# Database - Support #3871

## determine how to change codepages/locales during import

01/11/2019 12:45 PM - Greg Shah

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

**Description**

**Related issues:**

| | |
|---|---|
| Related to Runtime Infrastructure - Support #4549: reduce/eliminate installat... | **New** |
| Related to Database - Feature #4723: make it significantly easier to run data... | **New** |
| Related to Database - Bug #5085: uppercasing and string comparisons are incor... | **New** |

### History

**#1 - 01/11/2019 12:50 PM - Greg Shah**

I am placing previous email discussions here.

From Greg:

> How much development is needed for FWD to support converting the database codepage during import? We have a customer database instance which uses 8859-1 and they want that database to be imported as UTF-8.

From Eric:

> Not sure exactly what we will have to do for this. PostgreSQL is UTF-8 by default, IIRC. We implemented a custom locale for use with 8859-1 because Progress seemed to collate English strings "in its own special way". How does Unicode collation work with particular locales, such as German, French, Spanish or Dutch?
>
> As I recall, we never came up with a solution for the custom locale problem with SQL Server on Windows. PostgreSQL will have the same problem, if Progress is again non-standard with its collation.

From Ovidiu:

> We do not have a perfect match for Progress collation in SQL Server. I did some investigations but the same stands for PostgreSQL. I was hoping that, because PostgreSQL evolved on Linux platform, on Windows it would be more flexible in this regard, allowing the use of customized collation and maybe accept our en_US@p2j_basic locale the same way it does on Linux. I found in the installation directory some LC_MESSAGES resources, but they only contain message/errors localized messages. Nothing related to collations. In fact, if you look at the output of the initdb on windows ( https://proj.goldencode.com/projects/p2j/wiki/Database_Server_Setup_for_PostgreSQL_on_Windows#Creating-a-FWD-Specific-Database-Cluster), you can see the following line:
>
> creating collations ... not supported on this platform
>
> What this means is that the collations are done using the rules of the selected locale - I think, in that case "English_United States.1252".
>
> However, this is not mandatory a bad news. I agree that, for some edge-cases, FWD will not sort some strings the exact way OE does, and this is contrary to our FWD paradigm. But, OTOH, selecting the proper locale, the user will actually get the results on screen the way (s)he naturally expects.
>
> Regarding the import as UTF-8. Do we need special care for this? Don't we keep character data as Java Strings in memory (ie UTF-16)? So they have to be already free of the original CP. The conversion to a UTF8 database storage should be done automatically, by DB driver. Do I oversimplify this?

#### #2 - 01/11/2019 12:54 PM - Greg Shah

> Regarding the import as UTF-8. Do we need special care for this? Don't we keep character data as Java Strings in memory (ie UTF-16)? So they have to be already free of the original CP. The conversion to a UTF8 database storage should be done automatically, by DB driver. Do I oversimplify this?

I think you are on the right track. My understanding of this process is that the .d files must be read with the correct encoding (Window 1252 or 8859-1 or however the file are encoded). At that point any text is now in Java String instances which are Unicode. As long as the database itself is defined as UTF-8 then I assume this part is automatic.

Questions:

1. How do we control the encoding when we read the .d inputs? Do we have work here?
2. How do we ensure the database is UTF-8?
3. What do we need to do to check if the Progress UTF collation is compatible with the PostgreSQL UTF approach?
4. What am I missing?

> I agree that, for some edge-cases, FWD will not sort some strings the exact way OE does, and this is contrary to our FWD paradigm. But, OTOH, selecting the proper locale, the user will actually get the results on screen the way (s)he naturally expects.

The problem is that the application was written with different expectations. The users are expecting the OpenEdge behavior so I don't think a more intuitive collation will save us. When the application works differently in FWD, this will most often be considered a bug in FWD.

#### #3 - 01/14/2019 10:20 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Questions:
> 1. How do we control the encoding when we read the .d inputs? Do we have work here?

Normally, one should not worry as long as the correct encoding specified when the input stream is open. However, for parsing .d files we use our FileStream which, at the lowest level, processes one character at a time (see readLn() and read() methods). Before returning the data, the characters are passed through the currently set CharsetConverter.

2. How do we ensure the database is UTF-8?

For PostgreSQL, the ENCODING = 'UTF8'  option should do it. However, it requires that both LC_COLLATE and LC_CTYPE to be set to 'en_US.UTF8'.

3. What do we need to do to check if the Progress UTF collation is compatible with the PostgreSQL UTF approach?

If we create a table with 2^16 records: [i, char(i)], for i = 0..65535. Then sort/index the table according to 2nd column in both DB to a file - this will use the collation for ordering all these chars. Then compare the outputs for first column. Of course, this does not cover the full set of UTF, but the first 64Ki are the most used so this is just partial correctness test, not a completeness.

#### #4 - 07/04/2019 04:15 AM - Marian Edu

Greg, what exactly do you need us to do on this one?

Albeit related the code page character encoding and the collation are different things, for each encoding code page Progress supports a number of collations and there is also a crazy option to define your own collation table - no idea if that is used, I presume there is at least someone doing that since the feature is available but haven't seen that before :)

The code page is set when the database is created and then it can be changed either using dump&load or using proutil convchar convert. However, for collation one need to load the collation tables afterwards (there are various .df files in prolang folder). The database collation only affects how the data is sorted in database indexes, the 4GL client can use a different collation at runtime (-cpcoll). There are also character tables (define non-characters), case tables (lower/upper case), codepage conversion tables and word-break tables (for word indexes, used with CONTAINS)... all those are provided for different languages and one can customize them to 'compile' a specific convmap.cp file.

What test cases are needed here?

#### #5 - 07/04/2019 10:39 AM - Greg Shah

In this task we are understanding and resolving issues related to reading a database dump (.d files) which has data in one codepage and then converting that data during import processing so that when it is written into the target database it is in the correct target codepage.

As part of this set of issues we do want to consider the degree to which we can be expected to see collation issues in the result.  In other words, users of the original application would have seen a specific sorting in queries that may no longer match the sorting seen by default in the target codepage of the imported database.  We raise this issue here because it is typically the most visible difference that we have seen in how the 4GL sorts as compared to default sorting in standard operating system locales.

Likewise the character attribute tables and case tables have a direct affect on the converted code behavior.

For this task we can ignore the word break tables.  These are not entirely irrelevant, but for our purposes we will consider them in another task.

An underlying assumption of our approach so far is that the definition of a codepage is an international standard that the 4GL honors and does not modify or override in any way. I know of no way to define a new codepage in the 4GL (i.e. there is no such facility in convmap.dat). And if the 4GL was to implement such an idea, it would not make sense since no other technology on the planet would be able to exchange data with 4GL-specific codepages. Let me know if my assumption is wrong.

Given the above, the following questions need to be answered:

1. Is there any form of codepage conversion in the 4GL client or the OpenEdge database that is not defined explicitly using the codepage conversion tables in convmap.dat?

2. We need a way to test that the FWD codepage conversion implementation matches the results from the 4GL. The idea is to implement a tool to capture the conversion results given a specific pair of source codepage and target codepage. The results would need to be written to a file in some form that has no hidden codepage conversion. We would need some way to encode failures (input characters that cannot be converted to an output character in that given source/target pair). The idea is that we can run this tool on both the 4GL and FWD and compare the resulting output. I wonder if we need a tool to do the comparison of results since the format of the file will be custom and a simple diff won't be useful. In other words, a results comparison tool is probably needed to interpret the differences.

3. We need a way to test that the FWD charset implementation matches the results from the 4GL. The idea is to implement a tool to capture the collation, case conversion and character attribute results for a given codepage. The results would need to be written to a file in some form that has no hidden codepage conversion. We would need some way to encode failures if any exist. The idea is that we can run this tool on both the 4GL and FWD and compare the resulting output. If needed create a tool to do the comparison of results since the format of the file.

4. You stated that "database collation only affects how the data is sorted in database indexes". Is it correct that any sorting which is not based on an index match will be collated using the 4GL CPCOLL setting?

If the answer can be stated authoritatively without writing tests to determine the result, that is OK. But if the answer is unclear, then tests should be written.

Given the answers/tools in 1 and 2 we can determine the approach for the database import (and we can check our existing copepage conversion implementation).

The answers/tools in 3 and 4 will help us determine the problems we will see for data that has been converted. They will also let us test our implementation and fix it.

I think this work overlaps with #3753.

**#6 - 08/21/2019 05:21 AM - Marian Edu**

Greg Shah wrote:

> An underlying assumption of our approach so far is that the definition of a codepage is an international standard that the 4GL honors and does

> not modify or override in any way. I know of no way to define a new codepage in the 4GL (i.e. there is no such facility in convmap.dat). And if the 4GL was to implement such an idea, it would not make sense since no other technology on the planet would be able to exchange data with 4GL-specific codepages. Let me know if my assumption is wrong.

Your assumption is correct, at least to my knowledge.

> 1. Is there any form of codepage conversion in the 4GL client or the OpenEdge database that is not defined explicitly using the codepage conversion tables in convmap.dat?

No, if you try to convert from one codepage to another and there is no conversion table for that in convmap.dat there will be a runtime error (#6063).

> 2. We need a way to test that the FWD codepage conversion implementation matches the results from the 4GL. The idea is to implement a tool to capture the conversion results given a specific pair of source codepage and target codepage. The results would need to be written to a file in some form that has no hidden codepage conversion. We would need some way to encode failures (input characters that cannot be converted to an output character in that given source/target pair). The idea is that we can run this tool on both the 4GL and FWD and compare the resulting output. I wonder if we need a tool to do the comparison of results since the format of the file will be custom and a simple diff won't be useful. In other words, a results comparison tool is probably needed to interpret the differences.

I need to think about this a bit, what we could do is to write tests to convert from one codepage to another proven this is supported in convmap and then you can just compare the results from progress with what you get when running the same in fwd... can't think of any other comparison tool other than binary compare/diff right now :(

> 3. We need a way to test that the FWD charset implementation matches the results from the 4GL. The idea is to implement a tool to capture the collation, case conversion and character attribute results for a given codepage. The results would need to be written to a file in some form that has no hidden codepage conversion. We would need some way to encode failures if any exist. The idea is that we can run this tool on both the 4GL and FWD and compare the resulting output. If needed create a tool to do the comparison of results since the format of the file.
>
> 4. You stated that "database collation only affects how the data is sorted in database indexes". Is it correct that any sorting which is not based on an index match will be collated using the 4GL CPCOLL setting?

Yes this is correct, the CPCOLL only affects 'client side sorting' - any sorting not already done by the server using available indexes (or if the index used for filtering out data is not usable for sorting as requested by the BY options).

> If the answer can be stated authoritatively without writing tests to determine the result, that is OK. But if the answer is unclear, then tests should be written.

Not completely sure about this so maybe some tests could help but I need to think about what it could make sense here and how to approach this...

#### #7 - 09/03/2019 03:42 PM - Greg Shah

1. How do we control the encoding when we read the .d inputs? Do we have work here?

Normally, one should not worry as long as the correct encoding specified when the input stream is open. However, for parsing .d files we use our FileStream which, at the lowest level, processes one character at a time (see readLn() and read() methods). Before returning the data, the characters are passed through the currently set CharsetConverter.

My quick look at ImportWorker shows that we are ignoring the cpstream encoding specified in the dump file footer. See the DataFileReader.processPscHeader(). We read the encoding value but then we ignore it. I think that setConvertSource(encoding) should be called inside DataFileReader.processPscHeader().

2. How do we ensure the database is UTF-8?

For PostgreSQL, the ENCODING = 'UTF8' option should do it. However, it requires that both LC_COLLATE and LC_CTYPE to be set to 'en_US.UTF8'.

I think there is something more needed here. In particular, the ImportWorker needs to know the target codepage. If this is different from the source codepage in the DataFileReader, then we must probably implement the charset conversion before output to the database. I don't think this will be in the DataFileReader but somewhere else.

#### #8 - 09/09/2019 09:43 AM - Greg Shah

*- Assignee set to Sergey Ivanovskiy*

#### #9 - 09/20/2019 05:54 AM - Sergey Ivanovskiy

It follows that within ImportWorker namespace DataFileReader.processPscHeader() fills encoding field and this field is not used for data conversion. And the default CharsetConverter is used for this purpose. This default converter is given by the private static final field cc of FileStream. And

within Stream these fields are defined too

```
   /** External option value for standard character translations occur or not. */
   protected boolean convert = true;

   /** Internal value. Determines if standard character translations occur or not. */
   protected boolean _convert = false;

   /** Determines if codepage conversion decision is cached. */
   protected boolean convertCached = false;

   /** Source codepage for character conversion. */
   protected String sourceCp = null;

   /** Target codepage for character conversion. */
   protected String targetCp = null;

   /** Default value of the -cpstream option. */
   protected String streamCp = null;

   /** Default value of the -cpinternal option. */
   protected String internalCp = null;
```

It seems that only int read(), void write(String) and void writeCh(char) of FileStream methods can use CharsetConvertor.

#### #10 - 09/20/2019 06:07 AM - Sergey Ivanovskiy

*- Status changed from New to WIP*

#### #11 - 09/20/2019 06:09 AM - Sergey Ivanovskiy

What should be done in order to test solutions for this task?

#### #12 - 09/20/2019 06:18 AM - Constantin Asofiei

First step I think is to create a database in 4GL with non-default codepage/locale, add some records with non-ASCII chars and see how that gets imported in FWD.

#### #13 - 09/20/2019 07:05 AM - Constantin Asofiei

Some other things to check:

- to access the data dump, you need to start  the procedure editor; what happens if you set the -cpinternal, i.e. pro -cpinternal iso8859-1, and the database has another codepage?
- when you export the data, you can manually set the codepage in that dialog; what happens if you set a codepage different than the database codepage?  You can create multiple exports, in different codepages, and see how the data looks.
- more, you can do this in reverse - use a different -cpinternal and try to import the data back - does 4GL complain of something?  Is the data imported correctly?

I want to understand if we can rely on the cpstream value in the data dump file (the header at the end of it), to set the codepage for the data import.

#### #14 - 09/20/2019 01:45 PM - Sergey Ivanovskiy

Are these statements correct that -cpinternal code page is used for character variables and fields and -cpstream is used for character conversion while loading and exporting files via streams?

#### #15 - 09/20/2019 01:50 PM - Sergey Ivanovskiy

I used this command to rebuild indices for UTF-8 database and please look what was the output
proutil encode1.db -C idxbuild all

```
OpenEdge Release 11.6.3 as of Thu Sep  8 19:01:50 EDT 2016
The BI file is being automatically truncated. (1526)
Use "-cpinternal UTF-8" with idxbuild only with a UTF-8 database. (8557)
Index rebuild did not complete successfully
```

#### #16 - 09/20/2019 02:00 PM - Sergey Ivanovskiy

and proutil encode1.db -C idxbuild all -cpinternal UTF-8 was successful. It seems that for UTF-8 database encoding the 4GL internal encoding should be UTF-8 too.

#### #17 - 09/20/2019 02:48 PM - Greg Shah

Sergey Ivanovskiy wrote:

> Are these statements correct that -cpinternal code page is used for character variables and fields and -cpstream is used for character conversion while loading and exporting files via streams?

Generally, yes.  A better way to say it:

- CPINTERNAL is for any internal (in memory) processing or comparisons of text data
- CPSTREAM is the codepage assumed for data that is read from a stream or written to a stream

If the two refer to different codepages, then there is an implicit conversion for text data when read into memory from a stream or written to a stream from memory.

#### #18 - 09/22/2019 02:20 AM - Sergey Ivanovskiy

I created a 4GL databases encode2.db using UTF-8 encoding based on C:\Progress\OE116_64\prolang\utf\ICU-ru.df. Then I rebuilt its indices successfully

```
proutil encode2.db -C idxbuild all
```

Then run 4GL Procedure editor with this option -cpinternal UTF-8 and opened Data Dictionary and created this strings table

```
ADD TABLE "strings"
  AREA "Schema Area"
  DUMP-NAME "strings"

ADD FIELD "id" OF "strings" AS integer
  FORMAT "->,>>>,>>9"
  INITIAL "0"
  LABEL "ID"
  POSITION 2
  MAX-WIDTH 4
  COLUMN-LABEL "ID"
  ORDER 10

ADD FIELD "vl" OF "strings" AS character
  FORMAT "x(128)"
  INITIAL ""
  LABEL "VALUE"
  POSITION 3
  MAX-WIDTH 256
  COLUMN-LABEL "VALUE"
  ORDER 20

.
PSC
cpstream=UTF-8
.
0000000390
```

and executed this program to insert native words into strings table

```
VIEW DEFAULT-WINDOW.

FIND FIRST _Db.

DISPLAY _Db._Db-Xl-Name.

DEFINE VARIABLE ix AS INTEGER NO-UNDO.
/*
REPEAT ix = 1 TO NUM-DBS:
  DISPLAY DBCODEPAGE(ix).
END.
*/

MESSAGE "SESSION:CHARSET=" SESSION:CHARSET.

MESSAGE "SESSION:CPINTERNAL=" SESSION:CPINTERNAL.

FOR EACH strings:
  DELETE strings.
END.

CREATE strings.
strings.id = 1.

strings.vl = "кошка".

CREATE strings.
strings.id = 2.

strings.vl = "~u041a~u041e~u0428~u041a~u0410":U.
```

```
FIND FIRST strings WHERE id = 2.

MESSAGE strings.vl.
```

Then tried to dump data using Data Administration utility and got ? marks in the dumped file instead of native words

```
1 "?????"
2 "КОШКА"
.
PSC
filename=strings
records=0000000000002
ldbname=encode2
timestamp=2019/09/21-23:07:55
numformat=44,46
dateformat=mdy-1950
map=NO-MAP
cpstream=UTF-8
.
0000000030
```

Actually it is possible to form the dump manually for testing of import. What was incorrect in these described steps? It seems that 4GL Procedure Editor doesn't support UTF-8 or it needs to setup it for UTF-8?

#### #19 - 09/22/2019 03:03 AM - Sergey Ivanovskiy

The fragment of the converted program looks incorrect too

```
        strings.create();
        strings.setIdentifier(new integer(1));
        strings.setVl(new character("?????"));
        strings.create();
        strings.setIdentifier(new integer(2));
        strings.setVl(new character("u041au041eu0428u041au0410"));
```

#### #20 - 09/22/2019 04:30 PM - Greg Shah

Please see #4279-91 for how to fix the unicode character escape sequence.  I don't know what is wrong with the other case "кошка".

#### #21 - 09/23/2019 02:11 PM - Sergey Ivanovskiy

Greg Shah wrote:

> Please see #4279-91 for how to fix the unicode character escape sequence.  I don't know what is wrong with the other case "кошка".

Thank you. I was trying to run import db task manually

```
/usr/lib/jvm/java-8-openjdk-amd64/bin/java -agentlib:jdwp=transport=dt_socket,suspend=y,address=localhost:3969
2 -Xmx2g -XX:+HeapDumpOnOutOfMemoryError
-classpath /home/sbi/projects/test_import/p2j/build/lib/p2j.jar:/home/sbi/projects/test_import/deploy/lib/test
_import.jar:/home/sbi/projects/test_import/cfg:
-Djava.util.logging.config.file=/home/sbi/projects/test_import/cfg/logging.properties
-DP2J_HOME=.
-server
-Dfile.encoding=UTF-8
com.goldencode.p2j.pattern.PatternEngine -d 2 dbName="encode2" targetDb="h2"  url="jdbc:h2:/home/sbi/projects/
test_import/deploy/db/encode2;DB_CLOSE_DELAY=-1;MVCC=true;MV_STORE=FALSE"   uid="fwd_user"    pw="user"   maxT
hreads=2  schema/import   data/namespace encode2.p2o 2>&1 | tee data_import_$(date '+%Y%m%d_%H%M%S').log
```

but the output was that

```
Syntax:
   java PatternEngine [options]
                      ["<variable>=<expression>"...]
                      <profile>
                      [<directory> "<filespec>" | <filelist>]

where

   options:
      -d <debuglevel> Message output mode;
                      debuglevel values:
                        0 = no message output
                        1 = status messages only
                        2 = status + debug messages
                        3 = verbose trace output
      -c              Call graph walking mode
      -f              Explicit file list mode
      -h              Honor hidden mode
      -r              Read-only mode

   variable   = variable name
   expression = infix expression used to initialize variable
   profile    = rules pipeline configuration filename
   directory  = directory in which to search recursively for persisted ASTs
   filespec   = file filter specification to use in persisted AST search
   filelist   = arbitrary list of absolute and/or relative file names of persisted AST files to process (-f mo
de)
```

It seems that some input parameters are incorrect or missed.?

#### #22 - 09/23/2019 02:39 PM - Greg Shah

Yes, command line is wrong.  It seems like data/namespace encode2.p2o should not have a space in it.

#### #23 - 09/23/2019 02:48 PM - Sergey Ivanovskiy

This ant task works properly for this project:

```
  <target name="import.db.h2"
          description="Import data (.d files) into database."
          if="${db.h2}">
     <record name="import_db_h2_${db.name}_${LOG_STAMP}.log" action="start"/>
     <java classname="com.goldencode.p2j.pattern.PatternEngine"
           fork="true"
           failonerror="true"
           dir="${basedir}" >
        <jvmarg value="-Xmx1g"/>
        <jvmarg value="-Dfile.encoding=UTF-8"/>
        <arg value ="-d"/>
        <arg value ="2"/>
        <arg value ="dbName=${escaped.quotes}${db.name}${escaped.quotes}"/>
        <arg value ="targetDb=${escaped.quotes}h2${escaped.quotes}"/>
        <arg value ="url=${escaped.quotes}${sql.url.h2}${escaped.quotes}"/>
        <arg value ="uid=${escaped.quotes}${sql.user}${escaped.quotes}"/>
        <arg value ="pw=${escaped.quotes}${sql.user.pass}${escaped.quotes}"/>
        <arg value ="maxThreads=4"/>
        <arg value ="schema/import"/>
        <arg value ="data/namespace/"/>
        <arg value ="${db.name}.p2o"/>
        <classpath refid="app.classpath"/>
     </java>
     <record name="import_db_h2_${db.name}_${LOG_STAMP}.log" action="stop"/>
  </target>
```

#### #24 - 09/23/2019 03:16 PM - Sergey Ivanovskiy

This configuration

```
java -server -Xmx2g -XX:+HeapDumpOnOutOfMemoryError -classpath /home/sbi/projects/test_import/p2j/build/lib/p2
j.jar:/home/sbi/projects/test_import/p2j/build/lib/fwdaopltw.jar:/home/sbi/projects/test_import/deploy/lib/tes
t_import.jar:/home/sbi/projects/test_import/cfg: -Djava.util.logging.config.file=/home/sbi/projects/test_impor
t/cfg/logging.properties -Djava.library.path=/home/sbi/projects/4286a/build/lib -DP2J_HOME=. -Dfile.encoding=U
TF-8 com.goldencode.p2j.pattern.PatternEngine -d 2 dbName="encode2" targetDb="h2" url="jdbc:h2:/home/sbi/proje
```

```
cts/test_import/deploy/db/encode2;DB_CLOSE_DELAY=-1;MVCC=true;MV_STORE=FALSE" uid="fwd_user" pw="user" maxThre
ads=2 schema/import data/namespace/encode2.p2o
```

doesn't work too.

#### #25 - 09/23/2019 03:27 PM - Greg Shah

Debug into the PatternEngine to see why it is failing.

#### #26 - 09/23/2019 03:29 PM - Eric Faulhaber

Sergey Ivanovskiy wrote:

> This configuration
> [...]
> doesn't work too.

What exactly is not working? If there is something not looking right in the imported database and you believe the encoding has been handled correctly through the reading part of the import, I would urge you to focus on getting PostgreSQL working first. H2 is not a production target.

BTW, if you do work with H2, use MVCC=FALSE (we use PageStore, not MVSTORE; see https://github.com/h2database/h2database/issues/1204), though I don't think that has to do with any encoding issue.

#### #27 - 09/23/2019 04:22 PM - Sergey Ivanovskiy

It seems that H2 or PostgreSQL should have the same issue when importing UTF-8 dumps. OK. I will test PostgreSQL. It should be similar because ant script from hotel_gui project was used. The debugging of PatternEngine.main proved that the correct command should look like this one

```
java -Xmx2g -XX:+HeapDumpOnOutOfMemoryError -classpath /home/sbi/projects/test_import/p2j/build/lib/p2j.jar:/h
ome/sbi/projects/test_import/p2j/build/lib/fwdaopltw.jar:/home/sbi/projects/test_import/deploy/lib/test_import
.jar:/home/sbi/projects/test_import/cfg: -Djava.util.logging.config.file=/home/sbi/projects/test_import/cfg/lo
gging.properties -DP2J_HOME=. -server -Dfile.encoding=UTF-8 com.goldencode.p2j.pattern.PatternEngine -d 2 dbNa
me="encode2" targetDb="h2" url="jdbc:h2:/home/sbi/projects/test_import/deploy/db/encode2;DB_CLOSE_DELAY=-1;MVC
C=true;MV_STORE=FALSE" uid="fwd_user" pw="user" maxThreads=1 schema/import data/namespace/ encode2.p2o
```

The main function expects 11 parameters. Probably I should add all dependencies to classpath in order that this command works correctly.

#### #28 - 09/24/2019 08:50 AM - Sergey Ivanovskiy

Running this script for PostgreSQL

```
#!/bin/bash

cpath=""
for f in deploy/lib/*.jar
do
   cpath=$cpath:$f
done

echo $cpath

java -server -Xmx2g -XX:+HeapDumpOnOutOfMemoryError -classpath $cpath -Djava.util.logging.config.file=/home/sb
i/projects/test_import/cfg/logging.properties -DP2J_HOME=. -Dfile.encoding=UTF-8 com.goldencode.p2j.pattern.Pa
tternEngine -d 2 "dbName=\"encode2\"" "targetDb=\"postgresql\"" "url=\"jdbc:postgresql://localhost:5434/encode
2\"" "uid=\"fwd_user\"" "pw=\"user\"" "maxThreads=2" schema/import data/namespace/ encode2.p2o
```

throws exceptions

```
INFO:  Type match assertion disabled;  set "checkTypes" to true to enable
INFO:  Data export files will be read from 'data/dump/encode2/'
INFO:  Using 2 threads for import
0    [main] INFO  org.hibernate.dialect.Dialect  - HHH000400: Using dialect: com.goldencode.p2j.persist.dialec
t.P2JPostgreSQLDialect
./data/namespace/encode2.p2o
95   [main] INFO  org.hibernate.annotations.common.Version  - HCANN000001: Hibernate Commons Annotations {4.0.
1.Final}
100  [main] INFO  org.hibernate.Version  - HHH000412: Hibernate Core {4.1.8.Final}
102  [main] INFO  org.hibernate.cfg.Environment  - HHH000206: hibernate.properties not found
104  [main] INFO  org.hibernate.cfg.Environment  - HHH000021: Bytecode provider name : javassist
135  [main] INFO  org.hibernate.cfg.Configuration  - HHH000221: Reading mappings from resource: com/goldencode
/test_import/dmo/encode2/impl/MetaUserImpl.hbm.xml
321  [main] INFO  org.hibernate.cfg.Configuration  - HHH000221: Reading mappings from resource: com/goldencode
/test_import/dmo/encode2/impl/StringsImpl.hbm.xml
379  [main] INFO  org.hibernate.service.jdbc.connections.internal.ConnectionProviderInitiator  - HHH000130: In
stantiating explicit connection provider: org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvid
er
379  [main] INFO  org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider  - HHH010002: C3P0 us
ing driver: org.postgresql.Driver at URL: jdbc:postgresql://localhost:5434/encode2
380  [main] INFO  org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider  - HHH000046: Connect
ion properties: {user=fwd_user, password=****}
380  [main] INFO  org.hibernate.service.jdbc.connections.internal.C3P0ConnectionProvider  - HHH000006: Autocom
mit mode: false
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/sbi/projects/test_import/deploy/lib/slf4j-jdk14-1.6.1.jar!/org/slf4j/i
mpl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/sbi/projects/test_import/deploy/lib/slf4j-simple-1.6.1.jar!/org/slf4j/
impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.JDK14LoggerFactory]
Sep 24, 2019 3:46:54 PM com.mchange.v2.log.slf4j.Slf4jMLog$Slf4jMLogger$WarnLogger log
WARNING: Bad pool size config, start 3 < min 8. Using 8 as start.
1142 [main] INFO  org.hibernate.dialect.Dialect  - HHH000400: Using dialect: com.goldencode.p2j.persist.dialec
t.P2JPostgreSQLDialect
1150 [main] INFO  org.hibernate.engine.jdbc.internal.LobCreatorBuilder  - HHH000424: Disabling contextual LOB
creation as createClob() method threw error : java.lang.reflect.InvocationTargetException
1159 [main] INFO  org.hibernate.engine.transaction.internal.TransactionFactoryInitiator  - HHH000399: Using de
fault transaction strategy (direct JDBC transactions)
1162 [main] INFO  org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory  - HHH000397: Using ASTQueryTransla
torFactory
1165 [main] INFO  org.hibernate.dialect.Dialect  - HHH000400: Using dialect: com.goldencode.p2j.persist.dialec
t.P2JPostgreSQLDialect
1222 [main] INFO  org.hibernate.hql.internal.ast.ASTQueryTranslatorFactory  - HHH000397: Using ASTQueryTransla
torFactory
Skipping table meta_user;  previously imported
Skipping table strings;  previously imported
IMPORT ORDER:
1646 [main] WARN  org.hibernate.engine.jdbc.spi.SqlExceptionHelper  - SQL Error: 0, SQLState: 42P07
1646 [main] ERROR org.hibernate.engine.jdbc.spi.SqlExceptionHelper  - ERROR: relation "p2j_id_generator_sequen
ce" already exists
org.hibernate.exception.SQLGrammarException: ERROR: relation "p2j_id_generator_sequence" already exists
    at org.hibernate.exception.internal.SQLStateConversionDelegate.convert(SQLStateConversionDelegate.java:122
```

```
)
    at org.hibernate.exception.internal.StandardSQLExceptionConverter.convert(StandardSQLExceptionConverter.ja
va:49)
    at org.hibernate.engine.jdbc.spi.SqlExceptionHelper.convert(SqlExceptionHelper.java:125)
    at org.hibernate.engine.jdbc.spi.SqlExceptionHelper.convert(SqlExceptionHelper.java:110)
    at org.hibernate.engine.jdbc.internal.proxy.AbstractStatementProxyHandler.continueInvocation(AbstractState
mentProxyHandler.java:132)
    at org.hibernate.engine.jdbc.internal.proxy.AbstractProxyHandler.invoke(AbstractProxyHandler.java:80)
    at com.sun.proxy.$Proxy5.executeUpdate(Unknown Source)
    at org.hibernate.engine.query.spi.NativeSQLQueryPlan.performExecuteUpdate(NativeSQLQueryPlan.java:204)
    at org.hibernate.internal.SessionImpl.executeNativeUpdate(SessionImpl.java:1289)
    at org.hibernate.internal.SQLQueryImpl.executeUpdate(SQLQueryImpl.java:400)
    at com.goldencode.p2j.schema.ImportWorker.createIdentitySequence(ImportWorker.java:448)
    at com.goldencode.p2j.schema.ImportWorker.access$14(ImportWorker.java:424)
    at com.goldencode.p2j.schema.ImportWorker$Library.runImport(ImportWorker.java:770)
    at com.goldencode.expr.CE99.execute(Unknown Source)
    at com.goldencode.expr.Expression.execute(Expression.java:391)
    at com.goldencode.p2j.pattern.Rule.apply(Rule.java:497)
    at com.goldencode.p2j.pattern.RuleContainer.apply(RuleContainer.java:585)
    at com.goldencode.p2j.pattern.RuleSet.apply(RuleSet.java:1)
    at com.goldencode.p2j.pattern.PatternEngine.apply(PatternEngine.java:1652)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1531)
    at com.goldencode.p2j.pattern.PatternEngine.processAst(PatternEngine.java:1479)
    at com.goldencode.p2j.pattern.PatternEngine.run(PatternEngine.java:1034)
    at com.goldencode.p2j.pattern.PatternEngine.main(PatternEngine.java:2110)
Caused by: org.postgresql.util.PSQLException: ERROR: relation "p2j_id_generator_sequence" already exists
    at org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:2440)
    at org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:2183)
    at org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:308)
    at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:441)
    at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:365)
    at org.postgresql.jdbc.PgPreparedStatement.executeWithFlags(PgPreparedStatement.java:143)
    at org.postgresql.jdbc.PgPreparedStatement.executeUpdate(PgPreparedStatement.java:120)
    at com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.executeUpdate(NewProxyPreparedStatement.java:384)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.hibernate.engine.jdbc.internal.proxy.AbstractStatementProxyHandler.continueInvocation(AbstractState
mentProxyHandler.java:124)
    ... 18 more
Total records processed:  0 in 0:00:00.260 (0.000 records/sec)
Total sequences initialized: 0.
Reading merge DMO definitions...
Elapsed job time:  00:00:02.307
```

and running the corresponding script for H2 throws the same exceptions.

#### #29 - 09/24/2019 08:55 AM - Ovidiu Maxiniuc

This is normal. The database was already imported at least once.  Remove the sequence manually.
Note that if a table was already imported (if it contains at least one record) the import will skip it. In consequence, if you want to reimport a table you need to manually drop all records from that table.

#### #30 - 09/24/2019 09:15 AM - Sergey Ivanovskiy

Ovidiu, thank you.

Now the db import scripts work properly, although the import of UTF-8 data

1 "cat"
2 "КОШКА"
.
PSC
filename=strings
records=0000000000002
ldbname=encode2
timestamp=2019/09/23-07:07:33
numformat=44,46
dateformat=mdy-1950
map=NO-MAP
cpstream=UTF-8
.
0000000028

into encode2 PostgreSql database looks incorrect

```
id | identifier |               vl
----+------------+---------------------------------
  1 |          1 | cat
  2 |          2 | Đ\u009AĐ\u009EĐ¨Đ\u009AĐ\u0090
(2 rows)
```

At least I can dig into ImportWorker now.

#### #31 - 09/24/2019 09:46 AM - Sergey Ivanovskiy

What is the proper database encoding if data dump files are UTF-8? Should not it be UTF-8? Can it be LATIN-1 if another multibyte encoding is used?

The previous import is still correct. It needs to change the psql client encoding to LATIN-1 in order to read imported strings correctly by LATIN-1 database

```
encode2=# \encoding LATIN-1
encode2=# select * from strings;
id | identifier |     vl
----+------------+------------
1 |          1 | cat
2 |          2 | КОШКА
(2 rows)
```

#### #32 - 09/24/2019 09:50 AM - Greg Shah

> What is the proper database encoding if data dump files are UTF-8? Should not it be UTF-8? Can it be LATIN-1 if another multibyte encoding is used?

The objective of this task is to successfully process import when the .d data is encoded with a codepage that is different from the database codepage.

If the two encodings are the same, then we should not (and do not) convert the data.  Please fix the "2 different encodings" scenario, which we know is broken.

#### #33 - 09/24/2019 09:52 AM - Sergey Ivanovskiy

It seems that the client can use different encodings https://www.postgresql.org/docs/9.6/multibyte.html and hence the database driver level can be involved in order to get correct UTF-8 strings from LATIN-1 database.

#### #34 - 09/24/2019 10:15 AM - Sergey Ivanovskiy

Greg, I imported UTF-8 data into LATIN-1 PostgresSQL database using -Dfile.encoding=UTF-8 input parameter. I am not sure about this parameter -Dfile.encoding=UTF-8 because it overrides the java default charset.

#### #35 - 09/24/2019 10:22 AM - Greg Shah

Please read #3871-7 for the details on what you are meant to implement.

- For the input encoding,tThe DataFileReader.processPscHeader() must call setConvertSource(encoding).
- For the output encoding:
  - We must have a way to pass this as a configuration parameter for the import process OR we must have a database-independent way to query it dynamically from the database instance.
  - We must have a way to honor it with the import so that the character data is properly converted.

#### #36 - 09/25/2019 09:00 AM - Sergey Ivanovskiy

I don't know a database-independent way to query a database encoding. For PosgreSQL this query returns its encoding

```
SELECT pg_encoding_to_char(encoding) FROM pg_database WHERE datname = ?;
```

#### #37 - 09/25/2019 09:05 AM - Sergey Ivanovskiy

Google helps that this query follows ANSI standard schema view https://en.wikipedia.org/wiki/Information_schema

```
SELECT character_set_name FROM information_schema.character_sets;
```

#### #38 - 10/02/2019 01:54 PM - Sergey Ivanovskiy

Created task branch 3871a.

#### #39 - 10/07/2019 06:26 AM - Sergey Ivanovskiy

Let us consider this case in which the database has LATIN-1 encoding and the import table has WINDOWS-1251 encoding. The data can be read correctly from this import table and then string values can be converted into LATIN-1 strings and imported into the database. Then the client will read the imported data from this database as LATIN-1 strings because the database has LATIN-1 encoding. Finally the output will be incorrect as the client doesn't interpret the read bytes as WINDOWS-1251 strings. It seems that the client should have a knowledge about the encoding of the imported data. Please clarify what should be done in this case.

#### #40 - 10/07/2019 02:17 PM - Sergey Ivanovskiy

I encountered the following issue that the WINDOWS-1251 string decoded into the UTF-8 string (if UTF-8 is a default java encoding) was not encoded as LATIN-1 string. It seems in this case we need to use the target code page LATIN-1 in order to decode the original string into LATIN-1 string. This algorithm should work only if the charset used by the imported table has one byte per a char encoding. And in this case CharsetConverter can be used. For multibyte and unicode encodings this class doesn't work and this case should be considered separately.

#### #41 - 10/08/2019 05:15 AM - Sergey Ivanovskiy

I think that there is only one way to save UTF-8/Unicode strings into LATIN-1 database that is to convert UTF-8 strings to ones with ASCII/Unicode escapes.

#### #42 - 10/08/2019 05:25 AM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> I think that there is only one way to save UTF-8 strings into LATIN-1 database is to convert UTF-8 strings to ones with ASCII/Unicode escapes.

It needs to implement this tool's functionality
https://docs.oracle.com/javase/7/docs/technotes/tools/solaris/native2ascii.html

#### #43 - 10/08/2019 05:47 AM - Greg Shah

> Then the client will read the imported data from this database as LATIN-1 strings because the database has LATIN-1 encoding. Finally the output will be incorrect as the client doesn't interpret the read bytes as WINDOWS-1251 strings. It seems that the client should have a knowledge about the encoding of the imported data. Please clarify what should be done in this case.

What is the "client" in your example? Is it the converted code in the FWD server?

For the purposes of this task, you can assume that the FWD server will be expecting the database encoding. The result of the import must be the same as if the import data was originally encoded in the target codepage (LATIN-1 in this example).

> I encountered the following issue that the WINDOWS-1251 string decoded into the UTF-8 string (if UTF-8 is a default java encoding) was not encoded as LATIN-1 string.

It is not clear what you are talking about here. Where are the WINDOWS-1251 strings coming from? Why do I want them encoded as LATIN-1?

> I think that there is only one way to save UTF-8/Unicode strings into LATIN-1 database that is to convert UTF-8 strings to ones with ASCII/Unicode escapes.

Where are the UTF-8 strings coming from? Are they from the import data?

#### #44 - 10/08/2019 05:59 AM - Sergey Ivanovskiy

Greg Shah wrote:

> > Then the client will read the imported data from this database as LATIN-1 strings because the database has LATIN-1 encoding. Finally the output will be incorrect as the client doesn't interpret the read bytes as WINDOWS-1251 strings. It seems that the client should have a knowledge about the encoding of the imported data. Please clarify what should be done in this case.
>
> What is the "client" in your example? Is it the converted code in the FWD server?

The client is ImportWorker.

> For the purposes of this task, you can assume that the FWD server will be expecting the database encoding. The result of the import must be the same as if the import data was originally encoded in the target codepage (LATIN-1 in this example).

What is the expecting database encoding. Should it be the same as the imported data file?

> I encountered the following issue that the WINDOWS-1251 string decoded into the UTF-8 string (if UTF-8 is a default java encoding) was not encoded as LATIN-1 string.

> It is not clear what you are talking about here. Where are the WINDOWS-1251 strings coming from? Why do I want them encoded as LATIN-1?

The imported file has WINDOWS-1251 encoding. LATIN-1 encoding is the enconding of the target database. #3871-39.

> I think that there is only one way to save UTF-8/Unicode strings into LATIN-1 database that is to convert UTF-8 strings to ones with ASCII/Unicode escapes.

> Where are the UTF-8 strings coming from? Are they from the import data?

UTF-8 is the default encoding for my java environment (JRE).

**#45 - 10/08/2019 06:10 AM - Sergey Ivanovskiy**

> Where are the UTF-8 strings coming from? Are they from the import data?

UTF-8 is the default encoding for my java environment (JRE). In my case (ImportWorker) these unicode strings are sql insert queries. To describe it more precisely

```
com.goldencode.p2j.schema.ImportWorker$Library importTable
SEVERE: Dropped record #1 in strings.d due to error:  Batch entry 0 insert into strings (identifier, vl, id) v
alues (1, 'КОШКА', 2) was aborted: ERROR: character with byte sequence 0xd0 0x9a in encoding "UTF8" has no equ
ivalent in encoding "LATIN1"  Call getNextException to see other errors in the batch.
```

**#46 - 10/08/2019 06:16 AM - Greg Shah**

Sergey Ivanovskiy wrote:

> Greg Shah wrote:
>
> > Then the client will read the imported data from this database as LATIN-1 strings because the database has LATIN-1 encoding. Finally the output will be incorrect as the client doesn't interpret the read bytes as WINDOWS-1251 strings. It seems that the client should have a knowledge about the encoding of the imported data. Please clarify what should be done in this case.
>
> What is the "client" in your example?  Is it the converted code in the FWD server?
>
> The client is ImportWorker.

Any reads or writes to a database encoded as LATIN-1 would be expected to be in LATIN-1.  I don't know why the ImportWorker needs to read from the database.  The ImportWorker primariy writes to the database.

> Finally the output will be incorrect as the client doesn't interpret the read bytes as WINDOWS-1251 strings.

Why would there be WINDOWS-1251 strings in a LATIN-1 database?  The **entire** purpose of this task is to ensure that any data read from .d files in INPUT_ENCODING (WINDOWS-1251 in your example?) is converted to DATABASE_ENCODING (LATIN-1 in your example?) **before** it is inserted into the database.  That means that any data in the database will no longer be in the INPUT_ENCODING.

> > For the purposes of this task, you can assume that the FWD server will be expecting the database encoding.  The result of the import must be the same as if the import data was originally encoded in the target codepage (LATIN-1 in this example).
>
> What is the expecting database encoding. Should it be the same as the imported data file?

No.  The purpose of the task is to resolve the conflict caused by INPUT_ENCODING != DATABASE_ENCODING.

> > I encountered the following issue that the WINDOWS-1251 string decoded into the UTF-8 string (if UTF-8 is a default java encoding) was not encoded as LATIN-1 string.

> It is not clear what you are talking about here. Where are the WINDOWS-1251 strings coming from? Why do I want them encoded as LATIN-1?

The imported file has WINDOWS-1251 encoding. LATIN-1 encoding is the enconding of the target database. #3871-39.

There should not be any WINDOWS-1251 strings in the database. They should have been converted already by the ImportWorker.

> I think that there is only one way to save UTF-8/Unicode strings into LATIN-1 database that is to convert UTF-8 strings to ones with ASCII/Unicode escapes.

> Where are the UTF-8 strings coming from? Are they from the import data?

UTF-8 is the default encoding for my java environment (JRE).

We can assume that all data to be converted from UTF-8 to a SBCS must have a single byte encoding. We do NOT need to try to encode multi-byte sequences using escapes. There is no use case where this is needed. If there is no valid codepage conversion for the INPUT_ENCODING to the DATABASE_ENCODING, then you should throw an exception and end the import.

#### #47 - 10/08/2019 06:24 AM - Sergey Ivanovskiy

I don't know the way how to convert parameters for HSQL queries like this one

```
com.goldencode.p2j.schema.ImportWorker$Library importTable
SEVERE: Dropped record #1 in strings.d due to error:  Batch entry 0 insert into strings (identifier, vl, id) v
alues (1, 'КОШКА', 2) was aborted: ERROR: character with byte sequence 0xd0 0x9a in encoding "UTF8" has no equ
ivalent in encoding "LATIN1"  Call getNextException to see other errors in the batch.
```

The native word in this example comes from the imported WINDOWS-1251 file and is represented as UTF-8 string correctly. Then used Hibernate persistence api maps the target dmo object into insert query. This query has UTF-8 string that can't be persisted into LATIN-1 database.

#### #48 - 10/08/2019 06:26 AM - Sergey Ivanovskiy

*- File strings_WINDOWS_1251.d added*

This is that imported file encoded as WINDOWS-1251.

#### #49 - 10/08/2019 06:42 AM - Greg Shah

If there is no valid codepage conversion for the INPUT_ENCODING to the DATABASE_ENCODING, then you should throw an exception and end the import.

The user of the import must accept that it cannot convert data that has no representation.  That is an assumption of this task.  If it is not possible to convert WINDOWS-1251 into LATIN-1, then the import does not have to support it.

#### #50 - 10/08/2019 06:52 AM - Sergey Ivanovskiy

OK, it seems that the Hibernate uses UTF-8 internally because I experimented with US-ASCII encoding as a default encoding for JRE and the same error was thrown

```
om.goldencode.p2j.schema.ImportWorker$Library importTable
SEVERE: Dropped record #1 in strings.d due to error:  Batch entry 0 insert into strings (identifier, vl, id) v
alues (1, '?????', 2) was aborted: ERROR: character with byte sequence 0xd0 0x9a in encoding "UTF8" has no equ
ivalent in encoding "LATIN1"  Call getNextException to see other errors in the batch.
```

The imported file has a correct WINDOWS-1251 strings. I checked it manually with help of Bless.

#### #51 - 10/08/2019 10:32 AM - Sergey Ivanovskiy

Sergey Ivanovskiy wrote:

> OK, it seems that the Hibernate uses UTF-8 internally because I experimented with US-ASCII encoding as a default encoding for JRE and the same error was thrown
> [...]
> The imported file has a correct WINDOWS-1251 strings. I checked it manually with help of Bless.

I found that Hibernate supports https://docs.jboss.org/hibernate/orm/5.0/mappingGuide/en-US/html/ch03.html#basic-nationalized but these settings didn't have any effect on the internal conversion of the native strings into UTF-8 string with 0xd0 0x9a in its bytes sequence. It seems that it is a

cornerstone of this task.

#### #52 - 10/08/2019 10:37 AM - Sergey Ivanovskiy

I doesn't matter what one byte encoding is used. If it is internally mapped into UTF-8, then this issue can be observed when we will import the native data into LATIN-1 database.

#### #53 - 10/09/2019 08:49 AM - Sergey Ivanovskiy

*- File 3871_1.patch added*

Greg, please review the committed rev 11338 (3871a). It was tested with UTF-8 PostgresSql database.

#### #54 - 10/10/2019 03:46 AM - Sergey Ivanovskiy

I found that the UTF-8 source dump file is imported incorrectly now. The problem here is FileStream is supposed to be read from sources that have one byte encoding. Working to fix the case when the source encoding is UTF-8.

#### #55 - 10/10/2019 03:53 AM - Sergey Ivanovskiy

*- File strings_UTF_8.d added*

This it the test example. The question for me is that BOM (https://en.wikipedia.org/wiki/Byte_order_mark) is generated by 4GL for UTF-8 dump file isn't it? It seems that it is not added but some software can generate BOM for UTF-8 (Notepad for Windows).

#### #56 - 10/10/2019 03:01 PM - Sergey Ivanovskiy

*- Status changed from WIP to Review*

*- File 3871_2.patch added*

*- % Done changed from 0 to 100*

The committed revision 11343 (3871a) should fix the cases when the source code page is UTF8 or a different code page having one-byte encoding schema and the database is UTF-8.

#### #57 - 10/10/2019 03:10 PM - Sergey Ivanovskiy

Committed revision 11344 (3871a) fixed missed comment.

#### #58 - 10/11/2019 06:32 AM - Sergey Ivanovskiy

3871a was rebased up to revision 11346 over 11337 trunc.

#### #59 - 10/15/2019 04:29 PM - Ovidiu Maxiniuc

Review of r11346 / 3871a.

Generally I am OK with the new code as it seem logical, except for the query from ImportWorker.java:1040. Unfortunately, it will work only on PostgreSQL. We need this line to work in all cases, so a dialect-specific query string is needed. However, I do not know if the character-set is obtainable on all dialects using a simple query.

#### #60 - 10/16/2019 04:26 AM - Sergey Ivanovskiy

Ovidiu Maxiniuc wrote:

> Review of r11346 / 3871a.
>
> Generally I am OK with the new code as it seem logical, except for the query from ImportWorker.java:1040. Unfortunately, it will work only on PostgreSQL. We need this line to work in all cases, so a dialect-specific query string is needed. However, I do not know if the character-set is

obtainable on all dialects using a simple query.

For H2 database this resource http://www.h2database.com/html/advanced.html (Supported Character Sets, Character Encoding, and Unicode) states only that H2 database supports Unicode internally. For an example, INFORMATION_SCHEMA.SCHEMATA has DEFAULT_CHARACTER_SET_NAME but its value is 'Unicode' and it is not useful (http://h2database.com/html/systemtables.html). Could we set UTF-8 as a target encoding for H2 database?

This query

```
SELECT character_set_name FROM information_schema.character_sets;
```

should be ANSI standard.

It is important to set the target encoding

```
                if (targetCharset instanceof String)
                {
                   stream.setConvertTarget((String) targetCharset);
                }
```

because string values should be converted accordingly.

Another unsolved issue is how to turn Hibernate to support native queries encoded using one byte code page.

#### #61 - 10/16/2019 10:22 AM - Ovidiu Maxiniuc

It seems that not all database developers have chosen to honour ANSI standard.

For H2 (I updated my h2 test environment today to 1.4.200 (2019-10-14)), I do not have a information_schema.character_sets table. Instead I looked into the wrote the INFORMATION_SCHEMA and wrote following queries that might be useful:

- select SCHEMA_NAME, DEFAULT_CHARACTER_SET_NAME from INFORMATION_SCHEMA.SCHEMATA
- select TABLE_NAME, COLUMN_NAME, CHARACTER_SET_NAME from INFORMATION_SCHEMA.COLUMNS

Indeed, all results are "Unicode" for my test db, but their existence means a different one can be set.

When it comes to MS SQL Server, they are really different. They use the collations instead. The information about this can be found in sys schema. Please take a look here: Collation and Unicode support and View Collation Information.

#### #62 - 10/16/2019 12:00 PM - Sergey Ivanovskiy

Ovidiu, thank you for help. Should we support only PostgreSql, H2 and MS SQL Server?

#### #63 - 10/16/2019 01:31 PM - Ovidiu Maxiniuc

Yes, these are the only three supported dialects (see implementations of P2JDialect) at this moment.

#### #64 - 10/17/2019 04:33 PM - Sergey Ivanovskiy

At this moment I suppose now that PostgreSQL database has been configured to have UTF-8 encoding. If another one-byte charset encoding was set, then Hibernate insert queries could fail because UTF-8 was used by Hibernate to wrap native strings. I don't know how to set up Hibernate, if it is possible, to work with another one-byte charset. Although this query SELECT character_set_name FROM information_schema.character_sets; returns UTF-8 correctly. The actual encoding is supposed to be UTF-8. For H2 there are no documentation how to set its encoding to one-byte charset. For this database type we can use UTF-8. MSSQL has complex collation settings and I didn't find the documentation that could help to map some collation, for an example, SQL_Latin1_General_CP1_CI_AS, to some standard code page. The documentation states that MS SQL can support multi encoding for a one database and even for a one table and UTF-8 data is supported fully with new MS SQL version. Please see
https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support?view=sql-server-ver15#utf8
Committed revision 11347 changed logic for H2 and MS SQL. Please review and help with ideas if you know ways to solve these described issues.

#### #65 - 10/21/2019 02:48 PM - Greg Shah

Sergey Ivanovskiy wrote:

> At this moment I suppose now that PostgreSQL database has been configured to have UTF-8 encoding. If another one-byte charset encoding was set, then Hibernate insert queries could fail because UTF-8 was used by Hibernate to wrap native strings. I don't know how to set up Hibernate, if it is possible, to work with another one-byte charset. Although this query SELECT character_set_name FROM information_schema.character_sets; returns UTF-8 correctly. The actual encoding is supposed to be UTF-8. For H2 there are no documentation how to set its encoding to one-byte charset. For this database type we can use UTF-8. MSSQL has complex collation settings and I didn't find the documentation that could help to map some collation, for an example, SQL_Latin1_General_CP1_CI_AS, to some standard code page. The documentation states that MS SQL can support multi encoding for a one database and even for a one table and UTF-8 data is supported fully with new MS SQL version. Please see
> https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support?view=sql-server-ver15#utf8
> Committed revision 11347 changed logic for H2 and MS SQL. Please review and help with ideas if you know ways to solve these described issues.

What case are you describing?  It is invalid to have a database with strings encoded as ENCODING1 and then trying to work with that database as if it was ENCODING2.  It doesn't matter what the actual encoding types are.  If you try to use a different encoding than the one set for the database, it won't work properly **and we don't ever expect it to work properly**.

Have you carefully read my comments in #3871-46 and #3871-49?  I think you are spending time worrying about a problem that is not valid.  If I am misunderstanding, please help correct my mis-perception.

#### #66 - 10/21/2019 06:19 PM - Sergey Ivanovskiy

Greg Shah wrote:

> Sergey Ivanovskiy wrote:
>
> > At this moment I suppose now that PostgreSQL database has been configured to have UTF-8 encoding. If another one-byte charset encoding was set, then Hibernate insert queries could fail because UTF-8 was used by Hibernate to wrap native strings. I don't know how to set up Hibernate, if it is possible, to work with another one-byte charset. Although this query SELECT character_set_name FROM information_schema.character_sets; returns UTF-8 correctly. The actual encoding is supposed to be UTF-8. For H2 there are no documentation how to set its encoding to one-byte charset. For this database type we can use UTF-8. MSSQL has complex collation settings and I didn't find the documentation that could help to map some collation, for an example, SQL_Latin1_General_CP1_CI_AS, to some standard code page. The documentation states that MS SQL can support multi encoding for a one database and even for a one table and UTF-8 data is supported fully with new MS SQL version. Please see https://docs.microsoft.com/en-us/sql/relational-databases/collations/collation-and-unicode-support?view=sql-server-ver15#utf8 Committed revision 11347 changed logic for H2 and MS SQL. Please review and help with ideas if you know ways to solve these described issues.
>
> What case are you describing? It is invalid to have a database with strings encoded as ENCODING1 and then trying to work with that database as if it was ENCODING2. It doesn't matter what the actual encoding types are. If you try to use a different encoding than the one set for the database, it won't work properly **and we don't ever expect it to work properly**.

No, it is not the case that was described. I considered the case when DATABASE_ENCODING is not the same as INPUT_ENCODING and this encoding is one byte encoding. I described that internal Hibernate encoding of insert queries is UTF-8. Thus one byte input encoding will be converted into one byte database encoding and then into UTF-8 by Hibernate but the last encoding is not valid for one byte database encoding.

> Have you carefully read my comments in #3871-46 and #3871-49? I think you are spending time worrying about a problem that is not valid. If I am misunderstanding, please help correct my mis-perception.

Yes, this task is to import data in the case when DATABASE_ENCODING is not the same as INPUT_ENCODING.
The second issue that I tried to describe how to get DATABASE_ENCODING from the current database session. Now this task has only a solution for PostgreSQL. H2 has unicode encoding and it is not documented if it is possible to change its encoding. I guess that it uses UTF-8 by tests. MS SQL uses collations instead of encodings to set up what data can be stored in.

#### #67 - 10/22/2019 11:07 AM - Sergey Ivanovskiy

The part of this issue is related to support DBCODEPAGE function because answering this question gets the runtime implementation for this 4GL function.

#### #68 - 10/22/2019 11:10 AM - Sergey Ivanovskiy

I found that 4GL supports aliases for charsets that are not supported by java.nio.charset.Charset.forName. For an example, 1252. It needs to map all them into known encoding names.
https://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html

#### #69 - 10/22/2019 11:33 AM - Greg Shah

> No, it is not the case that was described. I considered the case when DATABASE_ENCODING is not the same as INPUT_ENCODING and this encoding is one byte encoding. I described that internal Hibernate encoding of insert queries is UTF-8. Thus one byte input encoding will be converted into one byte database encoding and then into UTF-8 by Hibernate but the last encoding is not valid for one byte database encoding.

I don't think this is correct.  When we read the .d files, we MUST set the CPSTREAM (which is the INPUT_ENCODING) to the same value as the .d files use.  This will convert the string contents in the .d into the default JVM encoding (which define the in-memory strings).  This is often UTF-8.

When we write any string data into the database, it must be converted to the DATABASE_ENCODING.  That ensures that the string data in the database has the correct encoding.  We either must read the database encoding automatically OR we must force the user to specify it when running the import worker.  Another option is to set the default JVM encoding to match the database encoding.

As long as the data in the INPUT_ENCODING can be converted to the default JVM encoding AND the data in the default JVM encoding can be converted to the DATABASE_ENCODING, then there will be no errors and the process will work properly.  If either of these conversions cannot occur, then we must raise an error and abort the import.

#### #70 - 10/22/2019 11:35 AM - Greg Shah

Sergey Ivanovskiy wrote:

> I found that 4GL supports aliases for charsets that are not supported by java.nio.charset.Charset.forName. For an example, 1252. It needs to map all them into known encoding names.
> https://docs.oracle.com/javase/7/docs/technotes/guides/intl/encoding.doc.html

We already handle this and much of the codepage conversion support in I18nOps.

#### #71 - 10/23/2019 05:57 AM - Sergey Ivanovskiy

I used the similar diff to the following one in order to setup the UTF-8 database with INPUT_ENCODING 1252 (WINDOWS-1252).

```
=== modified file 'src/com/goldencode/p2j/schema/ImportWorker.java'
--- src/com/goldencode/p2j/schema/ImportWorker.java    2019-10-17 20:27:02 +0000
+++ src/com/goldencode/p2j/schema/ImportWorker.java    2019-10-22 09:55:26 +0000
@@ -3169,6 +3169,22 @@
          encoding    = getMetadata("cpstream");    // eg. ISO8859-15
          // cc = new CharsetConverter(encoding);

+         boolean unsupported = false;
+
+         try
+         {
+            unsupported = !Charset.isSupported(encoding);
+         }
+         catch(IllegalCharsetNameException ex)
+         {
+            unsupported = true;
+         }
+
+         if (unsupported)
+         {
+            encoding = I18nOps.convmap2Java.get(encoding);
+         }
+
          setConvertSource(encoding);

          ldbname     = getMetadata("ldbname");     // eg. p2j_test
```

#### #72 - 11/06/2019 06:25 AM - Sergey Ivanovskiy

*- File GenDumpFiles.java added*

*- File dump_file_template.txt added*

I used this simple java code to generate dump files from dump_file_template.txt

#### #73 - 02/19/2020 02:12 PM - Greg Shah

*- Related to Support #4549: reduce/eliminate installation dependencies added*

#### #74 - 06/23/2020 08:01 AM - Greg Shah

*- % Done changed from 100 to 70*

*- Status changed from Review to WIP*

*- Assignee deleted (Sergey Ivanovskiy)*

Sergey Ivanovskiy wrote:

> I used the similar diff to the following one in order to setup the UTF-8 database with INPUT_ENCODING 1252 (WINDOWS-1252).
> [...]

Although I see some use in your code, I have the following concerns:

- If we don't support the necessary encoding names in I18nOps, then we should add that support into I18nOps. The intention here is for I18nOps to provide all the core functionality and helpers needed to handle I18N. I don't want I18N implementation code exploded into classes across FWD. ImportWorker is primarily about database import and so it should only be a **user** of I18nOps.
- We should never be directly using a map inside I18nOps from ImportWorker. Instead, we need to either:
  - Make a new helper method that calculates the correct encoding name. OR
  - Updates setConvertSource() to handle the encoding name properly. (this one seems best)

#### #76 - 06/23/2020 08:08 AM - Greg Shah

*- Assignee set to Eugenie Lyzenko*

#### #77 - 06/24/2020 12:02 AM - Eugenie Lyzenko

Analyzing the task history. The FWD functionality from my understanding is to have tow separate parts, runtime environment(FWD) and external DB to store the result. Only these two parts can have code page settings that can be the same or different. For FWD we always can know exactly what encoding is currently used. For DB it is not clear. I think we need to add some option inside the imported DB that can always exactly tell us what encoding is used for DB. When two encodings become exactly known we can do the rest via FWD engine. The idea is to keep data inside DB in encoding assigned for DB and work with data inside FWD with encoding defined for FWD.

#### #78 - 06/24/2020 07:05 AM - Greg Shah

This task is not about the runtime environment. This is only about database import.

What matters is that we must know the encoding of the .d files which are are reading. And this encoding must be set as the "conversion" source when we read those files. Since we use the FWD 4GL compatible stream reading code to read the .d files, we should use our standard I18N processing to handle this. The result will be processed inside of a JVM running UTF8 but we must convert the encoding as we read the files.

The JDBC processing already knows the encoding of the target database. I think it encodes it naturally from UTF8 (in the JVM) to the database encoding (which may be UTF8 or something else).

#### #79 - 07/01/2020 01:23 PM - Greg Shah

*- Related to Feature #4723: make it significantly easier to run database import added*

#### #82 - 12/10/2020 01:45 PM - Greg Shah

*- Assignee changed from Eugenie Lyzenko to Igor Skornyakov*

#### #83 - 12/17/2020 01:20 PM - Igor Skornyakov

Sorry, I'm confused. In my understanding, all we need is to use CODEPAGE-CONVERT with UTF-8 as the target codepage. The source codepage can be taken from the .d file footer or as an import command option (if all .d files use the same encoding which I think is a common case). JDBC accepts Java strings as field values and should convert to a database codepage automatically.
Have I missed something?
Thank you.

#### #84 - 12/17/2020 02:10 PM - Greg Shah

- We use our 4GL compatible stream processing to read the encoded .d files during import. These are the com.goldencode.p2j.util.Stream and its subclasses.
- These streams operate by default with the assumption that the files are encoded with the same encoding that is used in the 4GL process, which is referred to in the 4GL as CPINTERNAL.
- Java uses UTF-16 for String data. We also almost always run the JVM with UTF-8 as the encoding. In other words, we treat CPINTERNAL as UTF-8. Reading encoded text into character instances must be able to convert the data into Unicode.
- We can use Stream.setConvertSource(encoding) to set the .d encoding from DataFileReader.processPscHeader(). This will cause the stream to automatically convert the text data into Unicode and provide it as a String. I documented this idea in #3871-7. This will handle the equivalent of CODEPAGE-CONVERT without having to do any extra step.
- From there, I think JDBC already handles the conversion to the database encoding, right? So as long as the character instances have valid UTF-16 Strings, the rest should just work.

I actually think this is a pretty simple solution but perhaps I just don't understand the entire problem.

#### #85 - 12/17/2020 02:38 PM - Igor Skornyakov

Greg Shah wrote:

> - We use our 4GL compatible stream processing to read the encoded .d files during import. These are the com.goldencode.p2j.util.Stream and its subclasses.
> - These streams operate by default with the assumption that the files are encoded with the same encoding that is used in the 4GL process, which is referred to in the 4GL as CPINTERNAL.
> - Java uses UTF-16 for String data. We also almost always run the JVM with UTF-8 as the encoding. In other words, we treat CPINTERNAL as UTF-8. Reading encoded text into character instances must be able to convert the data into Unicode.
> - We can use Stream.setConvertSource(encoding) to set the .d encoding from DataFileReader.processPscHeader(). This will cause the stream to automatically convert the text data into Unicode and provide it as a String. I documented this idea in #3871-7. This will handle the equivalent of CODEPAGE-CONVERT without having to do any extra step.
> - From there, I think JDBC already handles the conversion to the database encoding, right? So as long as the character instances have valid UTF-16 Strings, the rest should just work.
>
> I actually think this is a pretty simple solution but perhaps I just don't understand the entire problem.

Thank you. This is a more detailed and correct description than mine and, in my understanding, it should work. However, at this moment I do not see how the CPSTRIM value is used in the ImportWorker. It is used in the creation of the CharsetConverter instance but this line is commented.

BTW: I understand from the OE documentation that the purpose CODEPAGE-CONVERT function is to make it possible using string literals in a way that doesn't depend on the encoding of the program source file. All other discussions of the codepages I've seen are related to conversion between internal representation of the character data and external world (files, databases, etc.). In this context, it seems logical to use our existing support for streams in the import.

#### #86 - 12/17/2020 03:31 PM - Greg Shah

> However, at this moment I do not see how the CPSTRIM value is used in the ImportWorker. It is used in the creation of the CharsetConverter instance but this line is commented.

It isn't handled today.

#### #87 - 12/17/2020 03:38 PM - Igor Skornyakov

Greg Shah wrote:

> > However, at this moment I do not see how the CPSTRIM value is used in the ImportWorker. It is used in the creation of the CharsetConverter instance but this line is commented.
>
> It isn't handled today.

Why not just implement your approach for now? Ignoring CPSTRIM value is obviously not good, but it will be addressed. Or we already have some real-life use cases that will not be properly handled in this way?
Thank you.

#### #88 - 12/17/2020 03:47 PM - Greg Shah

I think that honoring that value is the easist way to implement my approach.  We have to have that value as input.  Why force the end user to provide it when we already have access to it in the .d file?

#### #89 - 12/17/2020 03:57 PM - Igor Skornyakov

Greg Shah wrote:

> I think that honoring that value is the easist way to implement my approach.  We have to have that value as input.  Why force the end user to

> provide it when we already have access to it in the .d file?

I agree. We parse the footer anyway. However I think I've seen that for some .d files the import reports that the footer was not found. In this case we can either reject such .d file or use an (optional) provided value.

#### #90 - 12/17/2020 04:07 PM - Greg Shah

> I think I've seen that for some .d files the import reports that the footer was not found. In this case we can either reject such .d file or use an (optional) provided value.

Agreed.

#### #91 - 12/18/2020 10:59 AM - Igor Skornyakov

Since I have to get familiar with import, maybe be I will add the population of the word tables with postponed creation of their indexes and triggers? This should be substantially faster.
Thank you.

#### #92 - 12/18/2020 01:14 PM - Greg Shah

Yes, go ahead.

#### #93 - 12/20/2020 12:14 PM - Igor Skornyakov

Implemented CPSTREAM support on import.
Committed to 1587b rev 11873.
Re-working of the word tables' import is still in progress.
BTW: I think we can speed up the import if we will use PreparedStatement.addBatch and reWriteBatchedInserts JDBC URL attribute (when it is supported).
I will test it with word tables and PostgreSQL.

#### #94 - 12/21/2020 07:22 AM - Greg Shah

I prefer for these changes to go into 3821c directly.  Is there a reason they need to be build on top of 1587b?

#### #95 - 12/21/2020 07:44 AM - Igor Skornyakov

Greg Shah wrote:

> I prefer for these changes to go into 3821c directly.  Is there a reason they need to be build on top of 1587b?

The only reason is that we decided to add word tables' related changes. 1587b is a branch of 3821c. If you're OK with these changes I can easily put them to 3821c.

#### #96 - 12/21/2020 07:57 AM - Greg Shah

OK, I understand.  We need Eric to complete the code review of #1587 first.

#### #97 - 12/21/2020 07:59 AM - Igor Skornyakov

Greg Shah wrote:

> OK, I understand.  We need Eric to complete the code review of #1587 first.

Got it, thank you.

#### #98 - 12/21/2020 11:31 AM - Eric Faulhaber

Igor Skornyakov wrote:

> BTW: I think we can speed up the import if we will use PreparedStatement.addBatch and reWriteBatchedInserts JDBC URL attribute (when it is supported).

FWIW, I tried using reWriteBatchedInserts with PostgreSQL in the not-too-distant past and it caused a problem, but we were using it at runtime (and perhaps I was using it improperly). I don't recall whether I had tried it with import.

Constantin, do you remember what was going wrong with this setting? I recall that you reported the problem, and backing this setting out of the JDBC URL resolved the issue. All I can find on this now is #4011-665, but that is not related to what caused us to back out this setting.

#### #99 - 12/21/2020 11:37 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, do you remember what was going wrong with this setting? I recall that you reported the problem, and backing this setting out of the JDBC URL resolved the issue. All I can find on this now is #4011-665, but that is not related to what caused us to back out this setting.

By using reWriteBatchedInserts we will lose any and all information about the number of inserted records.  This will break logic which relies on the value returned by Statement.executeBatch() or Statement.execute() methods.

#### #100 - 12/21/2020 11:43 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Eric Faulhaber wrote:
>
> > Constantin, do you remember what was going wrong with this setting? I recall that you reported the problem, and backing this setting out of the JDBC URL resolved the issue. All I can find on this now is #4011-665, but that is not related to what caused us to back out this setting.
>
> By using reWriteBatchedInserts we will lose any and all information about the number of inserted records. This will break logic which relies on the value returned by Statement.executeBatch() or Statement.execute() methods.

I understand that with this option a batch insert will have "all or nothing" logic. So in the case of failure we can repeat an attempt not in a batch. But I understand that success is much more common.

#### #101 - 01/08/2021 11:12 AM - Igor Skornyakov

My changes seem to cause error on import the customer database:

```
     [java] SEVERE: Dropped record #1 in <table name>.d due to error:  org.postgresql.util.PSQLException: ERRO
R: character with byte sequence 0xef 0xbf 0xbf in encoding "UTF8" has no equivalent in encoding "LATIN1"
```

This is strange as the .d file comyains no records at all:

```
.
PSC
filename=<table>
records=0000000000000
ldbname=<db>
timestamp=2020/06/09-10:22:22
numformat=46,44
dateformat=dmy-1950
map=NO-MAP
cpstream=1252
.
0000000003
```

And I do not see the bytes mentioned in the error message in the hex dump.
Investigating...

#### #102 - 01/08/2021 01:16 PM - Igor Skornyakov

My changes for the codepage support on the database import seem to be not 100% correct - they prevent the normal import of the big customer database. I've temporarily rolled them back.
Committed to 1587b revision 11921.

#### #103 - 01/08/2021 03:33 PM - Eric Faulhaber

Are the code page changes in any way dependent upon the other changes unique to branch 1587b? If not, I would suggest keeping 1587b about the word index support (#1587) only, testing it, and merging it back to 3821c, and handling the code page changes separately. If the code page changes are risky (and so far, it seems to be so), perhaps put these in a new branch, based on 3821c, or else put them directly in 3821c.

OTOH, if you think you are quite close to having a stable solution for the code page support, keep going as you are.

I just don't want to hold up getting the word index support into 3821c, if the code page changes are likely to take a while yet to stabilize. It seems like you could add the SQL Server word index support incrementally either way.

#### #104 - 01/08/2021 03:54 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Are the code page changes in any way dependent upon the other changes unique to branch 1587b? If not, I would suggest keeping 1587b about the word index support (#1587) only, testing it, and merging it back to 3821c, and handling the code page changes separately. If the code page changes are risky (and so far, it seems to be so), perhaps put these in a new branch, based on 3821c, or else put them directly in 3821c.

These changes are independent.

> OTOH, if you think you are quite close to having a stable solution for the code page support, keep going as you are.

I do not think that the problems with code pages are really serious. I just do not want to distract from words right now. It will be clear by the end of the weekend.

> I just don't want to hold up getting the word index support into 3821c, if the code page changes are likely to take a while yet to stabilize. It seems like you could add the SQL Server word index support incrementally either way.

I understand and agree with this approach.

#### #105 - 01/13/2021 09:43 AM - Greg Shah

Please see #1587-229 and following notes for related discussions about code page failures during import.

#### #106 - 01/13/2021 09:57 AM - Igor Skornyakov

Greg Shah wrote:

> Please see #1587-229 and following notes for related discussions about code page failures during import.

Thank you. I'm considering this.

#### #107 - 01/15/2021 10:34 AM - Greg Shah

*- Related to Bug #5085: uppercasing and string comparisons are incorrect for some character sets added*

#### #108 - 04/09/2021 12:15 PM - Eric Faulhaber

*- Assignee changed from Igor Skornyakov to Ovidiu Maxiniuc*

#### #110 - 05/26/2021 10:21 PM - Ovidiu Maxiniuc

*- % Done changed from 70 to 100*

Now the requested text decoder is used when processing dump files.
At the same time, I fixed support for processing really big dump files (in case of >10GB files the footer could not be read correctly).
Enabled multi-byte encoding (the existing CC only allowed a 256-character 'palette').

Bzr revision: 12459.

#### #111 - 05/27/2021 11:53 AM - Greg Shah

Code Review Task Branch 3821c Revision 12459

The changes are a nice improvement.

1. FileStream.getCc() and FileStream.setCc() are unreferenced. We don't use it for converted code either. Is there a reason to leave them behind? I'd prefer to remove them rather than deprecate them.

2. Utils.getCharsetOverride() currently forces the charset to ISO8859-1 if the JVM default charset is not "UTF-8". Recent attempts to change the default encoding have not worked well, so I suspect we can't do that. My point here is that I think we should be using the CPSTREAM here instead of Utils.getCharsetOverride(). All the stream subclasses should be using that. Otherwise we are hard coding our stream input and output to ISO8859-1 for any of the stream code that uses override or cc. That is wrong. This was an existing issue, but since you are fixing the multi-byte support here, I think we need to resolve this too.

3. Please propose what you think should be done with the last 2 usages of the cc in FileStream (read() and writeCh()). I think we should get rid of those.

#### #112 - 05/27/2021 01:09 PM - Ovidiu Maxiniuc

Than you for the quick code review.

I will remove the CharsetConverter accessors. It is a clever way to handle character conversions. It is probably faster than Java's Charset but has a couple of disadvantages: it handles only 8-bit codepages (no multi-byte) and does not support error detection. There is one more usage of this class, in DirStream. Probably it should be replaced there, as well.

Utils.getCharsetOverride() is really strange at a first look, but as you said, the default should be CPSTREAM. This is the value the charset should use instead of "UTF-8" I wrote.

I tried to do the changes so that they will affect only the import. The FileStream is used in other places as well. The setTextMode() method is called only from ImportWorker, all the rest of usages will see the class as unchanged, including writing support. The class is not in a finished state: it is stable but there is still some work to do here. There are two binary modes, with apparently different semantics. Most likely they should be unified: start as a CPSTREAM text by default, use setBinary() to switch modes. This is accessible from ABL code. Only ImportWorker needs access to special setTextMode() method which specifies the encoding, too.

The read() method is almost done. In text mode the next char is to be returned, in binary mode I think the next byte. Have to test to see exactly.

Clearly, writing in text mode must use the configured charset and allow output of multi-byte characters. Again, I do not know what is the semantic of this method in binary mode. From the existing code it looks like the text mode fixes the EOLN terminator on-the-fly.

#### #113 - 06/10/2021 10:45 PM - Ovidiu Maxiniuc

The revision 12502 of 3821c handles the issues and suggestions from the ode review:

- the CharsetConverter which was limited to a single byte character encoding was replaced by Java Charset;
- CPSTREAM is the default CP for all streams. The last resort is ISO8859-1, not UTF-8.
- the import process can be configured with a default CP for each imported database in case the .d files do not end with a parsable PSC footer;
- added full-stream multibyte support (in r12459 the footer was read in binary/Latin-1 encoding, which was incorrect);
- improved support for BOMs: if one is detected, this takes precedence to default encoding from configuration file.

At this moment I do not know any issue left for this tracker. The invalid character handling will be implemented in other tracker.

#### #114 - 06/11/2021 07:10 AM - Greg Shah

Code Review Task Branch 3821c Revision 12502

The changes are good.

Isn't there still an issue with calling Configuration.isRuntimeConfig() on the client side from the Stream subclasses?  I thought we saw a regression related to that.

#### #115 - 06/11/2021 04:54 PM - Ovidiu Maxiniuc

Indeed, the usage of static configuration on client side (added in r12499) was incorrect so was reverted in 12502.
In fact there were two regression fixed there. The other was related to reading from DirStream. This is a particular case of a stream and my character-encoding error checking procedure I used for normal files didn't behave properly so that code was disabled and will be replaced when the invalid character handling will be implemented.

If you were referring to the last item in note 113, then the configuration file is used used by the ImportWorker. This is a server-side utility and it is allowed to access it for initialization of its static default values.

#### #116 - 06/14/2021 02:57 PM - Greg Shah

*- Status changed from WIP to Closed*

### Files

| | | | |
|---|---|---|---|
| strings_WINDOWS_1251.d | 183 Bytes | 10/08/2019 | Sergey Ivanovskiy |
| 3871_1.patch | 5.72 KB | 10/09/2019 | Sergey Ivanovskiy |
| strings_UTF_8.d | 181 Bytes | 10/10/2019 | Sergey Ivanovskiy |
| 3871_2.patch | 8.13 KB | 10/10/2019 | Sergey Ivanovskiy |
| GenDumpFiles.java | 3.73 KB | 11/06/2019 | Sergey Ivanovskiy |
| dump_file_template.txt | 163 Bytes | 11/06/2019 | Sergey Ivanovskiy |