User Interface - Feature #7884

Standard 4GL methods used for login screen authentication to be exposed in sessionless helper classes

10/08/2023 01:07 AM - Galya B

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	10%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to User Interface - Feature #3931: single sign-on for virtual desktop...	**Closed**

History

#1 - 10/08/2023 01:24 AM - Galya B

- Subject changed from Expose 4GL authentication methods relying on DB in sessionless helper classes to 4GL methods used for login screen authentication to be exposed in sessionless helper classes

4GL methods used by customers to authenticate in-app user & pass on login screens need to be provided in FWD for use without session with the introduction of SSO (#3931), since the login screen is not part of the 4GL app any more.

Embedded clients for example use setuserid to authenticate the user & pass against the DB. The implementation in ConnectionManager.authenticate relies on methods (in DatabaseManager, Dialect, etc.) using session with the client and security context, that are not available at the time of login (before client process spawn). A helper class to expose the same validation, but hiding the specifics of establishing the DB connection and eliminating the need for session is needed.

SsoAuthenticator implementation will often rely on connection to the DB for credentials verification and session management. A standardized approach that doesn't depend on session, needs to be provided.

Ref from #3931-438 to #3931-440.

#2 - 10/08/2023 01:41 AM - Galya B

- Related to Feature #3931: single sign-on for virtual desktop mode added

#3 - 10/08/2023 01:47 AM - Galya B

- Subject changed from 4GL methods used for login screen authentication to be exposed in sessionless helper classes to Standard 4GL methods used for login screen authentication to be exposed in sessionless helper classes

#4 - 10/13/2023 01:07 AM - Galya B

Refs #3931-457:

Greg Shah wrote:

> Can we compile a list of authentication methods in OpenEdge 4GL that can be applicable?
>
> I know only of setuserid. Its implementation relies on DatabaseManager, ConnectionManager, Dialect and other classes using session and context.

- "Legacy" style authentication
  - This is based on database level authentication.
  - There are two ways to do it:
    - SETUSERID()
    - CONNECT statement that includes -U (userid) and -P (password) connection parameters is a kind of implicit SETUSERID() for that database.
- "Modern" approach
  - CLIENT-PRINCIPAL object in cooperation with some SECURITY-POLICY features. The CLIENT-PRINCIPAL is a session-level security token.
  - Two ways to do it:
    - Using an OpenEdge SSO facility. This has to be configured external to the session. Then the CLIENT-PRINCIPAL is setup using SECURITY-POLICY:SET-CLIENT() or the built-in function SET-DB-CLIENT().
    - DIY (do it yourself) using CLIENT-PRINCIPAL:SEAL().

FYI, we have partial support for CLIENT-PRINCIPAL but not for the SSO modes. See #3752, #3810, #4380 and #6423.

#5 - 11/15/2023 07:19 AM - Greg Shah

We may want to work part of this task so that the Hotel GUI SSO is properly implemented as an example.

#6 - 11/15/2023 07:21 AM - Galya B

I need a definition of 'properly'.

#7 - 11/15/2023 07:21 AM - Greg Shah

Galya wrote:

> Actually hotel gui works properly and exemplary now (tested with h2). The raised concern was that it should not depend directly on the DB config structure in directory, that may change. This requires the core FWD DB classes to work without context. Also it may be an improvement to directly replicate the legacy authentication methods without context to allow for a login page before starting a client. If there are customers who use the legacy authentication methods for a login page, then it should be considered.

Indeed, the use of setuserid is very common. The Hotel GUI SSO example would optimally use a context-less setuserid in this case.

#8 - 11/15/2023 07:23 AM - Greg Shah

> I need a definition of 'properly'.

Directly showing the replacement of the original 4GL authentication method, especially doing so without us having to provide a workaround that might be quite fragile.

#9 - 11/15/2023 07:23 AM - Galya B

Greg Shah wrote:

> Indeed, the use of setuserid is very common.

As far as I understand setuserid is a common DB authentication method. Is it used though for login screens / main app authentication?

#10 - 11/15/2023 07:25 AM - Galya B

Galya B wrote:

> Greg Shah wrote:
>
> > Indeed, the use of setuserid is very common.
>
> As far as I understand setuserid is a common DB authentication method. Is it used though for login screens / main app authentication?

What I'm trying to figure out here is if an actual app will need setuserid on the login screen or hotel gui needs to be changed.

#11 - 11/15/2023 07:28 AM - Greg Shah

> > Indeed, the use of setuserid is very common.

> As far as I understand setuserid is a common DB authentication method. Is it used though for login screens / main app authentication?

Yes, that is the most common use case.

#12 - 11/15/2023 07:36 AM - Galya B

OK, so then we agree to proceed with reworking the core FWD DB classes to work without context, so that the DB check can be done on the Jetty thread on login attempt?

#13 - 11/15/2023 07:41 AM - Greg Shah

No, I don't think that is a likely approach. Those classes are tightly coupled to context, to the legacy transaction/block properties and error handling. It makes little sense to rework them for this case. The likely approach is to implement an external form of setuserid that hides the direct database access.

#14 - 11/15/2023 07:43 AM - Galya B

Greg Shah wrote:

> No, I don't think that is a likely approach. Those classes are tightly coupled to context, to the legacy transaction/block properties and error handling. It makes little sense to rework them for this case. The likely approach is to implement an external form of setuserid that hides the direct database access.

It will hide the implementation details, which can be considered an improvement, but wouldn't it still be a duplication of logic and config reading, this time inside the framework?

#15 - 11/15/2023 08:01 AM - Greg Shah

> It will hide the implementation details, which can be considered an improvement,

This isn't just an improvement, it is essential. Without this, the duplication of the setuserid logic is in every customer application which uses SSO for this authentication method, which may be a very large number of customers. We absolutely don't want our internal implementation of setuserid duplicated hundreds of times (maybe badly) in customer code.

> but wouldn't it still be a duplication of logic and config reading, this time inside the framework?

Yes. That is a small price to pay for hiding the implementation details of setuserid. With this approach, we can make changes to the setuserid implementation and it is fully under our control.

And it is possible that there is some worker code that can be made common.

#16 - 11/23/2023 08:38 AM - Galya B

There is something tricky with exposing setuserid before the session has started. It can be used to validate the login credentials, but it actually needs to be executed a second time after the session is started to ensure the DB connection authentication. I don't know how this will stand with customers.

#17 - 11/23/2023 09:11 AM - Greg Shah

It is a good point. I think what we are looking for here is to expose the authentication check itself, without the part that establishes the database identity.

Today, we don't do the database identity part in FWD. This is a long standing issue that needs to be resolved but we won't be resolving it in this task.

Eric: Do we have a task for the database identity problem?

#18 - 11/24/2023 05:37 AM - Galya B

- Status changed from New to WIP

7884a r14840: MetadataSecurityOps.NoSessionHelper class added to support sso login screen authentication.

#19 - 11/24/2023 05:37 AM - Galya B

- % Done changed from 0 to 100

- Status changed from WIP to Review

#20 - 11/24/2023 05:49 AM - Galya B

- % Done changed from 100 to 10

Basically the authentication code presented in #3931-435 is moved from HotelGuiSsoAuthenticator to MetadataSecurityOps.NoSessionHelper (to be kept close to the original 4GL method implementation). I've tested with reworked hotel_gui authenticator.

#21 - 11/27/2023 08:24 AM - Greg Shah

Ovidiu: Please review. Consider if there is any common code that can be shared with the existing session-based setuserid.

#22 - 11/28/2023 03:03 PM - Ovidiu Maxiniuc

Review of 7884a, r14840.

The new code in MetadataSecurityOps.java is well written. Just a small observation: at line 406 in MetadataSecurityOps, SecurityOps._encode(String) can be used to avoid temporary BDT wrapping for encoded password.

Indeed, the setUserId() method of static class NoSessionHelper seems very similar to ConnectionManager.authenticate(). One difference is that the latter is more dialect-friendly. A second one is that ConnectionManager.authenticate() uses persistence.getSingleSQLResult API, while the new method/class uses a dedicated connection, created at a lower level using credentials it reads directly from directory.

The QUERY_SELECT_COUNT_MATCHING_CREDS is plain SQL but probably will also work in all cases. To note that ConnectionManager.authenticate() is called from MetadataSecurityOps.setUserId().

Can these method be merged? I think the answer is positive, but common code is not that big. The largest parts of the methods are different because they work at different levels.

#23 - 11/29/2023 02:02 AM - Galya B

Ovidiu Maxiniuc wrote:

> Just a small observation: at line 406 in MetadataSecurityOps, SecurityOps._encode(String) can be used to avoid temporary BDT wrapping for encoded password.

Up in r14841.

> Can these method be merged? I think the answer is positive, but common code is not that big. The largest parts of the methods are different because they work at different levels.

It may seem at first that some parts can be reused, but Dialect and Persistence both rely on context. I've tried to use them originally. If references to both classes get removed from authenticate, there is nothing left. Even the select query can't be reused.

#24 - 12/01/2023 04:07 AM - Galya B

Is the "conversion" always SQL_MODE_DEFAULT. I don't see a case where NsConfig.setConversion is used.

#25 - 12/01/2023 10:08 AM - Greg Shah

> > > Indeed, the use of setuserid is very common.
>
> > As far as I understand setuserid is a common DB authentication method. Is it used though for login screens / main app authentication?
>
> What I'm trying to figure out here is if an actual app will need setuserid on the login screen or hotel gui needs to be changed.

Yes, setuserid() is needed on a login screen for a real application AND we want to migrate Hotel GUI to use the same approach.

#26 - 12/06/2023 03:40 AM - Galya B

Galya B wrote:

> Is the "conversion" always SQL_MODE_DEFAULT. I don't see a case where NsConfig.setConversion is used.

MetadataSecurityOps:

```
**     CA  20230607 The leading underscore for _User metadata table fields depends on the minimal conversion
**                  mode, so rely on this namespace attribute when resolving the field names.
```

```
   public static String getMetaFieldUserid(String ldbName)
   {
      // TODO:  this code is to be removed when we implement all metadata tables to use the underscore prefix
      // instead of the meta_ prefix, and the meta fields will use the underscore always.
      String fieldName = metaFieldUserId.computeIfAbsent(ldbName.toLowerCase(), (k) ->
      {
         int cvt = MatchPhraseConstants.SQL_MODE_DEFAULT;
         try
         {
            cvt = Configuration.getSchemaConfig().getSqlConversion(ldbName);
         }
         catch (Exception e)
         {
            // ignore
         }

         return cvt == MatchPhraseConstants.SQL_MODE_DEFAULT ? "userid" : "_userid";
      });

      return fieldName;
   }
```

Constantin, is the "conversion" always SQL_MODE_DEFAULT? I don't see it set in the project.

#27 - 12/06/2023 05:23 AM - Constantin Asofiei

Galya, yes, SQL_MODE_DEFAULT is the default if otherwise not specified in p2j.cfg.xml, as a conversion="minimal" attribute for the namespace node.

We can't hard-code the SQL table name.

#28 - 12/06/2023 07:31 AM - Galya B

7884a r14842 ready for review. The dialect and conversion specifics now shared between MetadataSecurityOps.NoSessionHelper and ConnectionManager with introducing a query building method reused by both classes. I've tested with hotel_gui H2 and Postgres.

#30 - 12/06/2023 04:20 PM - Greg Shah

Code Review Task Branch 7884a Revisions 14840 through 14842

The overall approach is reasonable. Ovidiu and Eric will have to assess if the persistence support meets are requirements and/or if there is any other common code that can be leveraged here.

My primary concern is that we don't support the ldbname parameter variant of setuserid(). I think that should be supported.

#31 - 12/06/2023 09:23 PM - Ovidiu Maxiniuc

I reviewed the changes in 7884a.

I am OK with the changes. My concerns were for:

- duplicate code;
- hardcoded table/column names
- dialect awareness

Now all of them seemed fixed.

For the second part. Provided the ldbname, the database configuration from registry can be read. Now the first connected/loaded database is used.

#32 - 12/07/2023 02:22 AM - Galya B

Ovidiu Maxiniuc wrote:

> For the second part. Provided the ldbname, the database configuration from registry can be read. Now the first connected/loaded database is used.

What registry works without context?

#33 - 12/07/2023 02:26 AM - Galya B

Greg Shah wrote:

> My primary concern is that we don't support the ldbname parameter variant of setuserid(). I think that should be supported.

Where is ldbname parameter coming from?

#34 - 12/07/2023 06:45 AM - Greg Shah

It will optionally be present in the original 4GL code. There are 2 forms:

- setuserid(userid, password)
- setuserid(userid, password, ldbname)

When the developer replaces the original 4GL login code with a hand-crafted SSO equivalent, they will naturally need to provide that value as well. Perhaps it is hard coded. Or maybe they configure it somewhere in their application. Or maybe they randomly assign one... Regardless, they will have to come up with the value or accept the default database. We just need to ensure there is an option for them to pass this.

#35 - 12/07/2023 07:06 AM - Galya B

r14843: setuserid(userid, password, ldbname).

#36 - 12/07/2023 07:21 AM - Greg Shah

Eric: Is org.apache.commons.lang3.tuple.* a supported/approved library for use in persistence? I see it used there but I have no recollection of this being explicitly discussed/approved.

#37 - 12/07/2023 07:27 AM - Greg Shah

Code Review Task Branch 7884a Revision 14843

1. I think the logic for obtaining the default database name (DatabaseManager.loadedOnStartup().get(0)) should not be hard coded here. It should be in some kind of common code.

2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.

3. I don't think it is a good idea to implement long lived connections to the database here, nor a completely new way to manage them. It is a hidden "leak" of connections that can't be managed.

Eric: Please review.

#38 - 12/07/2023 08:15 AM - Galya B

Greg Shah wrote:

> Code Review Task Branch 7884a Revision 14843
>
> 1. I think the logic for obtaining the default database name (DatabaseManager.loadedOnStartup().get(0)) should not be hard coded here. It should be in some kind of common code.
>
> 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.
>
> 3. I don't think it is a good idea to implement long lived connections to the database here, nor a completely new way to manage them. It is a hidden "leak" of connections that can't be managed.

Greg Shah wrote:

> We may want to work part of this task so that the Hotel GUI SSO is properly implemented as an example.

Galya B wrote:

> I need a definition of 'properly'.

Greg Shah wrote:

> Directly showing the replacement of the original 4GL authentication method, especially doing so without us having to provide a workaround that might be quite fragile.

The scope is changing, which is fine, if we know where it's going. I've asked a while ago if it's about reworking the core DB logic into sessionless one or just patching hotel_gui, but I'm still not sure which one it is.

#39 - 12/07/2023 08:18 AM - Galya B

Greg Shah wrote:

> 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.

How do we do "common code" with .rules (import_util.rules, where directory db configs are read)?

#40 - 12/07/2023 08:24 AM - Galya B

Greg Shah wrote:

> 3. I don't think it is a good idea to implement long lived connections to the database here, nor a completely new way to manage them. It is a hidden "leak" of connections that can't be managed.

1. It's expected to be only one connection.
2. Is it better to create a DB connection on each login attempt in a live system?
3. The sessionless helper can't share connection management with sessions.
4. There is nothing hidden. The connection is cleaned up on JVM shutdown.

#41 - 12/07/2023 08:40 AM - Greg Shah

Galya B wrote:

> Greg Shah wrote:
>
> > Code Review Task Branch 7884a Revision 14843
> >
> > 1. I think the logic for obtaining the default database name (DatabaseManager.loadedOnStartup().get(0)) should not be hard coded here. It should be in some kind of common code.
> >
> > 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.
> >
> > 3. I don't think it is a good idea to implement long lived connections to the database here, nor a completely new way to manage them. It is a hidden "leak" of connections that can't be managed.
>
> Greg Shah wrote:
>
> > We may want to work part of this task so that the Hotel GUI SSO is properly implemented as an example.
>
> Galya B wrote:

> > I need a definition of 'properly'.
>
> Greg Shah wrote:
>
> > Directly showing the replacement of the original 4GL authentication method, especially doing so without us having to provide a workaround that might be quite fragile.
>
> The scope is changing, which is fine, if we know where it's going. I've asked a while ago if it's about reworking the core DB logic into sessionless one or just patching hotel_gui, but I'm still not sure which one it is.

I thought I had answered this in #7884-25. We are reworking the code DB logic into sessionless code that is in FWD instead of Hotel GUI. As part of this, I expect Hotel GUI to be modified to use the new code.

#42 - 12/07/2023 08:49 AM - Greg Shah

> > 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.
>
> How do we do "common code" with .rules (import_util.rules, where directory db configs are read)?

Are you sure that is coming from the directory? Anyway, the directory database cfgs are there for runtime usage. Any usage from import is a side usage.

Eric/Ovidiu can better report on the mechanism, but it is probably buried deep in the guts of DatabaseManage.

#43 - 12/07/2023 08:50 AM - Galya B

Greg Shah wrote:

> I thought I had answered this in #7884-25. We are reworking the code DB logic into sessionless code that is in FWD instead of Hotel GUI. As part of this, I expect Hotel GUI to be modified to use the new code.

Not really. It is currently doing what you wanted in #7884-25, without incorporating resolved DB configs from context and sharing db connection management with sessions.

#44 - 12/07/2023 08:52 AM - Greg Shah

> 1. It's expected to be only one connection.

Per database instance. This may be many connections to a given cluster.

> 2. Is it better to create a DB connection on each login attempt in a live system?
> 3. The sessionless helper can't share connection management with sessions.

These are questions for Eric.

> 4. There is nothing hidden. The connection is cleaned up on JVM shutdown.

I mean it is a non-obvious place to find a custom implementation of a cache for connections. Future editors/readers of the code might not find this easily.

#45 - 12/07/2023 08:54 AM - Galya B

Greg Shah wrote:

> > 4. There is nothing hidden. The connection is cleaned up on JVM shutdown.
>
> I mean it is a non-obvious place to find a custom implementation of a cache for connections. Future editors/readers of the code might not find this easily.

It's sessionless. The login screen is available for the whole lifespan of the app. Where else are the readers going to look for it?

#46 - 12/07/2023 08:55 AM - Galya B

> 2. Is it better to create a DB connection on each login attempt in a live system?

This was a retorical question.

#47 - 12/07/2023 08:58 AM - Greg Shah

> > I thought I had answered this in #7884-25. We are reworking the code DB logic into sessionless code that is in FWD instead of Hotel GUI. As part of this, I expect Hotel GUI to be modified to use the new code.
>
> Not really. It is currently doing what you wanted in #7884-25, without incorporating resolved DB configs from context and sharing db connection management with sessions.

Are you asking if we are going to rework the code FWD persistence classes to be sessionless? The answer is still no, as discussed in #7884-13.

I agree that the current approach in 7884a is functionally close to what we need. The code reviews are just highlighting my concerns with making the implementation less fragile. Wherever we can push the logic back into common code used in persistence, then we will more likely survive future edits without breakage.

If the scope is not clear, help me understand what you need to know.

#48 - 12/07/2023 09:00 AM - Greg Shah

Galya B wrote:

> Greg Shah wrote:
>
> > > 4. There is nothing hidden. The connection is cleaned up on JVM shutdown.
> >
> > I mean it is a non-obvious place to find a custom implementation of a cache for connections. Future editors/readers of the code might not find this easily.
>
> It's sessionless. The login screen is available for the whole lifespan of the app. Where else are the readers going to look for it?

A future editor/reader of persistence code is not necessarily thinking about SSO or sessionless connections for login. Such a person may not even know that we have such things. To them it would be non-obvious.

#49 - 12/07/2023 09:01 AM - Galya B

Greg Shah wrote:

> Galya B wrote:
>
> > Greg Shah wrote:
> >
> > > > 4. There is nothing hidden. The connection is cleaned up on JVM shutdown.
> > >
> > > I mean it is a non-obvious place to find a custom implementation of a cache for connections. Future editors/readers of the code might not find this easily.
> >
> > It's sessionless. The login screen is available for the whole lifespan of the app. Where else are the readers going to look for it?
>
> A future editor/reader of persistence code is not necessarily thinking about SSO or sessionless connections for login. Such a person may not even know that we have such things. To them it would be non-obvious.

Such reader will need to learn to read the names of the classes and the javadoc, where it says:

```
   /**
    * Helper class to execute setuserid without client session and security context. Can be used by login
    * screens with SSO enabled, that is before the client is spawned.
    */
   public static class NoSessionHelper
```

#50 - 12/07/2023 09:10 AM - Galya B

Greg Shah wrote:

> > 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.
>
> How do we do "common code" with .rules (import_util.rules, where directory db configs are read)?

> Are you sure that is coming from the directory?

import_util.rules:

```
      <function name="buildDbImportProperties">
         [..]

         <rule>dbImportParams.put("dialect", dialectClass)</rule>
         <rule>dbImportParams.put("database_name", dbName)</rule>
         <rule>dbImportParams.put("connection.driver_class", driverClass)</rule>
         <rule>dbImportParams.put("connection.url", url)</rule>
         <rule>dbImportParams.put("connection.username", uid)</rule>
         <rule>dbImportParams.put("connection.password", pw)</rule>

         <rule>dbImportParams.put("show_sql", false)</rule>
         <rule>dbImportParams.put("jdbc.batch_size", 200)</rule>
         <rule>dbImportParams.put("c3p0.max_size", 24)</rule>
         <rule>dbImportParams.put("c3p0.min_size", 8)</rule>
         <rule>dbImportParams.put("c3p0.acquire_increment", 8)</rule>
         <rule>dbImportParams.put("c3p0.max_statements", 100)</rule>
         <rule>dbImportParams.put("cache.use_second_level_cache", false)</rule>
      </function>
```

> Anyway, the directory database cfgs are there for runtime usage. Any usage from import is a side usage.
>
> Eric/Ovidiu can better report on the mechanism, but it is probably buried deep in the guts of DatabaseManage.

This is the only place where driver_class is mentioned.

#51 - 12/07/2023 09:15 AM - Greg Shah

> Such reader will need to learn to read the names of the classes and the javadoc, where it says:

My comment stands. This javadoc is in the util package and it is not an obvious place for persistence readers/editors to look. If they don't know to look there they may not find it.

#52 - 12/07/2023 09:17 AM - Galya B

Greg Shah wrote:

> > > I thought I had answered this in #7884-25. We are reworking the code DB logic into sessionless code that is in FWD instead of Hotel GUI. As part of this, I expect Hotel GUI to be modified to use the new code.
> >
> > Not really. It is currently doing what you wanted in #7884-25, without incorporating resolved DB configs from context and sharing db connection management with sessions.
>
> Are you asking if we are going to rework the code FWD persistence classes to be sessionless? The answer is still no, as discussed in #7884-13.

Eric, the task is on hold, waiting for clarifications.

#53 - 12/07/2023 09:20 AM - Greg Shah

> > Anyway, the directory database cfgs are there for runtime usage. Any usage from import is a side usage.
> >
> > Eric/Ovidiu can better report on the mechanism, but it is probably buried deep in the guts of DatabaseManage.
>
> This is the only place where driver_class is mentioned.

When the FWD server starts, we don't make our database connections using import_util.rules. 99.999999% of database connections are at runtime from the FWD server and have nothing to do with import. The proper code for this is in the persistence package somewhere.

#54 - 12/07/2023 09:45 AM - Greg Shah

Supplement to the code review: the ldbname parameter must be processed case insensitively (e.g. storing/retrieving the connection using that name as a key is currently broken as compared with how the 4GL code would work).

#55 - 12/07/2023 09:50 AM - Galya B

Greg Shah wrote:

> Supplement to the code review: the ldbname parameter must be processed case insensitively (e.g. storing/retrieving the connection using that name as a key is currently broken as compared with how the 4GL code would work).

This started as a side task to SSO. I've read no documentation. Noone explained anything about how FWD works with databases. Requirements come in waves and each requirement implemented "breaks" the next "thing". This is an awesome learning experience, the curve is bumpy.

#56 - 12/07/2023 09:51 AM - Galya B

We're doing a default authentication for SSO login, where the customer should not know anything about the details, but suddenly even clusters come into the picture.

#57 - 12/07/2023 09:55 AM - Galya B

I've been obedient adding the overloaded method with the logical name, but where is the java SsoAuthenticator getting this logical DB name from, while keeping ignorant about the implementation details???

#58 - 12/07/2023 10:06 AM - Greg Shah

> I've been obedient adding the overloaded method with the logical name, but where is the java SsoAuthenticator getting this logical DB name from, while keeping ignorant about the implementation details???

The calling code will be written by a developer that is hand migrating existing login logic. That 4GL code will have some mechanism for determining the ldbname. We have no control over that but we do know that the developer would have the expectation that this setuserid would operate in a compatible manner with the 4GL version. All such 4GL builtin functions handle character data using the assumption that it is compared on a case-insensitive basis. Thus we should implement the same idea.

#59 - 12/07/2023 10:42 AM - Eric Faulhaber

Greg Shah wrote:

> Eric: Is org.apache.commons.lang3.tuple.* a supported/approved library for use in persistence? I see it used there but I have no recollection of this being explicitly discussed/approved.

It was not explicitly approved/discussed. I think Ovidiu brought this in during the massive wave of changes around the 4011e (i.e., remove/replace Hibernate) days. We had bigger things to get working, so I did not make a fuss.

#60 - 12/14/2023 02:13 AM - Galya B

Waiting for feedback from Eric and Ovidiu #7884-32, #7884-42 and #7884-44.

#61 - 01/04/2024 03:01 PM - Ovidiu Maxiniuc

In #7884-32 Galya B wrote:

> Ovidiu Maxiniuc wrote:
>
> > For the second part. Provided the ldbname, the database configuration from registry can be read. Now the first connected/loaded database is used.
>
> What registry works without context?

Sorry, it seems that I assumed that you use the database configuration of the first database, it was obtained from the registry. But it looks like this is not the case. The access to registry is necessary for obtaining this information, but since there is no context at the moment, this cannot be reached. It's a chicken and the egg problem.

To solve this I think a **temporary** anonymous connection/context is necessary, but this is not a simple task if the database contains at least a record in _User table (which forces the true authentication in 4GL). We need to discuss whether this is a viable solution.

In #7884-42 Greg Shah wrote:

> > > 2. The logic in connectToDb() which reads configuration from the directory is also something that should not be hard coded here. It should be in common code.
> >
> > How do we do "common code" with .rules (import_util.rules, where directory db configs are read)?
>
> Are you sure that is coming from the directory? Anyway, the directory database cfgs are there for runtime usage. Any usage from import is a side usage.
>
> Eric/Ovidiu can better report on the mechanism, but it is probably buried deep in the guts of DatabaseManager.

The import_util.rules exposes a set of functions for detection of some connection parameters. One of these is the connection url, if it is not already provided. However, this function is only used in import process, where the url is provided by the calling script. The other places it is called from are momentarily dead code (they still attempt to use Hibernate) and the result is incorrect (at least because the connection port is at most, guessed).

The DatabaseManager has a private static method getLocalConfiguration() which can provide you with the information you need (see returnedValue.getOrmSettings().config.getString("connection.url", null), for example).

In #7884-44 Greg Shah wrote:

> 2. Is it better to create a DB connection on each login attempt in a live system?

I am not sure a dedicated connection is necessary. Please try the following: use PersistenceFactory.getInstance(database) to obtain a Persistence object which comes with a configured connection and the dialect. This should create the instance on first access and reuse on subsequent attempts (wrong credentials). If the user is accepted, I expect the connection to be reused on database accesses on the user context.
Eric, if the solution works as expected is it OK to use it here? The Persistence is an object of a higher API and I feel this might not be the proper way to use it.

> 3. The sessionless helper can't share connection management with sessions.

My response to previous question tries to do exactly the opposite: use a connection which latter can be used by the user session. OTOH, a true independent/isolated helper should create a dedicated db connection and use it for its own needs. However, each user might want to authenticate to different database, so a set of dedicated connections should be maintained.

#62 - 01/04/2024 03:27 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Greg Shah wrote:
>
> > Eric: Is org.apache.commons.lang3.tuple.* a supported/approved library for use in persistence? I see it used there but I have no recollection of this being explicitly discussed/approved.
>
> It was not explicitly approved/discussed. I think Ovidiu brought this in during the massive wave of changes around the 4011e (i.e., remove/replace Hibernate) days. We had bigger things to get working, so I did not make a fuss.

Indeed, I was in need of a container to store a couple of objects together. While a simple array[2] would have been enough, I saw the Pair class already been used at that time and considering it a more elegant way, I decided to use it, assuming it was already approved. The usage escalated meanwhile.
At any rate, I think Pair and StringUtils are the only class from that package/library and their replacements can be easily written as part of FWD utilities. This would simplify a bit the project dependences. With a well chosen naming, the only changes in code would be dropping of the import statements.

#63 - 01/04/2024 03:56 PM - Ovidiu Maxiniuc

I had a look over 7884a/14843. I see it already uses some of the ideas I wrote above. I have only a few notes:

- I like the extra care of closing the optional database connections used during the NoSessionHelper;
- although the DB_NAME_CONNECTION_QUERY_TRIPLES is private static final, is it not truly immutable (as a scalar field with those attributes would be), so I think the name in all CAPS is not well fitted. Also, I would prefer a shorter name, like openConnections? Greg, please let me know if you agree;
- at line 23: if there are no databases with load_at_startup attribute set to true in directory, a NPE will be thrown in get(0).

#64 - 01/05/2024 02:39 AM - Galya B

We've started discussing how to access the DB configs in #7572-134 and I mentioned an issue noone paid attention to, that can be illustrated by the method mentioned by Eric:

Dialect.java:

```
public static Dialect getDialect(Database database) // this method apparently seems broken
{
  [..]
}
```

The method is not broken, but Database can't be found in the HashMap because of:

Database.java:

```
   /**
    * Calculate a hash code for this object.
    *
    * @return  Hash code.
    */
   @Override
   public int hashCode()
   {
      if (hashCode == null)
      {
         int result = 17;
         result = 37 * result + name.toUpperCase().hashCode();
         if (socketAddress != null)
         {
            result = 37 * result + socketAddress.hashCode();
         }

         result = 37 * result + type.hashCode();

         hashCode = result;
      }

      return hashCode;
   }

   /**
    * Test this object for equality (equivalence) with the given object.
    *
    * @param   o
    *          Object to test.
    *
    * @return  &lt;code&gt;true&lt;/code&gt; if this object is the same instance as the
    *          given object, or it is considered equivalent to the given
    *          object.  &lt;code&gt;Database&lt;/code&gt; instances are equivalent if
    *          they contain the same database name and P2J server address.
    */
   @Override
   public boolean equals(Object o)
   {
      if (this == o)
      {
         // Same instance.
         return true;
      }

      if (!(o instanceof Database))
      {
         // Impossible to be equivalent.
         return false;
      }

      Database that = (Database) o;

      // This comparison assumes that physical database names have been
      // canonicalized prior to instantiation of these Database instances.
      if (!this.name.equalsIgnoreCase(that.name))
      {
         return false;
      }

      if (!this.type.equals(that.type))
      {
         return false;
      }

      // This comparison will fail if different hostnames actually would
      // resolve to the same address.
      InetSocketAddress addr1 = this.socketAddress;
      InetSocketAddress addr2 = that.socketAddress;
      boolean sameAddr = (addr1 == null || addr2 == null
                          ? addr1 == addr2         // both null
                          : addr1.equals(addr2));  // equivalent

      return sameAddr;
   }
```

How am I supposed to know the socketAddress or type before fetching the DB configs for the first time, so that I can build the DB object?

#65 - 01/05/2024 02:42 AM - Galya B

Ovidiu Maxiniuc wrote:

> At any rate, I think Pair and StringUtils are the only class from that package/library and their replacements can be easily written as part of FWD utilities. This would simplify a bit the project dependences. With a well chosen naming, the only changes in code would be dropping of the import statements.

Greg, do I take care of this and remove the lib?

#66 - 01/05/2024 08:05 AM - Greg Shah

Galya B wrote:

> Ovidiu Maxiniuc wrote:
>
> > At any rate, I think Pair and StringUtils are the only class from that package/library and their replacements can be easily written as part of FWD utilities. This would simplify a bit the project dependences. With a well chosen naming, the only changes in code would be dropping of the import statements.
>
> Greg, do I take care of this and remove the lib?

No, let's not add extra work now.

#67 - 01/11/2024 08:23 AM - Galya B

Eric, Ovidiu, in case you've missed #7884-64, it still waits for your opinion. You both asked me to use Database object to find the configuration for the current database, but I need to know the the socketAddress to construct an object that can match the one in the HashMap. If I need to raise a more specific question, it would be: Do we expect to find several DBs with the same name and different socket in the maps? Do you know why socketAddress has been added to hashCode and equals?

#68 - 01/11/2024 10:46 AM - Eric Faulhaber

For most use cases of the Database object, the socketAddress will be null. However, FWD supports a mode where multiple servers have their own databases with a matching schema but different data, and server-to-server connections to those remote databases are allowed. For these remote connection cases, socketAddress will be non-null in a Database object. Within the local FWD server making these remote connections, these different Database objects differentiate remote instances of that database from each other and from the local instance.

Each FWD server is considered to be "authoritative" for its own local database instance, and when a server connects to a remote instance, it must first fetch the connection information from the remote/authoritative FWD server. It then makes a direct connection to the remote database using the connection information received from the authoritative server, and it must route all services for that remote database through a RemotePersistence object, rather than through a local Persistence object. This ensures locking and dirty database requests are managed by the authoritative server, since there can be only one server which manages these services for that database, whether connections to it are made locally or remotely. The socketAddress being part of the Database object's hashCode and equals computations allows PersistenceFactory to return the correct [Remote]Persistence object from its cache.

In terms of retrieving Dialect information, the Dialect.knownDialects should be populated with keys where the socketAddress is null, since these would be needed for the FWD server to look up the dialects of the databases for which it is authoritative. The Dialect.getDialect(Database) method maybe needs to use the Database(Database, Type) c'tor to null out any socket address which may be present when looking up a Dialect.

So, I think the short answer to your question is you do not need to know the socket address to look up the dialect; it should be null for this lookup.

#69 - 01/12/2024 03:02 AM - Galya B

Eric Faulhaber wrote:

> In terms of retrieving Dialect information, the Dialect.knownDialects should be populated with keys where the socketAddress is null, since these would be needed for the FWD server to look up the dialects of the databases for which it is authoritative.

Thanks for the explanation. I'm asked in #7884-61 to use PersistenceFactory.getInstance(Database). Does the same apply to PersistenceFactory.cache?

Another question to everyone: Does the cache contain only already loaded databases, instead of all configurations from directory? Exploring the Persistence object, it looks like the DB is already connected. Can we have the certainty that the authentication DB, selected by the customer writing the SsoAuthenticator implementation, is already present in cache and connected, when no client session has been created?

#70 - 01/12/2024 12:43 PM - Eric Faulhaber

Galya B wrote:

> Eric Faulhaber wrote:
>
> > In terms of retrieving Dialect information, the Dialect.knownDialects should be populated with keys where the socketAddress is null, since these would be needed for the FWD server to look up the dialects of the databases for which it is authoritative.
>
> Thanks for the explanation. I'm asked in #7884-61 to use PersistenceFactory.getInstance(Database). Does the same apply to PersistenceFactory.cache?

I'm not sure what you are asking. Since PersistenceFactory.cache is private, it is not meant to be accessed directly from outside code.

> Another question to everyone: Does the cache contain only already loaded databases, instead of all configurations from directory? Exploring the Persistence object, it looks like the DB is already connected. Can we have the certainty that the authentication DB, selected by the customer writing the SsoAuthenticator implementation, is already present in cache and connected, when no client session has been created?

If you mean the PersistenceFactory cache, I believe it is populated lazily, as databases are first connected. If you mean the Dialect.knownDialects cache, this is populated for all database configurations in the directory when Persistence.initialize is first invoked during server bootstrap.

#71 - 01/15/2024 02:44 AM - Galya B

Ovidiu Maxiniuc wrote:

> Please try the following: use PersistenceFactory.getInstance(database) to obtain a Persistence object which comes with a configured connection and the dialect.

Eric Faulhaber wrote:

> Galya B wrote:
>
> > Another question to everyone: Does the cache contain only already loaded databases, instead of all configurations from directory? Exploring the Persistence object, it looks like the DB is already connected. Can we have the certainty that the authentication DB, selected by the customer writing the SsoAuthenticator implementation, is already present in cache and connected, when no client session has been created?
>
> If you mean the PersistenceFactory cache, I believe it is populated lazily, as databases are first connected.

PersistenceFactory:

```
   public static Persistence getInstance(Database database)
   {
      [..]

      Persistence instance;
      synchronized (cache)
      {
         instance = cache.get(database);

         [..]
      }

      [..]

      return instance;
   }
```

Ovidiu, what do you think?

#72 - 01/18/2024 11:22 AM - Ovidiu Maxiniuc

Sorry, I do not know what you mean. That is, indeed, the code from PersistenceFactory.getInstance(Database database). It will return an existing object if one is cached or, otherwise, will use dialect information from DatabaseManager to construct a proper object for you. In second case, the advantage is that the new object can later be used by business code inside the current context.
The only condition here is that the DatabaseManager.dialects to contain the dialect for your database, meaning the database to be initialized at server startup. If you are using a non auto-connecting database, the call will fail.

#73 - 01/19/2024 02:47 AM - Galya B

Ovidiu Maxiniuc wrote:

> The only condition here is that the DatabaseManager.dialects to contain the dialect for your database, meaning the database to be initialized at server startup. If you are using a non auto-connecting database, the call will fail.

I was trying to figure out exactly this: What implicit constrains we're putting in place with the use of PersistenceFactory. Thanks for the clarification.

So only databases with load_at_startup will be eligible for executing the authentication query.

#74 - 01/19/2024 02:58 AM - Galya B

The requested changes are in r14844 and they don't work, because there is no SecurityContext on the Jetty thread:

```
24/01/19 09:53:54.523+0200 |  SEVERE | com.goldencode.p2j.util.MetadataSecurityOps$NoSessionHelper | ThreadNam
e:qtp729401599-62 | Can't authenticate the user hotel against the db.
java.lang.NullPointerException
    at com.goldencode.p2j.security.SecurityManager.isServerAccount(SecurityManager.java:4213)
    at com.goldencode.p2j.util.TransactionManager._isHeadless(TransactionManager.java:3069)
    at com.goldencode.p2j.util.TransactionManager$ContextContainer.obtain(TransactionManager.java:11148)
    at com.goldencode.p2j.util.TransactionManager.getTransactionHelper(TransactionManager.java:1745)
    at com.goldencode.p2j.util.ProcedureManager$ContextContainer.obtain(ProcedureManager.java:6372)
    at com.goldencode.p2j.util.ProcedureManager.getProcedureHelper(ProcedureManager.java:1111)
    at com.goldencode.p2j.persist.BufferManager.<init>(BufferManager.java:704)
    at com.goldencode.p2j.persist.BufferManager.<init>(BufferManager.java:612)
    at com.goldencode.p2j.persist.BufferManager$1.initialValue(BufferManager.java:624)
    at com.goldencode.p2j.persist.BufferManager$1.initialValue(BufferManager.java:622)
    at com.goldencode.p2j.security.ContextLocal.getImpl(ContextLocal.java:541)
    at com.goldencode.p2j.security.ContextLocal.get(ContextLocal.java:453)
    at com.goldencode.p2j.persist.BufferManager.get(BufferManager.java:773)
    at com.goldencode.p2j.persist.Persistence$Context.<init>(Persistence.java:4549)
    at com.goldencode.p2j.persist.Persistence$Context.<init>(Persistence.java:4543)
    at com.goldencode.p2j.persist.Persistence$1.initialValue(Persistence.java:760)
    at com.goldencode.p2j.persist.Persistence$1.initialValue(Persistence.java:757)
    at com.goldencode.p2j.security.ContextLocal.getImpl(ContextLocal.java:541)
    at com.goldencode.p2j.security.ContextLocal.get(ContextLocal.java:453)
    at com.goldencode.p2j.persist.Persistence.beginTransaction(Persistence.java:3571)
    at com.goldencode.p2j.persist.Persistence.beginTransaction(Persistence.java:3312)
    at com.goldencode.p2j.util.MetadataSecurityOps$NoSessionHelper.setUserId(MetadataSecurityOps.java:435)
    at com.goldencode.hotel.HotelGuiSsoAuthenticator.authenticate(HotelGuiSsoAuthenticator.java:136)
    at com.goldencode.p2j.main.WebDriverHandler.spawnWorker(WebDriverHandler.java:394)
    at com.goldencode.p2j.main.WebDriverHandler.handleStartClientRequest(WebDriverHandler.java:319)
    at com.goldencode.p2j.main.EmbeddedWebHandler.handle(EmbeddedWebHandler.java:203)
    at org.eclipse.jetty.server.handler.HandlerCollection.handle(HandlerCollection.java:146)
    at org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:127)
    at org.eclipse.jetty.server.Server.handle(Server.java:500)
    at org.eclipse.jetty.server.HttpChannel.lambda$handle$1(HttpChannel.java:386)
    at org.eclipse.jetty.server.HttpChannel.dispatch(HttpChannel.java:562)
    at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:378)
    at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:270)
    at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:311)
    at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:103)
    at org.eclipse.jetty.io.ssl.SslConnection$DecryptedEndPoint.onFillable(SslConnection.java:543)
    at org.eclipse.jetty.io.ssl.SslConnection.onFillable(SslConnection.java:398)
    at org.eclipse.jetty.io.ssl.SslConnection$2.succeeded(SslConnection.java:161)
    at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:103)
    at org.eclipse.jetty.io.ChannelEndPoint$2.run(ChannelEndPoint.java:117)
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.runTask(EatWhatYouKill.java:336)
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.doProduce(EatWhatYouKill.java:313)
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.tryProduce(EatWhatYouKill.java:171)
    at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.run(EatWhatYouKill.java:129)
    at org.eclipse.jetty.util.thread.ReservedThreadExecutor$ReservedThread.run(ReservedThreadExecutor.java:388
)
    at org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:806)
    at org.eclipse.jetty.util.thread.QueuedThreadPool$Runner.run(QueuedThreadPool.java:938)
    at java.lang.Thread.run(Thread.java:750)
```

#75 - 01/19/2024 03:33 AM - Galya B

Ovidiu Maxiniuc wrote:

- at line 23: if there are no databases with load_at_startup attribute set to true in directory, a NPE will be thrown in get(0).

Just a reminder that this was considered issue initially, now it's still the case.