignee set to Dănuț Filimon

From #6649-16, I understood that we should execute the lookup function with the unqualified filename, then resume searching using the multiple paths that are appended to generate other lookup paths. Does this mean that we have firstly to do a simple check of the legacyProgName (case 3 in the SourceNameMapper.convertName(String, Function) method)? Or should all the lookup functions be executed and check the unqualified name, then execute the functions again but now verifying the multiple paths available? I have an almost complete implementation for both cases, but I just want to confirm if I approached the issue correctly. The cache implementation is done (there will be a cache for each local context since the propath is also stored in the WorkArea) and I just need to do the test from #6649-30 again to see if there is any improvement.

#32 - 09/15/2023 10:14 AM - Greg Shah

I think the idea from #6649-17 has the greatest potential for improvement, though the caching from #6649-16 might also make a good contribution.

The #6649-17 idea is about leveraging the fact that in most applications, there is most often just 1 instance of a given unqualified program name in an entire application. Even when that is not the case, ususally there are very few instances of a given unqualified name. This means that reversing the lookup logic to fast-path the program name and ignore the pathing, is likely to be a big boost. I'd focus on that first, before the caching.

Even this #6649-17 can be done in a kind of 2 phase approach. Perhaps we implement a program-level "lookup helper" which gets instantiated for every separate unqualified program name. We could then lookup this helper in a map of unqualified program name keys (String) to program lookup helper instances. That helper could just implement a single lookup method which would have 2 implementations:

- Highlander case ("there can be only one") where the unqualified program name is a uniquely identifiable program in the entire application. This would just return the full path for that case, with no other processing needed other than confirming that any pre-pended path in the original search spec is not in conflict.
- Multi-path case where the original path of the search spec is checked against each of the possible matches and the correct one is returned.

Any mismatch and null is returned which would mean that there is no program that matches.

You'd need to consider to what degree we can avoid the propath scanning in this process. As much as possible, we would want to minimize it since it is so costly.

From there I'd like to see the performance benefit. After that we would want to consider the caching. I don't have a specific idea of the caching in mind but would want you to implement whatever would yield the best result.

#33 - 09/15/2023 10:18 AM - Greg Shah

Another thing to consider with this "search inversion" idea is whether we can avoid the propath scanning process by removing any propath match from the front of the search spec (by "search spec", I mean the original text passed in by the 4GL code that is driving the lookup). If the leftmost text matches anything in the propath, we can remove it and then the resulting text should match something in our name map or there is no match.

#34 - 09/19/2023 04:41 AM - Alexandru Lungu

Let me see if I understand your point correctly.

Bootstrap

- We have a map with keys as unqualified procedure names (like prop.p) and PathLookup instances.
- We iterate the name-map.xml registry and initialize the map:
  - For any proc.p entry, we accumulate all of its referrals (i.e. <path1>/proc.p, <path2>/proc.p, <path3>/proc.p)
  - If there is only one reference, we instantiate one SinglePathLookup. Inside the singlePathLookup, we store a Set<String> with all possible paths that can be resolved to the full-path in the end, according to the PRO-PATH. That is, if the full-path is a/b/c/proc.p and the PROPATH is .:a/b, then the set will have the following normalized entries: a/b/c/proc.p and c/proc.p.
  - If there are multiple references, we instantiate one MultiPathLookup: this stores a list of SinglePathLookup which should be checked sequentially.
- This map may be quite huge (number of files * size of PROPATH), but static.

In both scenarios

- If the legacy code uses <some-path>/target.p, the look-up in the map is done simply with target.p.
- For this target.p, we explore the map. If there is no entry, we return null.

There is one single target.p across the application

- In the look-up map, we would have target.p mapped to a SinglePathLookup implementation (highlander case)
- At this stage, we should only check if the <some-path> is compatible with the location of target.p. This is basically done already by SinglePathLookup pre-processing of PROPATH.
- For that matter, we need only to check if the normalized <some-path>/target.p is inside the SinglePathLookup set.
- The total effort to be done here: one map look-up + one set look-up.

There are more target.p across the application

- In the look-up map, we would have target.p mapped to a MultiPathLookup implementation (multi-path case)
- This will only iterate all entries of MultiPathLookup and check if either
  - All results are null, so the return is null.
  - One result is not null, so the return is known.
  - More than one result is not null, which means there is a conflict.
- The total effort to be done here: one map look-up + several set look-up (depending on how many procedures share the same name)

Issues

1. The "huge" map we are storing initially may be invalidated by a PROPATH change. I expect that such changes are not that often, so we may be able to rebuild the map fast enough. In fact, we don't need to rebuild the map, but the sets inside SinglePathLookup. These can be initialized lazily anyway (on first access / after PROPATH change) and lazily invalidated (on PROPATH change). This means we need to store a PROPATH version to be able to identify if the SinglePathLookup is stale or not. This may be very similar to how we implemented DmoVersioning, but without synchronization (?).
2. AFAIK, the PROPATH is per-session (please correct me if I am wrong here). So different sessions may need different SinglePathLookup resolutions. I think this can be resolved using ContextLocal pattern we have: basically, the SinglePathLookup will store the full-path of the target file (because this can't change at run-time). Any local context will build (lazily) its own Set based on the PROPATH. I wonder if we can actually merge some of these contexts, as in applications that don't have run-time PROPATH changes, there is a lot of redundant data being computed.

- I am aware that the current implementation would be refactored completely then - or renamed into something like convertNameDynamically. However, we still need some augmentation of SinglePathLookup to consider aliases and .r suffixes. I didn't yet analyzed this part.

Greg, I hope this is what you've meant in #6649-32. I provided some examples and flows just to be sure me and Danut understood the approach correctly. If yes, I can't think where the caching may actually fit. Maybe MultiPathLookup can implement a cache to reduce the traversal of all SinglePathLookup, but considering that there may be only some (< 5) such SinglePathLookup, the cache is too much of an overhead. We can discuss after we retrieve some statistics.

#35 - 09/20/2023 05:14 PM - Greg Shah

I think we are largely on the same page.

The one place I differ is that I was not thinking we would store the propath variants in the *PathLookup helpers. I was thinking we could handle this by "subtracting" any propath match from the input source (the target name). That would greatly reduce the variants that have to be stored AND it would avoid invalidation of the (now less) "huge" map when the PROPATH changes.

I would think that this way, we could use a single map for all sessions since it would be more directly representing the name map data.

You are right that the PROPATH is per-session, though it does follow this pattern:

- There will generally only be one or a small number of possible propaths.
- Assignment is rare, usually at the beginning of a session.

If we really need to have the propath variants mapped in our lookup helpers, then we would want to at least share the lookup data across all sessions which have the same propath. There is no reason to duplicate all the mappings for every session.

But I'm really wondering if we can get rid of the propath early, at the target input level.

> If yes, I can't think where the caching may actually fit.

You are probably right. Let's focus on this end-first lookup idea.

#36 - 09/21/2023 05:06 AM - Dănuț Filimon

I had to discard my initial implementation since it was going in the wrong direction and started again from scratch based on #6649-32 and #6649-34. There's still work to do and I made a list with what I've achieved and what's left to clarify/test/implement.

- SinglePathLookup and MultiPathLookup are two classes that will extend PathLookup;
- In SourceNameMapper.initMappingData() we populate a Map<String, PathLookup>. The nodes are already iterated in the method and the necessary data is already there. The mapping will be done using the simple program name (e.g. the simple name of **folder1/folder2/file.p** is **file.p**) so that paths like **folder1/file.p** and **folder2/file.p** will create a MultiPathLookup instance.
  - Each node will create a SinglePathLookup instance and the possiblePaths will contain the original filename and all the names without the propath appended;
  - If a SinglePathLookup instance was already created for one filename, it will create a MultiPathLookup instance and add the existent one together with previously created instance to the List<SinglePathLookup> of new object;
  - Further SinglePathLookup instances will be added for the same filename;
- The lookup will be done as mentioned by Alexandru, using the simple name for the map and searching for any path in the Set/List available. One thing to note is that the original filename will always be part of the set because an ExternalProgram is created for it for each node;

Some of the issues that I am addressing right now:

1. files ending in .r or no extension will not be found if the simple name is startproj.p;
2. p2jMap and p2jMap_ci insert the external program pname and the pname which are the same (?) and one map is searched based on the WorkArea.caseSens. The same can be said for the files without extensions since additional maps are created for them;
3. A conflict in MultiPathLookup doesn't necessarily mean that we don't have a match and the lookup should be done for each SinglePathLookup list, returning the best match at the end (or return immediately is the full path is the same as the one of the program that is being searched)
4. PROPATH change, lazy initialization and invalidation;

5. What should happen to the fallback processing case? This happens when there is no mapping and the program name references a valid Java class.
6. Aliases;
7. Second issue mentioned in #6649-34;

Greg Shah wrote:

> The one place I differ is that I was not thinking we would store the propath variants in the *PathLookup helpers. I was thinking we could handle this by "subtracting" any propath match from the input source (the target name). That would greatly reduce the variants that have to be stored AND it would avoid invalidation of the (now less) "huge" map when the PROPATH changes.

I've been thinking of this and I can provide an example from a partially tested customer application. After the SourceNameMapper.initMappingData() call, the map contained at least 12k entries (about 100 less that p2jMap) so we can say that there are lot less MultiPathLookup instances than we thought of. I also checked the number of possiblePaths from SinglePathLookup and the size of the list is 2-3, sometimes 4. In this case, the size of the final map would be considerable if you think that files without/with other extensions can exist.

Alexandru Lungu wrote:

> However, we still need some augmentation of SinglePathLookup to consider aliases and .r suffixes. I didn't yet analyzed this part.

In the matter of using aliases, I also don't have any idea on how it works. From what I see, aliases can be defined in the directory and are being "resolved" and then searched in the existent maps. There should be no problem creating a test case for this.

I am currently working and debugging using a customer application and can provide additional details if necessary.

#37 - 09/22/2023 06:33 AM - Dănuț Filimon

1. files ending in .r or no extension will not be found if the simple name is startproj.p;

It's possible to simply use the filename without the extension. I've made several changes to see how this would work:

- A SingleLookupPath will return the full path of the program since it should be the only obvious answer but this is not always the case. For example, **file1.p** and **file1.w** will generate a MultiPathLookup instance, but **file1.p** and **file1.txt** will only generate a SinglePathLookup for **file1.p** since **file1.txt** is not found in name_map.xml. This can cause conflicts when searching for the file1.txt which should return null. In this case, the lookup in SinglePathLookup should be done using the legacyProgName before being processed into a filename without extension;
- The MultiPathLookup will make use of the lookup method of the SinglePathLookup instances and will return immediately after finding a match.

I've also found a problem with this idea where the possible paths from a SinglePathLookup can't be matched with the given filename because any path which had the propath removed and then added to the list was not equal to it. For example, start was initialized with the path a/b/c/start.p, propath is .:a/b, possible paths are [a/b/c/start.p, c/start.p], we then search for the start.p file and it is not found. The solution would be to add the unqualified name to the possible paths.

#38 - 09/25/2023 06:16 AM - Alexandru Lungu

> The one place I differ is that I was not thinking we would store the propath variants in the *PathLookup helpers. I was thinking we could handle this by "subtracting" any propath match from the input source (the target name). That would greatly reduce the variants that have to be stored AND it would avoid invalidation of the (now less) "huge" map when the PROPATH changes.

I see your point. I guess I over-exaggerated with the finalization of the idea. I was lead by the idea that we want to avoid that O(|PROPATH|) at look-up time, so I attempted to flatten the algorithm. But this way, we are facing a lot of complications with per-session mappings and memory consumption.

On the other side, I don't see the gain in your solution comparing to what we have already implemented. This is because I am not quite sure that "subtracting" is such an easy / fast operation. We can have a/b PROPATH and ../c/target.p look-up key. In this case, a/target.p is a hit. In fact, I don't know if we can actually "subtract". We can just concatenate and canonicalize - but this is the exact process we are trying to avoid, right? Or maybe you are thinking of short-circuiting this by a simple PROPATH + look-up-key search and hope there is no ".." kind of logic there?

#39 - 09/25/2023 06:25 AM - Alexandru Lungu

Dănuț Filimon wrote:

> I've also found a problem with this idea where the possible paths from a SinglePathLookup can't be matched with the given filename because any path which had the propath removed and then added to the list was not equal to it. For example, start was initialized with the path a/b/c/start.p, propath is .:a/b, possible paths are [a/b/c/start.p, c/start.p], we then search for the start.p file and it is not found. The solution would be to add the unqualified name to the possible paths.

It is normal not to be found right? I mean, if you are running start.p and the only start.p in the application is a/b/c/start.p and this is not matched by any propath, the program should throw an error. the unqualified name hits only if it is the root or its path is in the PROPATH. There is no "implicit" search path in 4GL.

> It's possible to simply use the filename without the extension. I've made several changes to see how this would work:

Regarding extensions, I am even aware of a current regression in FWD. AFAIK, if you do RUN target (without extension), but the target procedure **has** extension .w an exception in thrown in FWD, but it works in 4GL. My point is that the extension logic should be carefully retested. Hopefully I recall correctly.

You logic looks OK, but note that you should group multiple files in the same MultiPathLookup **iff** there exists a look-up key that can resolute to both files in different context (different PROPATH values). Therefore, I think target.p and target.p should be part of the same MultiPathLookup, as the target key can resolute to both of them - please better test this scenario. Also consider target.p and target.r kind of scenarios. Or even files without extensions (target).

#40 - 09/25/2023 09:51 AM - Dănuț Filimon

Alexandru Lungu wrote:

> It is normal not to be found right? I mean, if you are running start.p and the only start.p in the application is a/b/c/start.p and this is not matched by any propath, the program should throw an error. the unqualified name hits only if it is the root or its path is in the PROPATH. There is no "implicit" search path in 4GL.

You are right! I've checked it again and this file is not found in the old implementation. I remember that the error specified the start.r which seemed weird, it was a quick fix on my end at the time so I did not pay too much attention to the cause. Thanks for the clarification.

> You logic looks OK, but note that you should group multiple files in the same MultiPathLookup iff there exists a look-up key that can resolute to both files in different context (different PROPATH values). Therefore, I think target.p and target.p should be part of the same MultiPathLookup, as the target key can resolute to both of them - please better test this scenario. Also consider target.p and target.r kind of scenarios. Or even files without extensions (target).

src1/**target.p** and **src2**/**target.p** will both have the **target** key, so it will create a MultiPathLookup. Currently the lookup methods for both SinglePathLookup and MultiPathLookup were either deleted or modified due to an improvement that I've made, this also required me to switch to unqualified names with extension as keys so the scenario **src1**/**target.p** and **src2**/**target.p** will still create a MultiPathLookup, but **src1**/**target.p** and **src2**/**target.w** will not. One important aspect of using keys without extension is that the number of MultiPathLookup instances did not increase that much, from the original implementation I could only notice around 60-100 new instances. In my opinion, the lookup became more complicated after taking into account files without extensions.

SinglePathLookup will not use the possiblePaths set anymore, the lookup will be done by PathLookup which will use the propath and caseSens from the WorkArea. This will reduce the memory usage and possible issues regarding propath changes. At the same time, caseSens value will be used to process the searched name and the PathLookup program name, returning the original program name if there is a match. This means that we can ignore caseSens and only search p2jMap.

4GL supports the following file types: .p, .r, .w, .cls, .i. I've also found .pf (parameter file), .cfg (configuration file), .db and .wrx (container for ActiveX, generated by compiling in AppBuilder). We can focus on the first 4 since .i is replaced directly in FWD and not searched. We can try to add the extension one by one and then search for any lookup instances. If multiple matches are found, we can fallback to the old implementation (this should also happen when there is a conflict in MultiPathLookup).

#41 - 09/25/2023 04:28 PM - Greg Shah

During conversion, we have a copy of the file-system of the 4GL project. That project might have some arbitrary path prefix that would not exist in the runtime OE system. We call that the "basepath". We then remove this basepath from the legacy program names making it a "relative name" that may include some pathing that corresponds to relative PROPATH entries. I mention this for future readers. The point is that the name map data (which we use to populate the SourceNameMapper can contain entries that are prefixed by relative propath entries.

The "subtracting" of the propath would have to occur on these relative names stored in the SourceNameMapper. By removing any such paths from those entries and from any passed in "search spec", I was hoping we could avoid the looping check by which we prepend each of the the propath entries to the search spec and then look for a match.

> We can have a/b PROPATH and ../c/target.p look-up key. In this case, a/target.p is a hit.

Can you help me understand what you mean here? By look-up key, do you mean the relative path read from the pname of the name map data? Or do you mean the "search spec" (the target name in the RUN statement)?

> In fact, I don't know if we can actually "subtract". We can just concatenate and canonicalize - but this is the exact process we are trying to avoid, right?

Yes

> Or maybe you are thinking of short-circuiting this by a simple PROPATH + look-up-key search and hope there is no ".." kind of logic there?

No, you're right that we need to handle the canonicalization. We definitely don't want to break the logic.

I think the primary problem with my idea is that by removing the propath from the search spec, we lose some knowledge because a more specific match is now able to be matched to more generic spec that might not have been possible in the original system.

#42 - 09/25/2023 04:32 PM - Greg Shah

> 4GL supports the following file types: .p, .r, .w, .cls, .i. I've also found .pf (parameter file), .cfg (configuration file), .db and .wrx (container for

> ActiveX, generated by compiling in AppBuilder). We can focus on the first 4 since .i is replaced directly in FWD and not searched. We can try to add the extension one by one and then search for any lookup instances. If multiple matches are found, we can fallback to the old implementation (this should also happen when there is a conflict in MultiPathLookup).

Actually, the 4GL allows any file name to be used. It does indeed have special processing for the .r case and the "no extension" case, but it is perfectly possible to RUN some-program.f., RUN some-program.frm. or even RUN some-program.danut.. We need to ensure we don't break that use case. It is rarely used but we have seen some applications that use program names other than .p and .w. We've even seen some RUN some-include.i. which works as long as the .i is in fact a syntactically correct 4GL program.

#43 - 09/26/2023 05:36 AM - Alexandru Lungu

I think we can agree that the extension topic is fully understood now.

> Can you help me understand what you mean here? By look-up key, do you mean the relative path read from the pname of the name map data? Or do you mean the "search spec" (the target name in the RUN statement)?

I mean that we have a search spec like ../c/target.p, the name-map entry is a/c/target.p and the PROPATH entry is a/b. In this case, the algorithm should succeed in finding the the file. My concern is how will these be combined, as "subtracting" the PROPATH entry from the name-map entry may not perfectly match the "search path" - so the "reversed searched" process is bad.

Now that I think of it, even my idea is broken (of keeping any possible precomputed "search spec" in a set), because there is an infinite number of possible search specs that may refer to a file (even with a single PROPATH entry).

> I think the primary problem with my idea is that by removing the propath from the search spec, we lose some knowledge because a more specific match is now able to be matched to more generic spec that might not have been possible in the original system.

The PROPATH may be found is in the file name (from name-map.xml), not in the search spec. As of my understanding, the search spec is the phrase used in the RUN statement (e.g. search spec is a.p as the statement is RUN a.p, even if the target file is x/y/a.p, where x/y is in the PROPATH).

Still thinking of how the solution we are talking about is going to be better than the current implementation.

#44 - 09/26/2023 08:31 AM - Dănuț Filimon

I retested #6649-30 using the new implementation and I got a total of **38119** calls to the new SourceNameMapper.convertNameDynamically method.

- The size of pathLookup map was **12583** with **12546** SinglePathLookup and **37** MultiPathLookup instances. In the MultiPathLookup there were a total of **88** references with an average of **2.37** SinglePathLookup instances per object;
- A total of **1581** different legacyProgName values were passed as an argument;
- From the **38119** calls to convertNameDynamically, no lookup was found in the map in **16291**, so convertName was used instead;
- From the **16291** calls to convertName, only **129** returned a value different from null, while **16162** calls returned null. The **129** calls searched for **.r** files, but I've seen just 6 cases were **.p** and **.w** files were searched. I am currently investigating this;
- The top three most used legacyProgName were called **4570**, **3056** and **1012** times, and took **38**, **41** and **20** milliseconds to complete. Note that only the first two match the same placement as in the previous test, while the program name that is now in third place was previously in the fourth.

#45 - 09/26/2023 08:38 AM - Greg Shah

> *The PROPATH may be found is in the file name (from name-map.xml)*, not in the search spec.

Yes, but a propath element (x/y in your example below) can also be present in the search spec. My point is that we have to treat the hard coded search spec (x/y/a.p) differently from the search spec a.p which we prefix with the propath elements before searching. The reason for this is that the prefixed versions have to be processed in the order of the propath elements and so the prefixed versions can match multiple versions of a.p if they exist in different "propathed" locations, while the non-prefixed version would only match a more specific instance of a.p. If we "subtract" the propath element from that search spec, then we would be treating it as the same thing as a.p and it could potentially match the wrong a.p. For example, if the propath was z/w:x/y, both z/w/a.p and x/y/a.p existed, then passing x/y/a.p should only match the second one while subtracting the x/y from the search spec would mean we would match z/w/a.p.

> As of my understanding, the search spec is the phrase used in the RUN statement (e.g. search spec is a.p as the statement is RUN a.p, even if the target file is x/y/a.p, where x/y is in the PROPATH).

Correct, we are using this terminology the same way.

#46 - 09/26/2023 09:24 AM - Dănuț Filimon

I've committed the implementation to ~~**6649a/rev.14741**~~ **6649b/rev.14741**. Please review and advise.

- propath can end or start with "/", this is not handled yet;
- path aliases still need to be tested, but I don't quite understand how to use them even after setting them up.

EDIT: mentioned for the wrong branch, the implementation was committed to **6649b**

#47 - 09/28/2023 04:06 AM - Dănuț Filimon

I've noticed something for a while, but I am only bringing this up now because it might be important. While doing the tests from #6649-30 and #6649-44, there were a lot of files with extensions that do not match any configured extension but are searched and always return null. In the #6649-44 test, these files were called **15218** times and wasted a **1.4 seconds**, the rest of the **22901** calls finish in **0.7 seconds**! I know that the legacyProgName can be a class method package.Start.execute, but this is the only case where I've seen it happen. What are your thought on this?

#48 - 09/28/2023 05:59 AM - Alexandru Lungu

Danut, can you share some of these extensions and the context in which they are required? I mean, what kind of files are searched for and what is the original intent? Are we talking about .wrx files maybe?
Please censor if there are any customer related names.

#49 - 09/28/2023 06:12 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, can you share some of these extensions and the context in which they are required? I mean, what kind of files are searched for and what is the original intent? Are we talking about .wrx files maybe?
> Please censor if there are any customer related names.

Of course, there are files with the .wrx extension, but are not called extensively (108 calls). I am mostly talking about .bmp files, but there are also files with .ini, .ico, .def, .xml, .csv extensions (low in number).

#50 - 09/28/2023 06:13 AM - Greg Shah

These are all coming from RUN statements?

#51 - 09/28/2023 06:16 AM - Constantin Asofiei

Lookup on the PROPATH can be done via the SEARCH function and IIRC even by runtime (i.e. when resolving images or ini files). So is not just RUN statement which is using SourceNameMapper.

#52 - 09/28/2023 06:21 AM - Dănuț Filimon

Greg Shah wrote:

> These are all coming from RUN statements?

No, I've looked up where a .bmp file is used and it can also be used in DEFINE BUTTON button IMAGE-UP FILE example.bmp .... Constantin is right.

#53 - 09/28/2023 06:27 AM - Greg Shah

This is exactly the kind of thing that the idea in #6649-17 would optimize. There will be no whatever.bmp in the map of base names and it can quickly be eliminated.

#54 - 09/28/2023 06:55 AM - Dănuț Filimon

But wouldn't it be faster to rule out such searches based on the file extension? Since the original lookup is done only on the files from the name_map file, it would be easier to say that the file was not found since the extension was not configured to be added.

I've also looked at the execution time of the top .bmp files. Currently, if the new convertNameDynamically fails, it will try to get the file using convertName like before. In this case, the time doubled or tripled when searching such files. We should take into account that files without extension or .r files will use the same old implementation. For example, we will try to verify if the file without an extension is valid and search it, but will not find anything because it's an .bmp. Isn't it an option to rule out such files beforehand?

#55 - 09/28/2023 07:06 AM - Greg Shah

If the more generic approach of #6649-17 is still too costly, then we can certainly put some logic in to check for the file extension and give a quick exit. Please note that we cannot hard code this extension list for the check. It is perfectly possible in OE to RUN something.bmp. if that file is actually a 4GL program. So we just need to ensure that we don't have any legacy program names that could match that extension.

#56 - 10/02/2023 08:17 AM - Dănuț Filimon

By removing unnecessary files from the lookup (files with an extension that don't match any legacy program name extension), the time spent searching a program was reduced by 50% from the original time. Prior to this change, I noticed **2x**/**3x** more time spent for this type of files compared to the original and increased the overall time spent in SourceNameMapper.convertNameDynamically.

I looked into how and why .bmp files are searched to possibly prevent these types of files from reaching SourceNameMapper and reducing the string manipulation on it's end. SourceNameMapper.lookupLegacyName is executed in FileSystemOps.searchPath while the files are actually resolved later by FileSystemOps.searchResourceJars. The lookupLegacyName call is made before obtaining the WorkArea and running searchResourceJars probably because it is less expensive. In conclusion, it is not be possible to remove such files from reaching SourceNameMapper because the converted code uses a direct call to searchPath with the intention of retrieving the mentioned files.

#57 - 10/02/2023 10:44 AM - Dănuț Filimon

- % Done changed from 0 to 50

Committed 6649b/rev.14742.

- Added r-code file searching;
- Modified PathLookup into an interface;
- Short-circuit files with extensions that don't match any legacy program extensions;

Please review.

#58 - 10/13/2023 11:18 AM - Alexandru Lungu

I skimmed the changes in 6649b and they are quite robust. Well done!
There is a lot of documentation there, so from this POV I think the changes are clear enough. There are some slight concerns regarding the use of :

instead of classical for loop and other minor topics, but I will list them on a second iteration of the code asap.

Maybe convertNameDynamically is not quite the most appropriate naming - what about fastConvertName?

Danut, is there something left to implement here? Do you have any performance overview for the changes? Anything left to test?

Also, **Greg**, can you provide a second review of the changes and confirm they are spot on with the requirements?

#59 - 10/16/2023 01:30 PM - Greg Shah

Hynek: Please review.

#60 - 10/17/2023 03:14 AM - Dănuț Filimon

Alexandru Lungu wrote:

> There is a lot of documentation there, so from this POV I think the changes are clear enough. There are some slight concerns regarding the use of : instead of classical for loop and other minor topics, but I will list them on a second iteration of the code asap.
>
> Maybe convertNameDynamically is not quite the most appropriate naming - what about fastConvertName?
>
> **Danut**, is there something left to implement here? Do you have any performance overview for the changes? Anything left to test?

Most of the testing was done before committing **6649b/rev.14742**. I remember that I still wanted to make a few documentation changes. At the same time, I didn't do any performance tests for this revision, but #6649-47 and #6649-56 mention the results of the change based on previous performance tests since this revision focused on short-circuiting unnecessary files.

Please note that when a r-code is passed, it is transformed into a p-code file without taking into account the caseSens value of the WorkArea. This needs to be changed too.

I can redo the test from #6649-44 if necessary, but focus more on the total time executed compared to previous test results.

#61 - 10/17/2023 01:21 PM - Hynek Cihlar

Code review 6649b revision 14742 and 14741.

Functionally the changes look OK. I'm throwing in some random ideas for performance.

convertNameDynamically:

- instead of

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

if (wa.caseSens == null)
```

```
{
    wa.caseSens = EnvironmentOps.getLegacyCaseSensitive();
}
```

it would be more efficient to store caseSens in the class instance as well as the other fields referenced in convertNameDynamically and avoid local.get() alltogether.

- I would consider using CaseInsensitiveHashMap and CaseInsensitiveHashSet instead of converting the legacy prog name to lower case in canonicalize.
- I wonder if replacing dblBackslashRegexp.matcher(legacyProgName).replaceAll(fileSep); with a manual char iteration with direct separator replacement would make much of a difference.
- Shouldn't the result of the conversion be cached? The cache would be invalidated when the input variables would change, like propath.

#62 - 10/19/2023 09:51 AM - Dănuț Filimon

Hynek Cihlar wrote:

> **convertNameDynamically:**
>
> - instead of
> [...]
> it would be more efficient to store caseSens in the class instance as well as the other fields referenced in convertNameDynamically and avoid local.get() alltogether.

If you take a look at the old convertName method implementation, it also used local.get() because it needed values such as propath, isWindows. In this case, those are needed too. I don't see how I can store the caseSens value in the class when the lookup should be done based on it's value that can be different in other contexts.
It seems that currently the convertName method calls getLegacyCaseSensitive() but it already has access to the WorkArea since it is passed as a parameter (this will be changed).

> - I would consider using CaseInsensitiveHashMap and CaseInsensitiveHashSet instead of converting the legacy prog name to lower case in canonicalize.

The string should not be lower cased at all before the path lookup because it might not find the file based on the simple program name. IMO, we can go ahead and not canonicalize the legacyProgName at all in the beginning since the lookup is done using just the simple program name and it should not be influenced by directory references at all, just the fileSep. The actual place where legacyProgName should be canonicalized is in the lookupPath method.

> - Shouldn't the result of the conversion be cached? The cache would be invalidated when the input variables would change, like propath.

We had a few discussions about the cache that resulted in it being more of an overhead than an improvement. There was a cache mentioned for MultiPathLookup but considering the low number of SinglePathLookup instances that it stored (found after testing a customer application) it was not talked much after.
From the same customer application, multiple files were called hundreds or thousands of times, the ones called hundreds of times have a total time of ~**6ms-12ms** and for thousands, ~**30ms-71ms**. These files were in low number so the cache will only benefit them, while the rest were **40** more in number and will not be searched too often.

#63 - 10/19/2023 11:38 AM - Hynek Cihlar

Dănuț Filimon wrote:

> Hynek Cihlar wrote:
> It seems that currently the convertName method calls getLegacyCaseSensitive() but it already has access to the WorkArea since it is passed as a parameter (this will be changed).

Right, this is probably not worth it. It would be a lot of work with uncertain result.

- I would consider using CaseInsensitiveHashMap and CaseInsensitiveHashSet instead of converting the legacy prog name to lower case in canonicalize.

> The string should not be lower cased at all before the path lookup because it might not find the file based on the simple program name. IMO, we can go ahead and not canonicalize the legacyProgName at all in the beginning since the lookup is done using just the simple program name

canonicalize beside the path also handles case sensitivity, which is needed on case insensitive file systems. So it seems to be still needed before the lookup.

- Shouldn't the result of the conversion be cached? The cache would be invalidated when the input variables would change, like propath.

> We had a few discussions about the cache that resulted in it being more of an overhead than an improvement. There was a cache mentioned for MultiPathLookup but considering the low number of SinglePathLookup instances that it stored (found after testing a customer application) it was not talked much after.
> From the same customer application, multiple files were called hundreds or thousands of times, the ones called hundreds of times have a total time of ~**6ms-12ms** and for thousands, ~**30ms-71ms**. These files were in low number so the cache will only benefit them, while the rest were **40** more in number and will not be searched too often.

I'm not that familiar with the anticipated test cases, but wouldn't the cache still be beneficial if it was shared among sessions?

#64 - 10/20/2023 05:32 AM - Dănuț Filimon

Hynek Cihlar wrote:

> I'm not that familiar with the anticipated test cases, but wouldn't the cache still be beneficial if it was shared among sessions?

If this cache is shared among sessions then it will cause some problems. Here's an example:

1. two different clients with different propaths;
2. the first one searches for a/b/file1.p and the second one searches for a/c/file1.p;
3. the first one uses the propath a/b and the second one uses the propath a/c;
4. both will introduce a new value into the cache for a/b/file1.p and a/c/file1.p;
5. when the first client searches for a/c/file1.p, it will retrieve the value from the cache because it was previously searched but it should not expect any file to be found.

#65 - 10/20/2023 06:04 AM - Hynek Cihlar

Another idea I'm not sure you already considered is precalculating the denormalized Progress names passed in to the convert method. I.e. all the constant expressions passed in to all the relevant statements (RUN, IMAGE, etc.) would be resolved and saved during conversion. On runtime during initialization, the saved list would be read and all the entries converted. During the actual Java class name resolution you would only read a single map in the ideal case, in the worst case you would have to convert the Progress name as it is done today. The precalculated map would be stored per session and would have to be recalculated on input changes (like propath).

#66 - 10/20/2023 06:06 AM - Hynek Cihlar

Hynek Cihlar wrote:

> The precalculated map would be stored per session and would have to be recalculated on input changes (like propath).

To optimize footprint you could store just the hashes in every session.

#67 - 10/20/2023 07:28 AM - Dănuț Filimon

Hynek Cihlar wrote:

> Another idea I'm not sure you already considered is precalculating the denormalized Progress names passed in to the convert method. I.e. all the constant expressions passed in to all the relevant statements (RUN, IMAGE, etc.) would be resolved and saved during conversion. On runtime during initialization, the saved list would be read and all the entries converted. During the actual Java class name resolution you would only read a single map in the ideal case, in the worst case you would have to convert the Progress name as it is done today. The precalculated map would be stored per session and would have to be recalculated on input changes (like propath).

This is an interesting idea. If we take all the files, including images, we should take into consideration that not all files need conversion. At the end, the number of converted entries would match those found in the name_map.xml if I am right. The bright side would be that the propath will be less used, but I don't think that storing a huge map for each session is ok, especially when that map will need to be built again if some parameters are changed.

If I were to talk of a per session cache it would be storing the values of convertNameDynamically/convertName which will generally be a lot less entries (even if 97% of the map entries are searched less than 100 times, but this can be said about the idea above too) and add up over time. Changing the propath will simply invalidate this cache and it will start to populate it again.

Please let me know what you think and if I understood you correctly.

#68 - 10/20/2023 07:30 AM - Alexandru Lungu

For the conversion side, do you mean preprocessing the path when converting to extract the base name (a.p from x/y/a.p), do toLowerCase eventually and other preprocessing stuff (extension related)? This way we simply input the base name of the file into our PathLookup routines without needing run-time processing. Well, I guess we can extend RUN_CALL_SITE_* for that matter, right? Use a second parameter like: InvokeConfig().setTarget("x/y/a.p", "a.p").

For run-time, I wonder if this is not again the same case for RUN_CALL_SITE_*. I mean, maybe we can do the path computation when we actually set the target in RUN_CALL_SITE_*. As these are static, I guess it is equivalent to your *On runtime during initialization, the saved list would be read and all the entries converted.*.

As for the per-session stuff, maybe we can store a hash inside the RUN_CALL_SITE_* for this exact matter. I am still thinking if we need to keep a map between propath hashes and their resolution **or** simply store the "last" resolution. In the first case, I am still thinking of the eviction policy (using WorkArea maybe?).

Please review. This can be started off by changing conversion, adding a second parameter to setTarget and simply pick up that name instead of processing it in convertName. For the run-time, I think the caching needs better investigation.

> Changing the propath will simply invalidate this cache and it will start to populate it again.

I agree with Hynek: we can hash the propath and share the same cache through different sessions.

#69 - 10/20/2023 08:25 AM - Dănuț Filimon

I've **committed 6649b**/**rev.14743** with the following changes:

- renamed convertNameDynamically to fastConvertName;
- check the file extension before getting the local context, it should slightly improve performance because the extension check is not dependent on the context;
- kept the canonicalize call, but called it right before the path lookup;
- modified r-code file check to also use caseSens;
- removed unnecessary getLegacyCaseSensitive call in convertName;
- used classical for loops.

I want to test if there is any performance improvement for #7767 with these changes.

#70 - 10/20/2023 08:31 AM - Hynek Cihlar

Alexandru Lungu wrote:

> For the conversion side, do you mean preprocessing the path when converting to extract the base name (a.p from x/y/a.p), do toLowerCase eventually and other preprocessing stuff (extension related)?

No I only meant to gather all the Progress file names during conversion. Having all the names that can be passed to SourceNameMapper means we can pre-calculate them to Java names at more convenient times, like during initialization.

Unless I'm missing something using RUN_CALL_SITE_* for the caching/precalculation isn't optimal. First these would only cover RUN statements and second the pre-calculation would be done when the converted classes were referenced. This would be too late.

#71 - 10/30/2023 06:30 AM - Alexandru Lungu

- Priority changed from Normal to High

Hynek Cihlar wrote:

> Unless I'm missing something using RUN_CALL_SITE_* for the caching/precalculation isn't optimal. First these would only cover RUN statements and second the pre-calculation would be done when the converted classes were referenced. This would be too late.

I see your point. Got stuck in the RUN statements, but other flow can use SourceNameMapper as well.

I will like to have this #6649 ongoing as it is really promising in terms of performance. I moved this to High.

What is implemented currently:

- A mean to search sources faster by "linking" the search spec base file name with a program name from name_map.xml. This happens when the program has an unique name. The only complexity here comes from PROPATH subtracting to check if the mapper should fail or not.
- A mean to aggregate the above in case multiple files are named the same.
- If none of the technique above succeed, the old-fashioned PROPATH traversal, canonicalize and search is done.
- Of course, everything regarding case sensitivity and extensions is handled here.

What is left to implement:

- Add a cache to the process above. The cache should be done "per-propath", so we store something like LRUCache<Long, SourceNameMapperCache>. The key is the hash of the propath and SourceNameMapperCache has the actual LRUCache and PROPATH. We need to double-check with the PROPATH to ensure that the hash succeeded. If not, I guess it is bad luck :/ Anyway, the LRUCache should store the map between search-spec and target procedure. Its size is based on CACHE_SIZE (see below).
- Change the run-time to consult the cache (eventually initialized at server start-up) or fallback to fastConvertName (or even convertName) and populate the cache. **Test and profile**
- Extract every bit of information available regarding paths from conversion and do a massive path resolution at server start-up. This is basically a "cache initialization" for the default PROPATH. At this point, we can set CACHE_SIZE as the number of information extracted at conversion time. If we have propath changes, we shouldn't invalidate. Just make use of the Map above:
  - **Hynek**, in case the PROPATH changes, should we have SourceNameMapperCache empty or populated with everything we found at conversion? It may take a while to instantiate a SourceNameMapperCache. Also, please let me know if I missed something from the last comments.
  - I think the best approach is to have a 2-level cache here. If you were thinking of something else, please advice.
  - **Test and profile** how much will this final solution impact performance?

Danut, please provide an estimate of this approach. Are you able to have them in a "merge-able" state by the end of this week? I would also like to have them tested with everything we have, so this will also take 1/2 days.

#72 - 10/30/2023 08:34 AM - Dănuț Filimon

A new path_map.xml file should be created, similar to name_map.xml, with nodes that specify the paths for each RUN statement (1:1, 1 RUN statement, 1 node). I will need to investigate how name_map.xml is created in the first place and how nodes are added to the file, the file should be created after all the files were checked for RUN statements to avoid duplicate paths. Any insight on how is this done is welcome. IMO, this should be the most time consuming part of the implementation, I expect the cache to be an easy implementation but it should still have some level of synchronization (probably using a ReentrantReadWriteLock).

I should be able to get it done until the end of the week (testing included).

#73 - 10/30/2023 08:41 AM - Alexandru Lungu

Danut:

- Not only RUN statements should be analyzed. Also anything that is going to be a path (like Hynek stated, IMAGE path).
- There is no actual "map" happening in this case. It is just a big list of paths you could find in the sources. Maybe use something like path_list.txt. No need to be an xml; just one entry per line is enough:

```
a/b/x.p
a/c.p
y.p
```

- Synchronization is required indeed. Hopefully is not that time consuming. However, doing a read also changes the LRU state (pushing the queried entry at the beginning of the list). I am quite circumspect on how we can manage to have this synchronized. Maybe we should stick to a Map inside SourceNameMapperCache instead of a LRUCache in the end :/

#74 - 10/30/2023 10:24 AM - Dănuț Filimon

A few cases that I found while running a Hotel_GUI conversion and should be mentioned:
- RUN VALUE(possiblePath) PERSISTENT SET hProc.
- RUN aProcedure("folder/file.p":U).
- RUN aProcedure(INPUT TARGET-PROCEDURE).
- RUN aProcedure. (it can be avoided if we check that aProcedure is INT_PROC type)

The case where VALUE() is used seems to be a little complicated and the following 2 should also be taken into consideration when searching for file paths in RUN statements.

#75 - 10/30/2023 11:48 AM - Alexandru Lungu

The first one is an expression, so a dynamic way to run a procedure. It can be disregarded for now. The following I think refer to internal procedures, not external. Maybe you should target only targets that have extensions (?) to be able to make a difference here. I don't think we should burden the path list with potential internal procedures. The ones that end up with .p or .w (or other valid extensions) should be extracted.

#76 - 10/31/2023 04:34 AM - Hynek Cihlar

Alexandru Lungu wrote:

- **Hynek**, in case the PROPATH changes, should we have SourceNameMapperCache empty or populated with everything we found at conversion? It may take a while to instantiate a SourceNameMapperCache. Also, please let me know if I missed something from the last comments.

Actually I didn't have much of the specific implementation ideas in mind. The two-level approach should work well I think. I especially like the idea of caching per specific propath configuration.

Regarding CACHE_SIZE, why would you want to limit the number of entries?

Another point is not to store the actual entries in memory but rather work with some kind of table-based approach. The primary entries of the conversion-time parsed file names would be stored in an indexed list (or an array) and all the references in the cache objects would be done as simple indexes to this list.

The cache initialization (during server start up or when new propath configuration is introduced) can be done asynchronously on a separate thread not to introduce wait-times. Until the cache is initialized (for the specific propath) SourceNameMapper can fall-back to fastConvertName directly.

#77 - 10/31/2023 04:41 AM - Hynek Cihlar

Perhaps also some at least basic runtime metrics should be gathered with our JMX infrastructure.

#78 - 10/31/2023 08:14 AM - Dănuț Filimon

I've analyzed how name_map.xml is created and took in consideration writing to a .txt file which should be simpler (I even found an example in the rules). The changes I made were added to annotations.xml right before KW_RUN clauses are rewritten/generated (also considered early_annotations.xml).

I am trying to write to a .txt file in the following way:

```
<post-rules>
  <rule>deleteFile(outputFilename, false)</rule> <!-- outputFilename is 'path_map.txt' -->
  <rule>fid = openStream(outputFilename, true)</rule> <!-- fid is a long value -->
  <rule>iter = paths.iterator()</rule>
  <while>iter.hasNext()
    <action>fname = #(java.lang.String) iter.next()</action>
    <action>fprintf(fid, '%s\n', fname)</action>
  </while>
  <rule>closeStream(fid)</rule>
</post-rules>
```

I found this code in rpt_file_list.xml but it fails when deleteFile is called (using FileOperationsWorker results in the same output) and after removing that call it fails at openStream.

Is there any better way to write a .txt file or should I try to write a .xml file and do something similar to name_map.xml? There is also the possibility that rpt_file_list.xml is not actually being used, but deleteFile appears in several other rules.

#79 - 10/31/2023 08:34 AM - Alexandru Lungu

Hynek Cihlar wrote:

> Regarding CACHE_SIZE, why would you want to limit the number of entries?

This should be the size of the cache; the cache can be extended at run-time with new entries. As there may be lots of run-time possible values, I thought of having it limited to avoid leaks. However, synchronization may represent a problem (if two threads are updating the cache) - so we can stick with a map without limit, as you said.

> Another point is not to store the actual entries in memory but rather work with some kind of table-based approach. The primary entries of the conversion-time parsed file names would be stored in an indexed list (or an array) and all the references in the cache objects would be done as simple indexes to this list.

The strings will be interned as they will appear in the Java code after conversion, right? They will reside in memory anyway, so when doing the checks in the map, the strings will be equal. Either way, we can check this hypothesis.

> The cache initialization (during server start up or when new propath configuration is introduced) can be done asynchronously on a separate thread not to introduce wait-times. Until the cache is initialized (for the specific propath) SourceNameMapper can fall-back to fastConvertName directly.

Cool idea, especially for times when PROPATH configuration changes.

#80 - 10/31/2023 08:45 AM - Greg Shah

> I've analyzed how name_map.xml is created and took in consideration writing to a .txt file which should be simpler (I even found an example in the rules). The changes I made were added to annotations.xml right before KW_RUN clauses are rewritten/generated (also considered early_annotations.xml).

Consider that if you emit a separate/new artifact, that must now be documented and managed forever as an extra set of steps/scripting. Why not just augment the existing name_map.xml to add more information?

> I found this code in rpt_file_list.xml but it fails when deleteFile is called (using FileOperationsWorker results in the same output) and after

> removing that call it fails at openStream.

I'm not sure based on what you've posted. What is the failure?

#81 - 10/31/2023 09:02 AM - Dănuț Filimon

Greg Shah wrote:

> Consider that if you emit a separate/new artifact, that must now be documented and managed forever as an extra set of steps/scripting. Why not just augment the existing name_map.xml to add more information?

This is because the files from RUN statements do not specify the full path like the name_map.xml, we gather them during conversion and then pre-calculate the PathLookup at initialization. This also means that we might have less propath checking when doing the lookup.

> I'm not sure based on what you've posted. What is the failure?

Both deleteFile and openStream result in the same error.

```
     [java]     at com.goldencode.p2j.convert.ConversionDriver.main(ConversionDriver.java:1284)
     [java] Caused by: com.goldencode.expr.ExpressionException: Expression error [fid = openStream(outputFilen
ame, true)]
     [java]     at com.goldencode.expr.Expression.getCompiledInstance(Expression.java:695)
     [java]     at com.goldencode.expr.Expression.execute(Expression.java:387)
     [java]     at com.goldencode.p2j.pattern.Rule.apply(Rule.java:500)
     [java]     at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:597)
     [java]     at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:98)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.apply(PatternEngine.java:1710)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1577)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1510)
     [java]     at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1062)
     [java]     ... 4 more
     [java] Caused by: com.goldencode.expr.CompilerException: Error parsing expression
     [java]     at com.goldencode.expr.Compiler.process(Compiler.java:378)
     [java]     at com.goldencode.expr.Compiler.compile(Compiler.java:298)
     [java]     at com.goldencode.expr.Expression.getCompiledInstance(Expression.java:687)
     [java]     ... 12 more
     [java] Caused by: com.goldencode.expr.UnresolvedSymbolException: No function resolved for instance (null)
 and variable (setter == false, name == (outputFilename))
     [java]     at com.goldencode.expr.ExpressionParser.method(ExpressionParser.java:1900)
     [java]     at com.goldencode.expr.ExpressionParser.primary_expr(ExpressionParser.java:847)
     [java]     at com.goldencode.expr.ExpressionParser.un_expr(ExpressionParser.java:1629)
     [java]     at com.goldencode.expr.ExpressionParser.prod_expr(ExpressionParser.java:1481)
```

#82 - 10/31/2023 09:43 AM - Alexandru Lungu

> Consider that if you emit a separate/new artifact, that must now be documented and managed forever as an extra set of steps/scripting. Why not just augment the existing name_map.xml to add more information?

I think the right answer to this is that name_map.xml is delegated to store a mapping between legacy procedures (and their parameters) and Java procedures. However, the list we are talking about it not related to "procedures" and "parameters" in any way. It is just a massive list with "search specs" we find through-out the application. This list is used to resolve the static search specs right from the server start-up (or at propath changes, asynch in a different thread).

We basically iterate the whole name_map and instantiate out look-up routines (single or multi). **Afterwards** we iterate the whole list and apply fastConvertName over each search spec we found.

#83 - 10/31/2023 10:38 AM - Greg Shah

> Both deleteFile and openStream result in the same error.
> [...]

The outputFilename variable needs to be defined before it is used.

> This is because the files from RUN statements do not specify the full path like the name_map.xml, we gather them during conversion and then pre-calculate the PathLookup at initialization. This also means that we might have less propath checking when doing the lookup.

Understood. Even if the data is slightly different, I would rather keep it together with the name mapping data because it is used in the same place and the costs of managing another artifact over the lifetime of FWD (decades?) and across the number of customer projects will be substantial.

So: please avoid the extra artifact.

Alexandru: If everythign can be calculated from existing name map data, great! If not, it is OK to add data to that file.

#84 - 10/31/2023 10:57 AM - Dănuț Filimon

Greg Shah wrote:

> Both deleteFile and openStream result in the same error.
> [...]

> The outputFilename variable needs to be defined before it is used.

The variable is defined and has the value path_map.txt assigned. Once I looked over closely I was defining outputFileName and using outputFilename. Thank you.

#85 - 10/31/2023 11:08 AM - Alexandru Lungu

Rebased 6649b to latest trunk. It is now at rev. 14801.

#86 - 11/01/2023 08:06 AM - Dănuț Filimon

Here's an update of the implementation progress:

1. Managed to extract filenames from RUN statements and write them to a file (path_map.txt) on separate lines;
2. path_map.txt is copied into the same folder as name_map.xml;
3. Created SourceNameMapper.initPathData method which is called from initMappingData. This method is called right after all name_map.xml entries are mapped to their Java correspondent and it reads all lines from the path_map.txt to create a Map<String, String[]> (the path and the result from fastConvertName) which is stored into a LRUCache<Long, SourceNameMapperCache>. At this point I want to ask why does the propath need to be a member of SourceNameMapperCache? I only use the propath to calculate a hash code and store the SourceNameMapperCache in the LRUCache. Currently I have no other use for the propath in the cache class, if there is no reason to include the propath we can just use the Map directly.
4. Added another fastConvertName that takes the WorkArea as a parameter. This is because we read multiple paths from the file and local.get() will be called each time. Using this method, local.get() is called only once and is available for each time. The default fastConvertName (with one parameter) will also call the new method with a null for the WorkArea because it can short-circuit the search based on the file extension (this is subject to change, since I am thinking on adding these values to the cache);
5. Added checkSourceCache method which uses a ReentrantReadWriteLock. It will lock when reading the cache and if there is no cache for the propath, create a new one.

I still need to:

- decide on the points (3, 4) mentioned above;
- add the values obtained from executing fastConvertName when the cache lookup fails;
- remove unnecessary processing from the implementation and clean up the code;
- reconvert an application and test;
- fix possible bugs in the code flow;
- add documentation.

#87 - 11/01/2023 08:25 AM - Greg Shah

As mentioned in #6649-83, please do not create the extra path_map.txt. Add the data into name_map.xml.

#88 - 11/01/2023 09:05 AM - Hynek Cihlar

Dănuț Filimon wrote:

> At this point I want to ask why does the propath need to be a member of SourceNameMapperCache?

This is to avoid potential conflicts of the hashed path entries. When you get a match you must compare the propath in the found cache object to make sure you got the right cache object.

#89 - 11/02/2023 09:16 AM - Dănuț Filimon

Committed 6649b/rev.14802.

- Managed to write all the data to name_map.xml, avoided creating path_map.txt. I modified the rules to create an additional node called path-mapping, which is stored and restored the same way as the other data from name_map.xml;
- Currently the cache is created only if there are search-specs found during conversion and only creates a new SourceNameMapperCache instance when a different propath is used that doesn't have a cache associated;

Still need to:

- Create the cache in a separate thread;
- Convert an application and test (I tried with Hotel_GUI but the application doesn't go beyond the login screen - probably due to being configured in the wrong way);
- Documentation (param, return);

Problems:

- Duplicate path-mappings nodes in name_map.xml;
- Hard-coded cache size in SourceNameMapper and node names in NameMappingWorker;

#90 - 11/03/2023 05:43 AM - Dănuț Filimon

I've tried to use a thread to create a cache but I get the following error:

```
java.lang.NullPointerException
    at com.goldencode.p2j.security.SecurityManager.isServerAccount(SecurityManager.java:4211)
    at com.goldencode.p2j.util.TransactionManager._isHeadless(TransactionManager.java:3131)
    at com.goldencode.p2j.util.EnvironmentOps.getClientOpSysName(EnvironmentOps.java:2890)
    at com.goldencode.p2j.util.EnvironmentOps.getOperatingSystem(EnvironmentOps.java:483)
    at com.goldencode.p2j.util.EnvironmentOps.getLegacyPlatform(EnvironmentOps.java:2783)
    at com.goldencode.p2j.util.EnvironmentOps.isLegacyPlatformWindows(EnvironmentOps.java:536)
```

```
    at com.goldencode.p2j.util.SourceNameMapper$1.initialValue(SourceNameMapper.java:327)
    at com.goldencode.p2j.util.SourceNameMapper$1.initialValue(SourceNameMapper.java:321)
    at com.goldencode.p2j.security.ContextLocal.getImpl(ContextLocal.java:541)
    at com.goldencode.p2j.security.ContextLocal.get(ContextLocal.java:453)
    at com.goldencode.p2j.util.SourceNameMapper.fastConvertName(SourceNameMapper.java:1895)
    at com.goldencode.p2j.util.SourceNameMapper.access$1900(SourceNameMapper.java:293)
    at com.goldencode.p2j.util.SourceNameMapper$2.run(SourceNameMapper.java:3056)
    at java.lang.Thread.run(Thread.java:750)
```

OTOH, I considered using the CacheManager to create the sourceCache since SourceNameMapper is initialized later than it by the StandardServer but I found that NameMappingWorker.classNameExists can also trigger the initialization method and is used in common-progress.rule.

I am currently addressing the duplicate path-mapping node problem but no luck so far, I've tried to see how mappings are stored and restored but I don't see how can these be restored in a duplicate state. I did not mention this, but the problem is currently handled when the cache is initialized by storing path mappings in a Set, so no duplicates will be looked up. Any advice is welcome here.

#91 - 11/03/2023 07:09 AM - Hynek Cihlar

Dănuț Filimon wrote:

> I've tried to use a thread to create a cache but I get the following error:

You won't be able to access the security context from the worker thread, so you will have to prepare all the required context beforehand for fastConvertName or convertName.

#92 - 11/03/2023 09:13 AM - Dănuț Filimon

Hynek Cihlar wrote:

> You won't be able to access the security context from the worker thread, so you will have to prepare all the required context beforehand for fastConvertName or convertName.

Thank you for the information, I've been digging up for a while and I think that a thread is not actually a good idea. I've found several nested methods called by fastConvertName and convertName that end up requiring access to the context (some of which I did not notice even after looking over the code): resolvePathAliases, getExternalProgram, EnvironmentOps.getSearchPath, setSearchPath, EnvironmentOps.getLegacyCaseSensitive. Rewriting these methods to accept the parameter needed from the context doesn't seem at all nice.

I was doing something like:

```
public static boolean method1(String pname) { return method1(null, pname); }
public static boolean method1(Boolean caseSens, String pname) { // if caseSens is null, get the context ... }
```

This is also a bit problematic when there is already a method with an additional parameter for that method.

#93 - 11/03/2023 12:54 PM - Hynek Cihlar

Dănuț Filimon wrote:

> Hynek Cihlar wrote:
>
> > You won't be able to access the security context from the worker thread, so you will have to prepare all the required context beforehand for fastConvertName or convertName.
>
> Thank you for the information, I've been digging up for a while and I think that a thread is not actually a good idea. I've found several nested methods called by fastConvertName and convertName that end up requiring access to the context (some of which I did not notice even after looking over the code): resolvePathAliases, getExternalProgram, EnvironmentOps.getSearchPath, setSearchPath, EnvironmentOps.getLegacyCaseSensitive. Rewriting these methods to accept the parameter needed from the context doesn't seem at all nice.

Yes, we certainly don't want to rewrite the core services.

To work around this, you make sure you setup the security context for the newly spawned thread. I think you could use ContextAwareThread for this. Constantin, please review this idea.

#94 - 11/06/2023 04:17 AM - Dănuț Filimon

It would be the best if we can have a working thread by using ContextAwareThread. Until the thread discussion is sorted out, I will take a look at it. Currently I still have the solution from #6649-92 up and running so I converted a large customer application over the weekend to check it's impact.

The conversion generated **61k path-mapping** nodes. Those nodes also contains duplicates since the problem with them being saved multiple times was not solved and it's still under investigation. After starting the application, it took **20 minutes** to parse **8k** path-mapping nodes and store ~**650** paths and their fastConvertName result. This is a lot of time considering that are tens of thousands of nodes left to parse so the client will benefit from the cache much, much later...

If the duplication problem is sorted out, there should be less nodes to parse and the time should be greatly reduced. As mentioned in #6649-90, the paths are stored in a Set to avoid calling fastConvertName for paths that were previously parsed.

Another issue should be that SourceNameCache.checkSourceCache will also create an entry in the cache if there is none available for the given

propath so this should also be handled into a thread (this one will not be slow like the initial thread because the paths are already saved) and no other cache can be created while there is no sourceCache.

#95 - 11/06/2023 04:22 AM - Hynek Cihlar

Danut, any numbers on the actual runtime performance with the cache in place?

#96 - 11/06/2023 04:28 AM - Dănuț Filimon

Hynek Cihlar wrote:

> Danut, any numbers on the actual runtime performance with the cache in place?

Not yet, I will test it ASAP and compare it to previous test results. The test will take a while because I have to wait for the cache to fully initialize. I also want to use incremental conversion to test multiple propaths (just the time for initializing the additional cache, not the actual time it takes to find the program).

#97 - 11/06/2023 04:38 AM - Alexandru Lungu

Danut, can you script the duplicates out for now, so you don't have to wait too much? This is just for the performance sake, so we can have some results asap.

#98 - 11/06/2023 04:39 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, can you script the duplicates out for now, so you don't have to wait too much? This is just for the performance sake, so we can have some results asap.

Will do so.

#99 - 11/06/2023 04:51 AM - Hynek Cihlar

The long init time could be brought down with splitting the entries processing on multiple threads. But for now let's focus on the main objective, the runtime performance.

#100 - 11/06/2023 06:23 AM - Hynek Cihlar

Hynek Cihlar wrote:

> The long init time could be brought down with splitting the entries processing on multiple threads.

And by persisting the cache.

#101 - 11/06/2023 07:28 AM - Dănuț Filimon

While testing, the cache is initialized with the default propath, but when a client is connecting, another propath is used which leads to a new cache being created. When trying to use a similar thread like the one I used for initialization, where I get the context and then pass it to the thread I get a NPE. The rest of the application doesn't use the cache because one for the necessary propath is was not created. I am currently implementing the ContextAwareThread and retesting.

#102 - 11/06/2023 08:47 AM - Greg Shah

> The long init time could be brought down with splitting the entries processing on multiple threads.

> And by persisting the cache.

Yes, if the time to build the cache (afer duplicates are removed) is significant, persisting the cache at server shutdown and doing a simple load at server startup is a good solution.

#103 - 11/07/2023 03:20 AM - Dănuț Filimon

I've finally got the results of the testing:

- The initial thread that firstly initializes the sourceCache took **519 seconds** (taking each path-mapping node from the xml, getting the attribute and running fastConvertName);
- Total number of path-mapping nodes (without duplicates) is **3336**;
- The application I tested used a different propath when starting a client and a second cache entry was initialized in **52 ms** because the paths were previously saved when creating the sourceCache entry the first time;
- For the first entry, **72 milliseconds** were spent running fastConvertName and getting it's results;
- The second time **18 milliseconds** were spent;
- The top 3 files in the output were searched **3592**, **1861** and **899** times, and took ~**9 ms** each (this is after the cache was initialized). The 4th file is also worth mentioning, it was searched **833** times and it took **17 ms**;

Overall the changes are showing good results, I am doing another round of testing to check for any regressions.

#104 - 11/07/2023 04:11 AM - Alexandru Lungu

Danut, please check if the number of entries is right. I expected to have >10k (at least the number of files, but also mind the paths of the images, etc.). Anyway, from 61k paths to ~3k is a lot of redundant work done. We absolutely need to find a way to trim the paths from the conversion side, as we are losing a lot of time (almost 10mins) to figure out the duplicates.

As for the results, they really look promising. I don't get the results on the first entry: 9ms * 3592 = ~30s, after optimization, is this right?. We are still losing half a minute resolving the same file all over again, 3.5k times, even with a cache?

#105 - 11/07/2023 04:22 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, please check if the number of entries is right. I expected to have >10k (at least the number of files, but also mind the paths of the images, etc.). Anyway, from 61k paths to ~3k is a lot of redundant work done. We absolutely need to find a way to trim the paths from the conversion side, as we are losing a lot of time (almost 10mins) to figure out the duplicates.
>
> As for the results, they really look promising. I don't get the results on the first entry: 9ms * 3592 = ~30s, after optimization, is this right?. We are still losing half a minute resolving the same file all over again, 3.5k times, even with a cache?

It's the total time, so 3.6k calls in ~9 ms, 1.8k in ~9ms and so on. The number of files is lacking because I only gathered filenames from RUN statements. I did not mention anything about the cache calls and time. The cache returned a value 2.7k times and had a total time of ~25ms, which doesn't actually look promising.

#106 - 11/07/2023 04:37 AM - Dănuț Filimon

AFAIK about paths to images, FileSystemOps.searchPath is used when searching any file and reaches fastConvertName with any file and an extension that was not found in the name_map.xml is ruled out before searching the cache (this was added before getting the context for performance). In my opinion, adding images to path-mappings will just slow down the creation process.

#107 - 11/07/2023 08:15 AM - Dănuț Filimon

My mistake, I just wanted to make an edit on my previous post.

#108 - 11/07/2023 09:49 AM - Dănuț Filimon

- % Done changed from 50 to 90

- Status changed from WIP to Review

Committed 6649b/rev.14803.

- Used AssociatedThread to initialize caches for each new propath;
- Added missing documentation;
- Made changes to checkSourceCache and created createSourceCacheEntry method.

Please review.

#109 - 11/07/2023 10:17 AM - Hynek Cihlar

Dănuț Filimon wrote:

> In my opinion, adding images to path-mappings will just slow down the creation process.

Please clarify. Did you mean the creation of the cache? This is expected, no? If it speeds up runtime mapping then the slow cache creation is fine. I think we want the cache to contain as much entries as possible as we need to consider not only the POC with the limited set of test scenarios.

#110 - 11/08/2023 02:35 AM - Dănuț Filimon

Hynek Cihlar wrote:

> Dănuț Filimon wrote:
>
> > In my opinion, adding images to path-mappings will just slow down the creation process.
>
> Please clarify. Did you mean the creation of the cache? This is expected, no? If it speeds up runtime mapping then the slow cache creation is fine. I think we want the cache to contain as much entries as possible as we need to consider not only the POC with the limited set of test scenarios.

Before starting working on the cache, I remarked that most of the time spent in fastConvertName was done searching files that will always return null. In the first place, fastConvertName should be a method that searches for the java class name of a progress procedure, so these additional file searches were not needed at all. For reference, I mentioned this case in #6649-47 and #6649-56. When iterating the name_map.xml nodes and creating the maps, I also store the file extensions and use them in fastConvertName to check if the current legacyProgName has a valid extension so that the program can be searched.

#111 - 11/08/2023 03:15 AM - Hynek Cihlar

Code review 6649b revisions 14800..14803.

Saving the RUN entries should be saved in a set rather than saving directly in xml. The set would be written to the file during post-rules. See how the other name_map.xml entries are saved in NameMappingWorker with storeMappings and restoreMappings.

What about incremental conversion? For this to work the name entries should be stored per external procedure. When an external procedure is incrementally reconverted all its name entries must be thrown away before new ones are encountered again.

In createSourceCacheEntry locking the whole sourceCache will decrease throughput when potentially multiple propaths are changed at once. It's enough to lock sourceCache when mapperCache is retrieved or stored.

The number of concurrent threads should be more under control. If there is enough of available resources we want to utilize them OTOH we don't want to bring the server down when too many cache create requests come in. I think Executors.newFixedThreadPool would do it, with the number of max threads configurable in the directory.

Different locks are used in createSourceCacheEntry and checkSourceCache when accessing sourceCache.

The cache has a functional flaw we didn't consider. We assumed the file system is static. So when a file is resolved in a dir A and later is moved from A to B the cache will still give A. There should be some configurable mechanism to either turn off the cache completely or partially for specific file patterns.

The footprint of the cache entries could be big with many different propath entries. SourceNameMapperCache should hold only indexes to the names stored in a single list/array.

But before doing any heavy functional changes first please find out the impact of the cache. Whether it is worth the hassle.

#112 - 11/08/2023 03:37 AM - Hynek Cihlar

Dănuț Filimon wrote:

> Hynek Cihlar wrote:
>
> > Dănuț Filimon wrote:
> >
> > > In my opinion, adding images to path-mappings will just slow down the creation process.
> >
> > Please clarify. Did you mean the creation of the cache? This is expected, no? If it speeds up runtime mapping then the slow cache creation is fine. I think we want the cache to contain as much entries as possible as we need to consider not only the POC with the limited set of test scenarios.
>
> Before starting working on the cache, I remarked that most of the time spent in fastConvertName was done searching files that will always return null. In the first place, fastConvertName should be a method that searches for the java class name of a progress procedure, so these additional file searches were not needed at all. For reference, I mentioned this case in #6649-47 and #6649-56. When iterating the name_map.xml nodes and creating the maps, I also store the file extensions and use them in fastConvertName to check if the current legacyProgName has a valid extension so that the program can be searched.

Does it mean that the non-program files don't require the additional fastConvertName logic (not considering the cache) and so they would introduce unnecessary overhead if processed through fastConvertName? On the other hand if there are many of them (or there are many non-program file lookups) then would it make sense to cache them in lookupLegacyName?

#113 - 11/08/2023 04:03 AM - Dănuț Filimon

Hynek Cihlar wrote:

> Does it mean that the non-program files don't require the additional fastConvertName logic (not considering the cache) and so they would introduce unnecessary overhead if processed through fastConvertName?

Yes, only program files that match the pattern defined in build.properties will be converted and only those will be added to name_map.xml. Any other file should be irrelevant when SourceNameMapper.lookupLegacyName is called.

> On the other hand if there are many of them (or there are many non-program file lookups) then would it make sense to cache them in lookupLegacyName?

In terms of number, these files do not amount to much, but are called multiple times. In my opinion, checking the extension is faster than locking and reading from the cache.

#114 - 11/08/2023 07:21 AM - Dănuț Filimon

Hynek Cihlar wrote:

> Code review 6649b revisions 14800..14803.
>
> Saving the RUN entries should be saved in a set rather than saving directly in xml. The set would be written to the file during post-rules. See how the other name_map.xml entries are saved in NameMappingWorker with storeMappings and restoreMappings.

This change solves the path-mapping node duplication.

> What about incremental conversion? For this to work the name entries should be stored per external procedure. When an external procedure is incrementally reconverted all its name entries must be thrown away before new ones are encountered again.

Incremental conversion can duplicate path-mapping nodes, I still need to look into it.

> But before doing any heavy functional changes first please find out the impact of the cache. Whether it is worth the hassle.

I retested a customer application and got the following results for the cache:

- Total time spent in fastConvertName: **619.73 ms**;
- Total cache time (from successful hits where non-null values were returned): **99.82 ms**;
- The number of cache hits (when cache returned a value) was **21288** from the total **35971** fastConvertName calls. From the cache hits, **2524** returned a value and **18764** returned null (in this case it will continue to execute the rest of the fastConvertName).
- Average fastConvertName execution time from the total number of calls when the cache fails: **0.0185 ms**;
- Average execution time of fastConvertName when cache returns a non-null value: **0.0395 ms**;

I am currently looking into the other ideas mentioned in #6649-111.

#115 - 11/08/2023 07:22 AM - Dănuț Filimon

- Status changed from Review to WIP

#116 - 11/08/2023 08:06 AM - Hynek Cihlar

Dănuț Filimon wrote:

- The number of cache hits (when cache returned a value) was **21288** from the total **35971** fastConvertName calls. From the cache hits, **2524** returned a value and **18764** returned null (in this case it will continue to execute the rest of the fastConvertName).

The cache should eliminate 100% of the heavy execution of fastConvertName for all the encountered constant file names in the legacy sources. How come there are 18764 such misses?

#117 - 11/08/2023 08:15 AM - Dănuț Filimon

Hynek Cihlar wrote:

> The cache should eliminate 100% of the heavy execution of fastConvertName for all the encountered constant file names in the legacy sources. How come there are 18764 such misses?

There are two cases:
1. We search a program that doesn't have an entry in the cache, checkSourceCache will return null;
2. We search a program that has an entry in the cache, but fastConvertName returned null when the cache was initialized;

The problem is that I can't differentiate between which case is executed. I am still thinking on how I should address this, check if the key exists before or return an empty String[] and then add new values to the cache during future evaluations.

#118 - 11/08/2023 10:50 AM - Hynek Cihlar

Dănuț Filimon wrote:

> The problem is that I can't differentiate between which case is executed. I am still thinking on how I should address this, check if the key exists before or return an empty String[] and then add new values to the cache during future evaluations.

Please do so. We will need some performance tests on the customer test cases to see how much of an improvement the cache will make. Until we don't have these numbers let's focus only on the changes that provide performance improvements. Things like multiple threads, persistence, directory configuration, etc. can wait until we have good understanding of the core performance impact.

#119 - 11/09/2023 01:01 AM - Dănuț Filimon

Got it, I went and deleted checkSourceCache and integrated it's functionality in fastConvertName, this was the best idea that I came for to avoid returning an empty String[].

Hynek Cihlar wrote:

> The footprint of the cache entries could be big with many different propath entries. SourceNameMapperCache should hold only indexes to the names stored in a single list/array.

I also made the changes to the cache so that when no entry is found, the result from fastConvertName will be added for the legacyProgName searched. The path-mapping duplication was solved and mentioned previously, but the incremental conversion still duplicates/deletes nodes so I still need to look into it. As you mentioned, we can use a list to store the paths from the nodes since there will be no duplicates, but should the legacyProgName searched be added to this list and mapped appropriately in the cache with the fastConvertName result? This could work since there is no propath involved and when a new propath cache is created, the results will be retrieved from the cache instead of relying on fastConvertName to lookup / fallback to convertName.

#120 - 11/09/2023 02:23 AM - Hynek Cihlar

Dănuț Filimon wrote:

> Got it, I went and deleted checkSourceCache and integrated it's functionality in fastConvertName, this was the best idea that I came for to avoid returning an empty String[].
>
> Hynek Cihlar wrote:
>
> > The footprint of the cache entries could be big with many different propath entries. SourceNameMapperCache should hold only indexes to the names stored in a single list/array.
>
> I also made the changes to the cache so that when no entry is found, the result from fastConvertName will be added for the legacyProgName searched. The path-mapping duplication was solved and mentioned previously, but the incremental conversion still duplicates/deletes nodes so I still need to look into it. As you mentioned, we can use a list to store the paths from the nodes since there will be no duplicates,

I mentioned list, but more precisely this should be a sorted set. A collection that will not allow duplicates and can be accessed with an index. I will call this master entries sets. There will be two of them. One for the query file name strings and second for the resolved file name strings. Their only purpose is to make sure the Java strings don't occupy the heap multiple times.

> but should the legacyProgName searched be added to this list and mapped appropriately in the cache with the fastConvertName result?

Do you mean to add entries to the sorted set, which don't yet exist there? Yes, for anything that will be added to the cache, the master sets will need to be updated, too.

> This could work since there is no propath involved and when a new propath cache is created, the results will be retrieved from the cache instead of relying on fastConvertName to lookup / fallback to convertName.

When new propath is created you will have to recalculate everything because the outcomes may change with the different propath.

#121 - 11/09/2023 06:17 AM - Dănuț Filimon

I retested the performance of the cache after saving the fastConvertName results of program names that are not found in the initial mappings and obtained better results compared to #6649-114:

- Total time spent in fastConvertName: **298.764 ms**. Total calls: **36900**;
- Total time when the cache returns successfully: **44.495 ms**. Total calls: **14700**;
- The rest of the calls make the times when values are introduced into the cache: **8656** and times when the extension is not valid: **13544**;
- Average call time when the program name is found in the cache (time/number of calls): **0.003 ms**;
- Average call time when the program name is not found in the cache and lookup/convertName is done: **0.0281 ms**;
- Average call time when program extension is invalid: **0.00075 ms**;

There is clearly an improvement when using a cache. I had to modify the implementation and solve a few concurrency problems before testing the scenario. Currently, there is a single problem left with using the query file set that is used to populate the cache, I am currently using a copy of the list, but somehow the unmodifiable set I am creating from the group file set is still concurrently accessed.

Hynek, what do you think about the results? Also, I could not find a collection that does not allow duplicates and has access through an index. SortedSet does not support access through index. Did you mean something else, possibly something we already have implemented?

#122 - 11/09/2023 09:26 AM - Hynek Cihlar

Dănuț Filimon wrote:

> I retested the performance of the cache after saving the fastConvertName results of program names that are not found in the initial mappings and obtained better results compared to #6649-114:
>
> - Total time spent in fastConvertName: **298.764 ms**. Total calls: **36900**;
> - Total time when the cache returns successfully: **44.495 ms**. Total calls: **14700**;
> - The rest of the calls make the times when values are introduced into the cache: **8656** and times when the extension is not valid: **13544**;
> - Average call time when the program name is found in the cache (time/number of calls): **0.003 ms**;
> - Average call time when the program name is not found in the cache and lookup/convertName is done: **0.0281 ms**;
> - Average call time when program extension is invalid: **0.00075 ms**;
>
> There is clearly an improvement when using a cache. I had to modify the implementation and solve a few concurrency problems before testing the scenario. Currently, there is a single problem left with using the query file set that is used to populate the cache, I am currently using a copy of the list, but somehow the unmodifiable set I am creating from the group file set is still concurrently accessed.
>
> Hynek, what do you think about the results?

The number look good on the paper. The question is how will it perform in the actual customer scenario.

> Also, I could not find a collection that does not allow duplicates and has access through an index. SortedSet does not support access through index. Did you mean something else, possibly something we already have implemented?

In that case a sorted list/array will do. Or maybe better two hash maps, one for the lookup index->name and the other for the lookup name->index. Also hash maps will handle insertions better.

#123 - 11/09/2023 09:31 AM - Alexandru Lungu

> The number look good on the paper. The question is how will it perform in the actual customer scenario.

I can take the changes and test with them on my profiling tests. Let me know when the changes are completely ready by moving this into "Internal Test". At that point, I will start reconverting and running the performance tests.

#124 - 11/10/2023 09:02 AM - Dănuț Filimon

- Status changed from WIP to Review

Committed 6649b/rev.14804.

- After solving the path-mapping node duplication problem, I reverted the changes from NameMappingWorker.java because nodes were still being duplicated by the storeMappings method;
- Created SOURCE_CACHE_SIZE property used to set the size of the sourceCache;
- Used an ExecutorService instance which allows a single thread at a time to create sourceCache entries, it is also used when reading the path-mapping nodes from the name_map.xml file which doesn't necessarily need to use an AssociatedThread (this needs to be changed to use a normal thread but I've missed it);
- Removed chechSourceCache method and integrated it in fastConvertName;
- Modified createSourceCacheEntry method, it will be the only method that can instantiate sourceCache and create new entries;
- Created writeConvertedCache which will add a new path to the SourceNameMapperCache stored in the cache for the current propath and will add this path to the saved ones we saved from name_map.xml;
- Added another fastConvertName method which takes an additional boolean parameter (by default is false). When the parameter is true, it will avoid checking the cache, creating a new entry or writing to an existing one. This should be exclusively used by the thread to avoid concurrency problems related to locking;
- SourceNameMapper will store a Map<String, Integer> instance which will map the program name from the name_map.xml and the ones added when writing new entries to the cache. Each SourceNameMapperCache instance will also store a Map<Integer, String[] which will use the index of the program to get the correct converted result. An AtomicInteger instance is retrieved and incremented accordingly whenever a new program is added into the first map.

There are a lot of changes and I also left a few comments where I saw them necessary. Note that I addressed most comments from #6649-111 but there's still problems with incremental conversion where nodes are still duplicated, normal conversion should be good to go in this case. I also didn't use Executors.newFixedThreadPool, but Executors.newSingleThreadExecutor.

Please review.

#125 - 11/12/2023 04:29 AM - Hynek Cihlar

Code review 6649b revision 14804.

initializationPaths.size() == 0 in createSourceCacheEntry should be protected.

Ideas for future improvements:

- Making the cache init threads lower priority.
- Sharding and using multipe threads for the cache init. By sharding I mean to split the init path entries into multiple chunks and submit each chunk to the executor.
- The number of threads for the cache init should be externally configurable.
- The LRU cache max number of entries should be externally configurable. When you already spend the precious resources to build the cache you will also want to have the control over the eviction.
- When the cache is being initialized only the affected propaths should be locked, the other propaths should be still available for reading.

#126 - 11/13/2023 05:06 AM - Dănuț Filimon

- Status changed from Review to WIP

Committed 6649b/rev.14805 and 6649b/rev.14806:

- Made sourceCache configurable through CacheManager. I previously mentioned that the SourceNameMapper can be initialized during conversion and wasn't sure if we should use the CacheManager in this case. Since the SourceNameMapper is initialized before the CacheManager, we can use it to instantiate the sourceCache with default/configured value. Note that the cache manager was initially created for caches from the persist package, but it can be used in other packages as long as the cache sizes were read from the directory configuration. To configure the size of the cache you need to add the following in the persistence/cache-size container:

```
            <node class="container" name="13">
              <node class="string" name="class-name">
                <node-attribute name="value" value="com.goldencode.p2j.util.SourceNameMapper"/>
              </node>
              <node class="integer" name="size">
                <node-attribute name="value" value="16"/>
              </node>
            </node>
```

- Replaced the AssociatedThread with a normal thread in initMappingData so that the mappingExecutor will have a single purpose - to initialize the cache and it's entries. This will also help if we split the init path entries and submit them to the executor, assuming that multiple threads will be used;
- Removed unnecessary logic from the path mapping initializer thread and I noticed that we should allow the sourceCache to be initialized even if we have no path-mappings so the if (pmaps == null) { return; } that was added should either be removed or assign true to pathMappingInitialized before returning (this check was removed in **rev.14806**);

Alexandru, the current implementation can be tested. I will address the improvement ideas mentioned by Hynek in #6649-125, but I would like you to reconvert and run your performance tests. I don't know if this issue should be moved to Internal Test if I still intend to work on it.

#127 - 11/13/2023 05:09 AM - Dănuț Filimon

I also tried to find out the cause of the incremental version where nodes are duplicated but after running a ./gradlew all on the branch I can't replicate the problem. During incremental conversion, nodes were removed/added correctly and I did not notice any other problems.

#128 - 11/13/2023 05:16 AM - Dănuț Filimon

- Status changed from WIP to Internal Test

#129 - 11/16/2023 12:40 AM - Dănuț Filimon

Committed 6649b/rev.14807. Fixed a NPE when there are no path-mapping nodes in name_map.xml.

#130 - 11/17/2023 04:04 AM - Alexandru Lungu

I converted a large application and generated a name_map.xml with ~39k (~7k of OO classes) class-mappings and only ~1k path mappings. From my POV, this is quite low comparing to the number of files in the project. I don't say the changes are bad - it is just that it is an unexpected low number. I am compiling and profiling the solution right now - fingers crossed.

#131 - 11/17/2023 06:26 AM - Alexandru Lungu

I've done some tests, and there were no obvious problems. I am moving this to the profiler asap.

However, I want to have some optimizations first of all:

- Avoid computing the hash code of the propath on each look-up. Consider caching it in the work area. Mind that my test has ~70 propath entries.
- Can we have a faster way to compare two PROPATHs for equality? Right now, they are both iterated and a string-by-string comparison is done. Please make SourceNameMapper.setSearchPath LRU cache the results (low size; maybe somewhere at 64). This way, each context will pick up the only instance associated with a propath and == will work faster.

#132 - 11/17/2023 07:08 AM - Alexandru Lungu

Some extra ideas:

- Also, what about convertedNames? Can't it possible leak as we are constantly adding to it the results of fastConvertName? I am thinking of an example of a loop that invokes functions dynamically with generated name. In some cases, the function may not exist, so we cache null. If we run this example multiple times, we may end up with really big convertedNames that leak. While I agree that initializationPaths has a predefined size (based on name_map.xml), this doesn't hold for convertedNames. Please advice. In my tests, these maps don't leak - but it is possible with some badly designed 4GL code. My example has a convertedNames with ~2.3k entries.
- Maybe we can even skip the canonicalize. We can attempt to do the map look-up without canonicalizing. If there is no index cached, we can retry we the canonicalized version.
- I would tend to prioritize Hynek;s When the cache is being initialized only the affected propaths should be locked, the other propaths should be still available for reading.

Danut please fix #6649-131 - I am planning to start performance testing with 6649a as soon as possible.

#133 - 11/17/2023 07:33 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut please fix #6649-131 - I am planning to start performance testing with 6649a as soon as possible.

Committed 6649b/rev.14808. Made the necessary changes for #6649-131. I will write an answer for #6649-132 shortly.

#134 - 11/17/2023 08:03 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Some extra ideas:
>
> - Also, what about convertedNames? Can't it possible leak as we are constantly adding to it the results of fastConvertName? I am thinking of an example of a loop that invokes functions dynamically with generated name. In some cases, the function may not exist, so we cache null. If we run this example multiple times, we may end up with really big convertedNames that leak. While I agree that initializationPaths has a predefined size (based on name_map.xml), this doesn't hold for convertedNames. Please advice. In my tests, these maps don't leak - but it is possible with some badly designed 4GL code. My example has a convertedNames with ~2.3k entries.

This is of great concern and I will address it immediately. Initially I had a method that accessed the cache (checkSourceCache) which returned the result from the cache, the problem with this method is that it returned null when there was no entry and null for a key that was introduced and it's result was null. This lead to this method being removed and incorporated in fastConvertName, and a few other changes made it possible to check the initial map of search-specs to see if there should be an entry in the cache and also check the actual SourceNameMapperCache.convertedNames map using containsKey so that we are sure that this program was searched previously and we can return null. I did not use a LRUCache for SourceNameMapperCache because I couldn't confirm if the null value returned is because that is the result of fastConvertName or if the key doesn't exist in the cache. If we add the containsKey method for ExpiryCache, we can easily resolve this.

> - Maybe we can even skip the canonicalize. We can attempt to do the map look-up without canonicalizing. If there is no index cached, we can retry we the canonicalized version.

We have to avoid unnecessary cache lookups so I'll have to look into it a bit. I remember a problem where the program name was cached with slashes instead of backslashes, this may lead to the same issue.

> - I would tend to prioritize Hynek;s When the cache is being initialized only the affected propaths should be locked, the other propaths should be still available for reading.

I was thinking that I could pass the sharded initialization paths and the propath to the executor (assuming that multiple threads are used) and only do a write lock when creating the cache/entry. If you have a different idea in mind, let me know.

#135 - 11/17/2023 08:13 AM - Alexandru Lungu

Review of 6649b/rev.14808

Mind that the search-path cache is not synchronized. Please add synchronize to that access of that cache. Don't bother using a reentrant lock as the goal of the cache is not to save time, but to ensure that the same instances are returned for the same input (with a high probability).

> We have to avoid unnecessary cache lookups so I'll have to look into it a bit. I remember a problem where the program name was cached with slashes instead of backslashes, this may lead to the same issue.

I was thinking that in 99% of the cases, canonicalize is not required. However, this was my quite random thought. You may ignore this this point.

> I was thinking that I could pass the sharded initialization paths and the propath to the executor (assuming that multiple threads are used) and only do a write lock when creating the cache/entry. If you having a different idea in mind, let me know.

I think we need to make sure we have multiple locks (one per SourceNameMapperCache maybe). This way, we don't have to synchronize users that have different propaths. Consider having one RWLock per SourceNameMapperCache.

#136 - 11/17/2023 08:23 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Mind that the search-path cache is not synchronized. Please add synchronize to that access of that cache. Don't bother using a reentrant lock as the goal of the cache is not to save time, but to ensure that the same instances are returned for the same input (with a high probability).

Committed 6649b/rev.14809.

#137 - 11/20/2023 02:34 AM - Hynek Cihlar

Alexandru Lungu wrote:

> I've done some tests, and there were no obvious problems. I am moving this to the profiler asap.
>
> However, I want to have some optimizations first of all:
>
> - Avoid computing the hash code of the propath on each look-up. Consider caching it in the work area. Mind that my test has ~70 propath entries.

Do you mean 70 individual propath entries or 70 paths in a single propath?

- Can we have a faster way to compare two PROPATHs for equality? Right now, they are both iterated and a string-by-string comparison is done. Please make SourceNameMapper.setSearchPath LRU cache the results (low size; maybe somewhere at 64). This way, each context will pick up the only instance associated with a propath and == will work faster.

Can we use HashMap (or HashSet)?. It already takes care of hash collisions and does it in a way that it doesn't iterate all the entries but only the colliding ones.

#138 - 11/20/2023 04:00 AM - Alexandru Lungu

Hynek Cihlar wrote:

> Do you mean 70 individual propath entries or 70 paths in a single propath?

One single long propath with 70 paths. To do the look-up, the convertName function will theoretically have to iterate 70 entries. From my POV, this is big.

> - Can we have a faster way to compare two PROPATHs for equality? Right now, they are both iterated and a string-by-string comparison is done. Please make SourceNameMapper.setSearchPath LRU cache the results (low size; maybe somewhere at 64). This way, each context will pick up the only instance associated with a propath and will work faster.

> Can we use HashMap (or HashSet)?. It already takes care of hash collisions and does it in a way that it doesn't iterate all the entries but only the colliding ones.

Do you mean using HashMap instead of LRU cache? My concern is that two users that have the same original propath will end up with two different arrays (storing different string instances). Thus, will never short-circuit the comparison. That is why I want to have the **same** array instance with the **same** string instances generated for two different users - to benefit from instance equality (==). FYI, right now, the propaths are compared using Arrays.equals which does char-by-char comparison for each propath path (in my case, ~70).

#139 - 11/20/2023 06:03 AM - Alexandru Lungu

On a large POC, the total improvement of the changes in 6649b was of -0.3%.

#140 - 11/21/2023 09:35 AM - Dănuț Filimon

6649b was rebased to **trunk/rev.14834** and the current revision is **14845**.

Committed 6649b/rev.14846:

- Used one lock per SourceNameMapperCache for read/write operations;
- Synchronized access to sourceCache and initializationPaths;
- Added containsKey() to ExpiryCache to avoid the potential map leak mentioned by Alexandru;
- Slightly reduced hash computation of the propath by passing it as a parameter.

Please review and advise. In my opinion, the lock used in SourceNameMapperCache has it's own methods for read/write lock since convertedCache.lock.readLock().lock() wasn't descriptive, I am thinking on reducing these to only 2 methods (one for read and one for write) where I can pass a Runnable.

#141 - 11/21/2023 11:24 AM - Hynek Cihlar

Alex, when you test the large app, do you use 7156b with the changes from 6649b on top? Will the latest 7156b work OK?

#142 - 11/22/2023 04:11 AM - Alexandru Lungu

I tested with a rebased 6649b from 7156b (7156b at rev. 14846, branched from trunk rev. 14822 + all 8 commits from 6649b). Everything is fine with this configuration (clean, convert, jar, import, deploy)!

#143 - 11/24/2023 02:11 AM - Dănuț Filimon

There is an issue with how the cache entries are built. The AssociatedThread doesn't use the correct context when building the cache entry, but the cache key and SourceNameMapperCache use the correct propath. When running the task for the thread, the context uses a propath with 9 entries, but the one actual one that needs to be used has 12 entries. Is this a problem with the AssociatedThread implementation? This was really hard to notice since the larger propath contains all the elements from the smaller one.

#144 - 11/24/2023 06:05 AM - Dănuț Filimon

After investigating, I found out that the mappingExecutor creates an AssociatedThread with the current context, meaning that when a new source cache entry is created it will tun a task using the same context it was created (which is the expected behavior). There are no problems with the AssociatedThread, it's just an implementation issue that I am actively working on right now. One of the possible solutions was to remove the executor for good and create the AssociatedThread in createSourceCacheEntry but this causes other threads from different appservers to throw exceptions so it's not a correct solution. Synchronizing the method is also not an option.

I looked over the ExecutorService and we can simply initialize the mappingExecutor and not override the newThread method. We will simply create the AssociatedThread with the task and set it up before submitting it to the executor. But the local context propath obtained also doesn't match the propath used, on a second look another WorkArea is initialized! This turns out to be more complicated, the new WorkArea doesn't have any initial value set so it goes through the statements and sets them up again.

It is also not an option to pass the propath to the fastConvertName, almost all values from the WorkArea influence how the output will look like so this will result in all of them being required. Are there any options left which I didn't think of?

At the moment, only when the executor is not used and separate threads are created the context is correct, in the rest of the cases a new one is initialized which I don't quite understand why, especially the case where the thread is created when it is needed and then submitted.

#145 - 11/24/2023 06:11 AM - Dănuț Filimon

- Status changed from Internal Test to WIP

#146 - 11/28/2023 05:50 AM - Dănuț Filimon

- Status changed from WIP to Review

Committed 6649b/rev.14847 which solves the problem mentioned in #6649-144. By not keeping the thread idle, the correct context is used when the source cache entry is created. Hynek/Alexandru, please review.

#147 - 11/28/2023 09:20 AM - Dănuț Filimon

I ran a large POC with **7156b/rev.14846** and the changes from **6649b/rev.14835 - 14847** and obtained good results (the only thing to note here is that I borrowed the name_map.xml from Alexandru since he reconverted the code and I let the application initialize the path-mappings map before starting the test). From 5 runs each (baseline and 7156b with 6649b) I obtained ~**6.5%** improvement for the **average of the last 20 runs** and ~**3%** improvement for the **total average of the 100 runs**.

#148 - 12/02/2023 03:36 PM - Hynek Cihlar

Code review 6649b revisions 14840..14847.

In createSourceCacheEntry the object submitted to the mappingExecutor is a Thread, but it should be a plain Runnable. The executor will use default thread factory and so it will really create plain Thread objects instead of AssociatedThread. Also the actual thread will not receive the custom name due to the default thread factory.

In createSourceCacheEntry the LRU cache is created with default size 16384 with grow policy STRICT. Shouldn't it be created with grow policy LENIENT?

Some other points for the future:

When createSourceCacheEntry is called while !pathMappingInitialized, shouldn't the request be enqueued and cache initialized later when pathMappingInitialized?

The implementation still doesn't take into account the dynamic nature of the underlying file system. The cache result should be somehow invalidated when the related file system portion changes or there should be configurable exclusions for files known to be dynamic. This is something to be considered/resolved in the future.

The sizes of the LRU caches should be externally configurable.

I think the worker threads should receive a lower priority.

The logic gets more and more complex so it would be nice to be able to know how does it perform and provide necessary adjustments. For example to get the LRU cache sizes right. For this some JMX counters would be nice.

#149 - 12/05/2023 03:15 AM - Dănuț Filimon

Hynek Cihlar wrote:

> In createSourceCacheEntry the object submitted to the mappingExecutor is a Thread, but it should be a plain Runnable. The executor will use default thread factory and so it will really create plain Thread objects instead of AssociatedThread. Also the actual thread will not receive the custom name due to the default thread factory.

The AssociatedThread is necessary since the local context is required by the thread, I switched the implementation and used a ThreadFactory to create and name the thread for the task.

> In createSourceCacheEntry the LRU cache is created with default size 16384 with grow policy STRICT. Shouldn't it be created with grow policy LENIENT?

Initially the cache entry was a Map, but a potential leak was noticed that included creating file names dynamically and I changed it to a LRUCache. We can use LENIENT, as long as the cache remains closer to the imposed limit. From what I understood, it will keep trying to remove items and keep it closed to the specified size.

> Some other points for the future:
> ...
> The sizes of the LRU caches should be externally configurable.

The sizes are configurable through CacheManager (sourceCache and SourceNameMapperCache.convertedNames), are you talking about the searchPathCache?

> I think the worker threads should receive a lower priority.

This was mentioned before and it should be no problem setting the priority of the threads to either 3 or 4.

I **committed 6649b/rev.14848** where I made the following changes:

- Changed the grow policy of convertedNames from STRICT to LENIENT;
- The threads are configurable and the directory specification for the persistence/cache-size container is as follows:

```
<node class="container" name="13">
    <node class="string" name="class-name">
        <node-attribute name="value" value="com.goldencode.p2j.util.SourceNameMapper"/>
    </node>
    <node class="integer" name="size">
        <node-attribute name="value" value="16384"/>
    </node>
    <node class="string" name="discriminator">
        <node-attribute name="value" value="names"/> <!-- the value must be "names" as specified here -->
    </node>
</node>
<node class="container" name="14">
    <node class="string" name="class-name">
        <node-attribute name="value" value="com.goldencode.p2j.util.SourceNameMapper"/>
    </node>
    <node class="integer" name="size">
        <node-attribute name="value" value="8"/>
    </node>
    <node class="string" name="discriminator">
        <node-attribute name="value" value="source"/> <!-- the value must be "source" as specified here --
>
    </node>
</node>
```

- I looked into the the SynchronousQueue used by the executor and it is clearly not the best fit for the current situation since it does not have a capacity. I replaced it with a LinkedBlockingQueue with a capacity of 5;
- Created SourceThreadFactory which will create the AssociatedThread for the executor, it will also create them with a lower priority (4);

- Fixed a few typos and documentation.

  When createSourceCacheEntry is called while !pathMappingInitialized, shouldn't the request be enqueued and cache initialized later when pathMappingInitialized?

Note that for **3k** path-mapping nodes, the thread will take ~**5-8 minutes** to parse all the nodes before initializationPaths is populated with all the names. In this case if we let requests to be enqueued, we will have to keep track of requests for the same propath so that we don't have a request for a cache that is already waiting to be created.

#150 - 12/05/2023 05:14 AM - Hynek Cihlar

Dănuț Filimon wrote:

> Hynek Cihlar wrote:
> Initially the cache entry was a Map, but a potential leak was noticed that included creating file names dynamically and I changed it to a LRUCache. We can use LENIENT, as long as the cache remains closer to the imposed limit. From what I understood, it will keep trying to remove items and keep it closed to the specified size.

I didn't consider the possibility of dynamic queries. Yes, with this in mind it makes sense to limit the LRU cache on the number of entries. Whether it makes sense to make the LRU cache LENIENT or not I will keep up to you.

> > Some other points for the future:
> > ...
> > The sizes of the LRU caches should be externally configurable.
>
> The sizes are configurable through CacheManager (sourceCache and SourceNameMapperCache.convertedNames), are you talking about the searchPathCache?

If searchPathCache is not externally configurable, then I meant searchPathCache :-).

> I **committed 6649b/rev.14848** where I made the following changes:
>
> - Changed the grow policy of convertedNames from STRICT to LENIENT;
> - The threads are configurable and the directory specification for the persistence/cache-size container is as follows:
>   [...]
> - I looked into the the SynchronousQueue used by the executor and it is clearly not the best fit for the current situation since it does not have a capacity. I replaced it with a LinkedBlockingQueue with a capacity of 5;

With the LinkedBlockingQueue on the limit, adding more tasks to the executor will cause RejectedExecutionException. Is this intended?

createSourceCacheEntry should detect the same propath already queued (if it doesn't already). Also I think it would make sense to prioritize the work queue based on the demand. The propaths requested the most should receive the highest priority. With this in place I would reconsider the need to limit the capacity of the work queue.

> > When createSourceCacheEntry is called while !pathMappingInitialized, shouldn't the request be enqueued and cache initialized later when pathMappingInitialized?

> Note that for **3k** path-mapping nodes, the thread will take ~**5-8 minutes** to parse all the nodes before initializationPaths is populated. In this case if we let requests to be enqueued them we will have to keep track of requests for the same propath so that we don't have a request for a cache that is already waiting to be created.

There is probably not a big functional impact of queuing the requests anyway. If you don't see any benefit there, just please disregard this.

Btw. those 5-8 minutes. These really include only reading the 3K XML entries and putting them in initializationPaths? This is surprisingly long considering there is no heavy logic involved.

#151 - 12/05/2023 05:33 AM - Dănuț Filimon

Hynek Cihlar wrote:

> With the LinkedBlockingQueue on the limit, adding more tasks to the executor will cause RejectedExecutionException. Is this intended?

I tested the threads using **3** as the priority and I the same exception when using SynchronousQueue. There was no space in the queue because there is no capacity for this type of BlockingQueue. I switched to a LinkedBlockingQueue for this reason, to allow more jobs to be queued. But yes, it can cause a RejectedExecutionException if the capacity (5) is exceeded.

> createSourceCacheEntry should detect the same propath already queued (if it doesn't already).

It does not. The cache is created based on the hasCache value in fastConvertName and is the only check done for the cache entry.

> Btw. those 5-8 minutes. These really include only reading the 3K XML entries and putting them in initializationPaths? This is surprisingly long considering there is no heavy logic involved.

Yes. I am also wondering why there is so much time spent there and I think it's worth looking into it. I've made several changes so this is why the time is **5-8**, I will measure the time again before anything else.

#152 - 12/05/2023 08:58 AM - Dănuț Filimon

Hynek Cihlar wrote:

> The logic gets more and more complex so it would be nice to be able to know how does it perform and provide necessary adjustments. For example to get the LRU cache sizes right. For this some JMX counters would be nice.

I created two counter for cache hits/misses and tested a large application. I got a total of **55155 hits** and **2192 misses**, these exclude calls where the cache is avoided (a cache entry is created) and calls that are short-circuited because the extension is not found in allowedLegacyExtensions. The **3k** path-mappings + the **2.2k** cache misses amount to **5.2k** cache entries while the LRUCache has a size of **16384**.

#153 - 12/08/2023 05:54 AM - Dănuț Filimon

Committed 6649b/rev.14849 where I made the following changes:
- Created MonitorRunnable which allows identifying previously submitted propaths using their hash code. Basically you initialize a MonitorRunnable with the **propathHash** and **task** (Runnable instance), the hash is stored in a HashSet with synchronized access and whenever createSourceCacheEntry is called, the hash provided must not be a part of the available set.
- Removed the limit on the LinkedBlockingQueue, making it an unbound queue, after a series of tests.
- Made searchPathCache configurable, the discriminator used is "search" and the default value is **64**.

The best part is that the set is not causing an overhead, just during the initial stages when the cache entries are initialized. The createSourceCacheEntry method is called just when there is no cache available, or the avoidCache value is set to true.

Hynek, can you take a quick look at the changes and tell me your thoughts? We can leave the prioritization part of the propath for later or ignore it.

#154 - 12/08/2023 08:38 AM - Alexandru Lungu

Rebased 6649b to latest trunk. 6649b is now at rev. 14878.

#155 - 12/11/2023 01:19 PM - Hynek Cihlar

Code review 6649b revision 14878.

MonitorRunnable.identifiers are never cleared. Shouldn't an entry be removed, when the particular identifier is added to the LRU cache?

LOG.info("Previously submitted task identifier was found, skipping the task."); should be logged with a finer log level. The condition may happen often clogging the log output.

Otherwise the changes look good.

#156 - 12/12/2023 02:54 AM - Alexandru Lungu

- % Done changed from 90 to 100

- Priority changed from High to Urgent

Danut, please mark this as Urgent; we need to test it asap.

Thus, address the review from Hynek and lets move on with the testing:

- Please apply the changes from 6649b over 7156b to do the testing of the following:
  - profile POC app [ddf]
  - run a large customer application [ddf]
- Simply test with 6649b:
  - smoke test a large customer application [ddf]
- Run regression tests
  - Do the legacy set-up of ChUI regression tests (if the xfer set-up doesn't work for you). Add your changes to 6667e and run the tests.

Constantin, can you run the ETF tests with 6649b once ready? Thank you.

#157 - 12/12/2023 02:55 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Constantin, can you run the ETF tests with 6649b once ready? Thank you.

Will do.

#158 - 12/12/2023 03:32 AM - Dănuț Filimon

I addressed the review from Hynek in **6649b.rev.14879** where I added a removeIdentifier method and changed the logging level when an identifier was already queued. I'll make the tests my top priority now and will try to share the results asap.

#159 - 12/12/2023 03:33 AM - Dănuț Filimon

- Status changed from Review to Internal Test

#160 - 12/12/2023 04:44 AM - Hynek Cihlar

Dănuț Filimon wrote:

> I addressed the review from Hynek in **6649b.rev.14879**

Code review. The changes are good.

#161 - 12/12/2023 05:07 AM - Dănuț Filimon

I smoke tested a large customer application with 6649b and everything worked well. Currently running POC tests.

#162 - 12/12/2023 07:04 AM - Dănuț Filimon

The results of the POC test are in :)

	Baseline (4 cold tests avg)	7156b + 6649b (5 cold tests avg)	Difference (%)
average of the last 20 runs	15113.25	14066.2	-7.05
total average	14821.5	14214.2	-4.09

The performance improvement is a bit better from the previously one mentioned in #6649-147.

#163 - 12/12/2023 11:18 AM - Alexandru Lungu

Danut, is it possible to test 6649b without reconverting? This should theoretically run with no problem and build up the cache at run-time. I will like to do some regression tests on that part.

EDIT: maybe attempt to run the POC without a precomputed name_map.xml. I am curious if in that case we can see any improvement.

#164 - 12/12/2023 02:31 PM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, is it possible to test 6649b without reconverting? This should theoretically run with no problem and build up the cache at run-time. I will like to do some regression tests on that part.

Of course! I actually tested the large customer application without the path-mapping nodes (since I changed the sources 2 times for the app already since my last conversion). It will run with no problems.

> *EDIT*: maybe attempt to run the POC without a precomputed name_map.xml. I am curious if in that case we can see any improvement.

As for the POC, you already provided me with the converted name_map.xml and I've been using it since. Usually, the time it takes to read the ~600 path-mapping nodes is the same with the time it takes to initialize the appservers. If necessary, I can do run the tests again with the original name_map.xml.

The only thing worth noting is that a cache entry will not be created until all the path-mapping nodes are read. I actually considered removing this functionality, as the cache works well even without these nodes found during conversion and because it takes a lot of time to read all the nodes, making the cache unavailable during this time.

#165 - 12/13/2023 06:45 AM - Alexandru Lungu

Danut, lets focus merging this by EOD. What testing is left?

#166 - 12/13/2023 06:46 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Danut, lets focus merging this by EOD. What testing is left?

Are all changes in 6649? I can go ahead with ETF testing.

#167 - 12/13/2023 06:48 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, lets focus merging this by EOD. What testing is left?

Currently fixed the import.db for the customer application to test the changes with 7156b (according to #7156-149). I had a few problems with the dockers last night so I still need to build the docker image for the regression tests, this is going to take a while since I need to test the customer application first.

#168 - 12/13/2023 07:46 AM - Alexandru Lungu

Danut, I am planning to do rebase of 6649b to the latest trunk now.

#169 - 12/13/2023 08:02 AM - Alexandru Lungu

Rebased 6649b to latest trunk. It is now at rev. 14886.

Constantin, you can start the ETF tests. These tests are nice because they also do the conversion, generating some optional artifacts that will better test the run-time.

#170 - 12/13/2023 09:01 AM - Dănuț Filimon

I finished testing a customer application with 7156b and 6649b and got between 18 (first run) and 21 (seconds run) tests that failed. Alexandru asked me to test #7669-1 since I've already warmed up the server. Sadly it is not possible to run the chui regression tests today, but those will be my top priority the next morning!

#171 - 12/13/2023 09:06 AM - Alexandru Lungu

It is normal to have 18-21 tests failed on that regression tests.

As of my understanding: if ETF passes, only ChUI regression tests are left to pass. I think the amount of testing here is already considerable. If you can deliver an answer on the Majic regression tests first thing tomorrow morning, I think we are fine with it and proceed merging at that time.

#172 - 12/13/2023 09:52 AM - Dănuț Filimon

I ran the #7669-1 test and one of the users failed during the run 75 due to a StaleRecordException. I am running 7156b/rev.14846 which is not the latest and it seems that this exception is solved in 7156b/rev.14878. The rest of the users ran without any other problems.

#173 - 12/13/2023 10:08 AM - Greg Shah

Alexandru Lungu wrote:

> It is normal to have 18-21 tests failed on that regression tests.
>
> As of my understanding: if ETF passes, only ChUI regression tests are left to pass. I think the amount of testing here is already considerable. If you can deliver an answer on the Majic regression tests first thing tomorrow morning, I think we are fine with it and proceed merging at that time.

Agreed.

#174 - 12/14/2023 05:37 AM - Dănuț Filimon

Great news! I've ran the regression tests and all tests passed successfully.

#175 - 12/14/2023 08:49 AM - Constantin Asofiei

Together with Alexandru we've managed to find a cause and solution for the trunk regression. I'm working on running ETF for 6649b.

#176 - 12/14/2023 09:43 AM - Alexandru Lungu

Constantin, if 6649b passes ETF, we can merge it to trunk. Please let me know when to proceed.

#177 - 12/14/2023 10:32 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Constantin, if 6649b passes ETF, we can merge it to trunk. Please let me know when to proceed.

There are a few tests which fail compared with trunk, I'm trying to understand why.

#178 - 12/14/2023 10:58 AM - Constantin Asofiei

Constantin Asofiei wrote:

> Alexandru Lungu wrote:
>
> > Constantin, if 6649b passes ETF, we can merge it to trunk. Please let me know when to proceed.
>
> There are a few tests which fail compared with trunk, I'm trying to understand why.

ETF can be considered as passed with 6649b.

#179 - 12/14/2023 11:01 AM - Alexandru Lungu

As this was the last testing required for #6649, I am preparing to merge 6649b.

#180 - 12/14/2023 11:08 AM - Greg Shah

Yes, please merge now.

#181 - 12/14/2023 11:19 AM - Alexandru Lungu

- Status changed from Internal Test to Test

Branch 6649b was merged to trunk rev 14874 and archived.

#182 - 12/15/2023 02:51 AM - Dănuț Filimon

Database Configuration was updated with the necessary information for configuring the cache size of SourceNameMapperCache.convertedNames, searchPathCache and sourceCache.

During this task, a lot of changes were suggested and were not prioritized due to small concerns. Those changes should be nice to have in the future and I'll go over the notes, collect and create a separate issue to have them investigated and added.

#183 - 12/18/2023 02:13 PM - Eric Faulhaber

Dănuț Filimon wrote:

> Database Configuration was updated with the necessary information for configuring the cache size of SourceNameMapperCache.convertedNames, searchPathCache and sourceCache.
>
> During this task, a lot of changes were suggested and were not prioritized due to small concerns. Those changes should be nice to have in the future and I'll go over the notes, collect and create a separate issue to have them investigated and added.

As part of this proposed effort, the cache size configuration probably needs to be generalized and moved outside of the persistence section of the directory. The containers probably should be inverted, such that the cache-size container encloses the persistence and other containers.

Likewise, the cache size configuration documentation should be moved to a more general-purpose area of the docs, and referenced/linked from those places that need it. It started as persistence-specific, but as we continue to define configurable caches and expand their use beyond the persistence area (like the source name mapper caches), we will need the documentation for them.

#184 - 12/19/2023 03:32 AM - Alexandru Lungu

I agree with Eric. Fine tuning cache-sizes is a common pattern when deploying. I would also like a mechanism that can report the cache hit/miss ratio and so some "suggestions" on the optimal cache sizes. By now, the work is quite manual using JMX. However, we need to consider activating it and retrieve some insights on the cache hit/miss ratio across **all** caches that are created using the CacheManager:

```
if (<cache instrumentation is available>)
   return InstrumentedCache(new LRUCache()); // proxy pattern (static is enough for performance sake)
else
   return LRUCache();
```

This can be added as a flag in the cache-size (or rather cache-config node), that will enclose persistence.

I think we have a task for that - Danut, please, search for it. Otherwise, create another task with this exact specification (refactor of directory cache config and add cache instrumentation). Assign it to yourself and mark as High.

#185 - 12/19/2023 03:39 AM - Dănuț Filimon

Alexandru, I remember you also mentioned it in #6815. But at that time you asked Constantin for feedback on the idea. Let me know if I should go ahead with #6815. Currently I am working on #7496.

#186 - 12/19/2023 10:13 AM - Greg Shah

Is 6649a a live branch?

#187 - 12/20/2023 02:06 AM - Dănuț Filimon

Greg Shah wrote:

> Is 6649a a live branch?

It should not be, all the work I've done was in 6649b.

#188 - 12/20/2023 12:18 PM - Greg Shah

6649a has been dead-archived.

#189 - 12/22/2023 08:04 PM - Constantin Asofiei

Created task branch 6649c from trunk rev 14905

6649c rev 14906 fixes a race condition in SourceNameMapper propath cache calculation - was found while testing a very large customer application. The scenario: you have multiple agents (not necessarily for the same appserver) using the same PROPATH; the per-propath cache build was not taking in consideration 'build it only once'.

#190 - 12/23/2023 02:34 PM - Constantin Asofiei

6649c was merged to trunk rev 14906 and archived.

#191 - 03/12/2024 05:45 PM - Greg Shah

- Related to Feature #6407: name_map.xml improvements added

Files

getInternalEntry.png	339 KB	01/24/2023	Constantin Asofiei
cache_internal_entries.diff	5.81 KB	05/04/2023	Hynek Cihlar