Base Language - Feature #2135

implement COPY-LOB language statement

04/25/2013 04:39 PM - Greg Shah

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

Description

Related issues:

Related to Base Language - Feature #1635: implement MEMPTR/RAW support	**Closed**	**01/19/2013**	**05/03/2013**
Related to Base Language - Feature #4768: finish copy-lob support	**Closed**		

History

#1 - 05/28/2013 09:18 AM - Greg Shah

An early form of COPY-LOB support was created in #1635. It is only useful for testing purposes and it is not complete in function nor does it have proper error handling and options.

#2 - 03/23/2016 04:52 PM - Greg Shah

- Target version deleted (Milestone 12)

#3 - 10/20/2018 09:05 AM - Greg Shah

- Estimated time deleted (40.00)

At this time, the following source/target pairs have basic support (see com.goldencode.p2j.util.LargeObjectOps):

```
From           To
------------   ------------
MEMPTR         LONGCHAR
LONGCHAR       MEMPTR
File           MEMPTR
MEMPTR         File
File           LONGCHAR
LONGCHAR       File
```

Full error handling and compatibility is not yet implemented for these.

#4 - 10/20/2018 09:09 AM - Greg Shah

As phase 2 of development, please add basic support for this set of source/target pairs:

```
From            To
------------    ------------
CLOB            LONGCHAR
LONGCHAR        CLOB
POLY            File
BLOB            File
CLOB            MEMPTR
LONGCHAR        LONGCHAR
```

The POLY case is a BUFFER-VALUE reference to a CLOB or BLOB field.

This set is based on usage in a customer POC.

#5 - 11/25/2018 10:36 AM - Greg Shah

As phase 3 of development, the following will all be needed:

```
From                 To
-----------------    ------------
BLOB                 MEMPTR
BLOB                 LONGCHAR
BLOB                 POLY
BLOB                 DB_REF_NON_STATIC
BLOB                 BLOB
DB_REF_NON_STATIC    BLOB
MEMPTR               BLOB
MEMPTR               CLOB
MEMPTR               DB_REF_NON_STATIC
DB_REF_NON_STATIC    MEMPTR
DB_REF_NON_STATIC    POLY
CLOB                 File
CLOB                 CLOB
File                 BLOB
File                 POLY
File                 DB_REF_NON_STATIC
POLY                 MEMPTR
POLY                 LONGCHAR
MEMPTR               POLY
MEMPTR               MEMPTR
LONGCHAR             POLY
LONGCHAR             BLOB
```

The POLY cases are BUFFER-VALUE references to CLOB or BLOB fields. The DB_REF_NON_STATIC is really the same thing.

This set is based on usage in the post-POC customer project.

At this point it just doesn't make sense to implement individual signatures for COPY-LOB. The correct approach is to implement a common interface in each of the possible source/target types, then we can implement a small set of methods where the source and target are always instances of that interface and only the extra parameters that may be needed will cause additional signatures to be added.

In this phase, COPY-LOB should have complete support and full compatibility including error handling.

#6 - 12/13/2018 02:18 AM - Eric Faulhaber

- Assignee set to Eric Faulhaber

#7 - 12/15/2018 03:58 AM - Eric Faulhaber

- Status changed from New to WIP

Task branch 3750a/11334 includes basic support for copying between all combinations of CLOB, BLOB, longchar, and memptr; however, without honoring code page settings. I have not tested POLY and DB_REF_NON_STATIC. I reworked the LargeObjectOps.copy method variants to use the new LargeObject interface instead of the BinaryData and longchar classes.

#8 - 12/16/2018 08:12 PM - Eric Faulhaber

- % Done changed from 0 to 50

Branch 3750a/11356 adds conversion support for DB_REF_NON_STATIC and POLY on either side of the copy, in combination with all previously supported types. The only type of copy which is not currently supported is FILE <--> FILE, which works in the 4GL. We currently convert this to LargeObjectOps.readFromFile, which is not correct (and won't compile, because there is no form of that API which takes a file on both sides).

As part of this update, I've refactored the LargeObjectOps API to reduce the number of the read/write file methods considerably. This was done by substituting LobFile for the file name parameters (which could be character or String previously), and LargeObject for the data parameters (which could be BinaryData and longchar previously).

Note that the runtime support for COPY-LOB is not fully functional; specifically, the read/write file methods were not tested after refactoring, and there are some known issues that need to be addressed (see TODOs in the code).

The ultimate goal is to refactor LargeObjectOps further, such that LobFile implements LargeObject. This will allow all forms of COPY-LOB to convert to some variant of the LargeObjectOps.copy method, so that we can get rid of the read/write file methods and support FILE <--> FILE naturally.

#9 - 12/21/2018 04:23 AM - Constantin Asofiei

Eric, is this copy-lob from v1 to v2 overlay at idx. version of COPY-LOB supported?

#10 - 12/21/2018 08:32 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric, is this copy-lob from v1 to v2 overlay at idx. version of COPY-LOB supported?

In my recent round of changes, I did not (intentionally) change the handling of any options parameters (like overlay at idx). I was just refactoring the API to reduce the possible types of source and target objects, so that more source/target combinations are supported without exploding the number of method variants.

Hopefully, I did not break conversion of this particular set of options.

#11 - 01/03/2019 01:11 PM - Eric Faulhaber

Constantin Asofiei wrote:

> Eric, is this copy-lob from v1 to v2 overlay at idx. version of COPY-LOB supported?

Ovidiu added runtime stubs for this scenario in 3750a. Runtime implementation is still needed.

#12 - 05/06/2019 01:57 AM - Marian Edu

Just to be clear on this one, what is already tested and what do we need to test further on it? Just the CLOB/BLOB (buffer field) options with all other variants or parts of that are already covered as well?

#13 - 05/07/2019 08:31 AM - Greg Shah

Marian Edu wrote:

> Just to be clear on this one, what is already tested and what do we need to test further on it? Just the CLOB/BLOB (buffer field) options with all other variants or parts of that are already covered as well?

For this we need complete coverage. We should make sure to test each possible type as both source and target:

```
FILE
BLOB
CLOB
MEMPTR
LONGCHAR
::
return value from DYNAMIC-FUNCTION() or DYNAMIC-INVOKE()
```

The last two are special in that the type is "polymorphic" (POLY). In other words, we do not know the type until runtime. The actual type is one of the others such as :: returning a CLOB field or BLOB field. These POLY cases cause problems for any code that must know types at conversion time. We must implement runtime behavior to handle them properly.

In addition to testing all of the types, please test the different options:

- FROM
  - STARTING AT n
  - FOR len
- TO
  - OVERLAY AT x
  - OVERLAY AT x TRIM
  - APPEND
- encoding
  - NO-CONVERT
  - CONVERT SOURCE charset TARGET charset
    - charset is same as the default cp
    - charset is not the same as default cp
    - which default codepage is used here? cpstream, cpinternal or does it differ based on which source type and target type is used? for example, is a codepage fixed longchar honored?

As usual, we need to have tests that show the different error cases and boundary conditions. This should include issues such as missing files, invalid filenames, unknown value, unallocated memptr, empty memptr, negative or 0 AT positions, negative or 0 FOR length, invalid codepage names, unavailable buffers, too much data to copy.

#14 - 05/17/2019 07:37 AM - Constantin Asofiei

There is a case where the source or target in a COPY-LOB is POLY - in this case, we have a SourceLob(BaseDataType) attempt - but this fails, even after adding the BDT c'tor (because for non-bdt case, the value is both LargeObject and BaseDataType, and the compile gets confused, as the call becomes ambiguous.

What is the correct way to solve this?

#15 - 05/17/2019 09:32 AM - Eric Faulhaber

What is the test case? I tested with several forms of POLY. For example:

```
define temp-table tt no-undo field f1 as blob.
define buffer b-tt for tt.
define var v1 as memptr.
define var hrec as handle.
define var hfld as handle.
...
assign
   hrec = buffer b-tt:handle
   hfld = hrec:buffer-field("f1").
...
copy-lob hfld:buffer-value to v1.
copy-lob v1 to hfld:buffer-value.
```

...with 3751a/11314 converts to:

```
Tt_1_1.Buf tt = TemporaryBuffer.define(Tt_1_1.Buf.class, "tt", "tt", false, false);
Tt_1_1.Buf bTt = TemporaryBuffer.define(tt, "bTt", "b-tt");
...
memptr v1 = UndoableFactory.memptr();
handle hrec = UndoableFactory.handle();
handle hfld = UndoableFactory.handle();
...
hrec.assign(bTt);
hfld.assign(hrec.unwrapBuffer().bufferField("f1"));
...
new LobCopy(new SourceLob((LargeObject) hfld.unwrapBufferField().value()), new TargetLob(v1)).run();
new LobCopy(new SourceLob(v1), new TargetLob((LargeObject) hfld.unwrapBufferField().value())).run();
```

In this case, we cast to LargeObject.

Likewise,

```
copy-lob hrec::f1 to v1.
copy-lob v1 to hrec::f1.
```

...converts to:

```
new LobCopy(new SourceLob((LargeObject) hrec.unwrapDereferenceable().dereference("f1")), new TargetLob(v1)).ru
n();
new LobCopy(new SourceLob(v1), new TargetLob((LargeObject) hrec.unwrapDereferenceable().dereference("f1"))).ru
n();
```

#16 - 05/17/2019 09:34 AM - Constantin Asofiei

The case is copy-lob b to h:buffer-field(tt1.f1):buffer-value.

BTW, casting to LargeObject is not safe in the long term - as you don't know what the POLY will be, at compile time. And if is something other than LargeObject, runtime will fail badly.

#17 - 05/17/2019 09:38 AM - Greg Shah

> And if is something other than LargeObject, runtime will fail badly.

True. It is my understanding that only CLOB and BLOB fields will work. I guess everything else generates a runtime error. This should be included in the testcases.

#18 - 05/17/2019 09:39 AM - Eric Faulhaber

Constantin Asofiei wrote:

> The case is copy-lob b to h:buffer-field(tt1.f1):buffer-value.

What are the data types of b and tt1.f1 in this case?

#19 - 05/17/2019 09:41 AM - Eric Faulhaber

I did not actually add the casting in my recent work; this was there already with the old API. What is the preferred approach for POLY parameters elsewhere in FWD?

#20 - 05/17/2019 09:41 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin Asofiei wrote:
>
> > The case is copy-lob b to h:buffer-field(tt1.f1):buffer-value.
>
> What are the data types of b and tt1.f1 in this case?

b is blob and tt1.f1 is char.

#21 - 05/17/2019 09:46 AM - Constantin Asofiei

Eric Faulhaber wrote:

> I did not actually add the casting in my recent work; this was there already with the old API. What is the preferred approach for POLY parameters elsewhere in FWD?

Usually a c'tor or method overload which accepts a BDT and the runtime decides what to do with the value.

#22 - 05/17/2019 09:52 AM - Eric Faulhaber

Constantin Asofiei wrote:

> b is blob and tt1.f1 is char.

So b is a field? How is it defined?

#23 - 05/17/2019 09:54 AM - Greg Shah

All POLY cases will be BDT, right? There is no other possibility for a field (:: or BUFFER-FIELD:BUFFER-VALUE()) or the return value from a function (DYNAMIC-FUNCTION()) or method return value (DYNAMIC-INVOKE()). Can we always cast POLY to BDT?

The only questions I have:

- What does BUFFER-FIELD:BUFFER-VALUE() or :: return for CLOB or BLOB? Is this a BDT?
- What happens in DYNAMIC-INVOKE() for a VOID return type? I guess the 4GL generates an error in this case.

#24 - 05/17/2019 09:55 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin Asofiei wrote:
>
> > b is blob and tt1.f1 is char.
>
> So b is a field? How is it defined?

Sorry, b is a memptr, not a field.

#25 - 05/17/2019 09:59 AM - Constantin Asofiei

Greg Shah wrote:

> All POLY cases will be BDT, right? There is no other possibility for a field (:: or BUFFER-FIELD:BUFFER-VALUE()) or the return value from a function (DYNAMIC-FUNCTION()) or method return value (DYNAMIC-INVOKE()). Can we always cast POLY to BDT?

Well, a POLY can always be an extent value (Java array)... that is another whole can of warms to worry about.

> The only questions I have:
>
> - What does BUFFER-FIELD:BUFFER-VALUE() or :: return for CLOB or BLOB? Is this a BDT?

CLOB and BLOB in FWD are BDTs.

> - What happens in DYNAMIC-INVOKE() for a VOID return type? I guess the 4GL generates an error in this case.

Yes, you get a Invalid use of dynamic invoke of VOID method '%s' in an expression.

#26 - 05/17/2019 10:18 AM - Greg Shah

> Well, a POLY can always be an extent value (Java array)... that is another whole can of warms to worry about.

It is my understanding that you cannot have an extent for CLOB or BLOB fields. Are any other kinds of fields supported in COPY-LOB? I didn't know it was possible to pass a CHARACTER type.

If a function or method returns an extent (for LONGCHAR or MEMPTR)... is there a failure in COPY-LOB?

In other words, is there any working case where an extent can be passed to COPY-LOB, or do we just have to handle the error processing?

But your point is well taken that an extent can be passed at runtime, so do we need the signature to be Object for the POLY case?

#27 - 05/17/2019 10:22 AM - Constantin Asofiei

Greg Shah wrote:

> Well, a POLY can always be an extent value (Java array)... that is another whole can of warms to worry about.

It is my understanding that you cannot have an extent for CLOB or BLOB fields. Are any other kinds of fields supported in COPY-LOB? I didn't know it was possible to pass a CHARACTER type.

The target is h:buffer-field(tt1.f1):buffer-value., the BUFFER-VALUE, which is POLY. tt1.f1 was there just to get the field's name.

> But your point is well taken that an extent can be passed at runtime, so do we need the signature to be Object for the POLY case?

I would make it Object. But many of current POLY APIs in FWD are not extent-proof - they will fail.

#28 - 05/17/2019 10:38 AM - Eric Faulhaber

Constantin Asofiei wrote:

> Sorry, b is a memptr, not a field.

Please post the full test case.

#29 - 05/17/2019 10:40 AM - Greg Shah

> But your point is well taken that an extent can be passed at runtime, so do we need the signature to be Object for the POLY case?

I would make it Object.

Or we could add a second POLY set of signatures for BDT[]. You can't return an array of anything else in these case, so that would limit it and make it less likely to be unexpectedly used later.

> But many of current POLY APIs in FWD are not extent-proof - they will fail.

Yes. Let's work on the solution for COPY-LOB and then we will consider retrofitting it to the other use cases later.

#30 - 05/17/2019 03:04 PM - Eric Faulhaber

I'm implementing SourceLob and TargetLob c'tors which accept Object to handle the POLY case.

#31 - 05/17/2019 03:38 PM - Eric Faulhaber

Eric Faulhaber wrote:

> I'm implementing SourceLob and TargetLob c'tors which accept Object to handle the POLY case.

The fix is in 3751a/11324. This also removes the casting to LargeObject from the other POLY cases which already were compiling.

The test for this particular case:

```
define temp-table tt no-undo field f1 as clob field name as char init "f1".

define var v1 as memptr.
define var hrec as handle.

set-size(v1) = 3.
put-byte(v1, 1) = ASC("A").
put-byte(v1, 2) = ASC("B").
put-byte(v1, 3) = ASC("C").

hrec = temp-table tt:handle.
hrec = hrec:default-buffer-handle.
hrec:buffer-create().

copy-lob v1 to hrec:buffer-field(tt.name):buffer-value.
copy-lob hrec:buffer-field(tt.name):buffer-value to v1.

set-size(v1) = 0.
```

The copy-lob statements convert to:

```
new LobCopy(new SourceLob(v1), new TargetLob(hrec.unwrapBuffer().bufferField(tt.getName()).unwrapBufferField()
.value())).run();
new LobCopy(new SourceLob(hrec.unwrapBuffer().bufferField(tt.getName()).unwrapBufferField().value()), new Targ
etLob(v1)).run();
```

#32 - 07/03/2019 12:11 PM - Eric Faulhaber

Do we have an API to get the size of a remote file? I'm using a stream (via StreamFactory.openFileStream) to be able to write to the file, but I need to know its size beforehand, for an error check.

#33 - 07/03/2019 12:16 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Do we have an API to get the size of a remote file? I'm using a stream (via StreamFactory.openFileStream) to be able to write to the file, but I need to know its size beforehand, for an error check.

Never mind, just found FileStream.getLen.

#34 - 07/07/2019 04:50 PM - Eric Faulhaber

Please review the first pass runtime implementation of COPY-LOB in branch 4056a. The relevant changes are in revisions 11329-11333.

The following issues still need attention:

- TODOs in code, primarily those where errors need to be handled with a more appropriate error code and message.
- Check undo behavior (if any) with test cases.
- Investigate flushing behavior (e.g., copy to a CLOB, then a FIND with a separate buffer for the same record does not pick up the change).
- Determine correct code page for a database LOB field (currently defaults to longchar implementation for CLOB, -cpinternal for BLOB).
- Implement byte-order-marker (BOM) handling for files.
- Remove LobFile from version control (no longer used).
- Remove javadoc references to LargeObjectOps (no longer used).

Open questions:

- What is the structure/size/layout of the BOM for file input/output?
- What is a "CLOBCP" mentioned in developer reference (in the matrix of code page handling in the COPY-LOB statement section)? How does it relate to a "CLOBDB"?

#35 - 07/07/2019 09:55 PM - Greg Shah

Code Review Task Branch 4056a Revisions 11329-11333

I'm OK with the changes.

#36 - 07/17/2019 04:46 PM - Constantin Asofiei

Eric, when COPY-LOB uses as target an uninitialized memptr (unknown) or if its size doesn't fit the source, then the memptr is automatically resized.

I'm fixing this.

#37 - 08/08/2019 05:12 PM - Constantin Asofiei

In real-life applications, we import 4GL lob files in the 100s of MB range; I assume there would be a lazy approach on loading these? If so, it might be tricky with Hibernate, if there is no session around, at the time that is accessed.

#38 - 11/25/2019 04:10 AM - Constantin Asofiei

Eric, what is the use case documented like this in longchar.write:

```
// complex case: overlay data over existing data, possibly extending the length
```

I can't find a recreate for this, neither with clob or longchar.

#39 - 11/25/2019 01:54 PM - Eric Faulhaber

The idea was that you have an existing longchar instance which has data in it and a COPY-LOB (overlay) operation sets new data into it which is longer than the original. I thought I tested this in testcases/uast/lobs.p, but perhaps not. Are you finding it works differently than I implemented it?

#40 - 11/25/2019 01:56 PM - Constantin Asofiei

Eric Faulhaber wrote:

> The idea was that you have an existing longchar instance which has data in it and a COPY-LOB (overlay) operation sets new data into it which is longer than the original. I thought I tested this in testcases/uast/lobs.p, but perhaps not. Are you finding it works differently than I implemented it?

I have a use case where the longchar has something like xxxxxxxxxxxxxxxxxxxx and copy-lob wants to copy value 123 to it - and instead of 123, FWD ends up with 123xxxxxxxxxxxxxxxxx (i.e. it overlays the data over the existing content). Have you seen anything like this in 4GL tests, where the data is overlayed?

#41 - 11/25/2019 03:38 PM - Constantin Asofiei

I think I understand now - the overlay option (which is 'active' in FWD only if offset is non-null) is not treated properly. Also, I see checks like value == data, where both are i.e. String or byte[] - what's the reason here, comparing references?

#42 - 11/25/2019 04:07 PM - Eric Faulhaber

Constantin Asofiei wrote:

> I think I understand now - the overlay option (which is 'active' in FWD only if offset is non-null) is not treated properly.

Thanks, I was struggling to remember...

> Also, I see checks like value == data, where both are i.e. String or byte[] - what's the reason here, comparing references?

I suspect that was meant as an optimization to preempt unnecessary work, but I'm not sure where you are looking. To where are you referring, exactly?

#43 - 11/25/2019 04:53 PM - Constantin Asofiei

Eric Faulhaber wrote:

> I suspect that was meant as an optimization to preempt unnecessary work, but I'm not sure where you are looking. To where are you referring, exactly?

This code in clob (similar exists in other LOB classes):

```
   public void write(String data, int offset, boolean trim)
   {
      int dataLen;

      if (data == null || (dataLen = data.length()) == 0 && !trim || offset == 0 && value == data)
      {
```

#44 - 11/25/2019 04:58 PM - Eric Faulhaber

This was meant to be a quick exit if there was nothing to do. Note the rest of the block:

```
      if (data == null || (dataLen = data.length()) == 0 && !trim || offset == 0 && value == data)
      {
         // TODO: is this right for the null case? Can data be null? If so, set to unknown?

         // nothing to write or data already is stored in this object (we don't do a deep equals
         // check because it could be expensive and the common case is likely a full assignment
         // anyway, so don't take the hit here, just quickly check for object reference equality)
         return;
      }
```

Did you find a flaw in this approach?

#45 - 11/26/2019 01:49 AM - Constantin Asofiei

The value == data I don't think can ever be true, as when we assign a byte array, we copy it to another instance. Also, a String will "be equal" once assigned only if is intern'ed, right? That's why I'm a little confused when this equality can be true.

#46 - 11/26/2019 12:14 PM - Eric Faulhaber

Hm. That check may have survived from an earlier implementation. I think you are right.

#47 - 11/26/2019 12:18 PM - Eric Faulhaber

Well, then again, look at Text.assign(Text value). It stores the String object inside the Text passed to it, not the Text instance itself. So, the String object within could be the same instance, whether it was intern'd or not.

#48 - 11/26/2019 12:49 PM - Constantin Asofiei

Eric Faulhaber wrote:

> Well, then again, look at Text.assign(Text value). It stores the String object inside the Text passed to it, not the Text instance itself. So, the String object within could be the same instance, whether it was intern'd or not.

For byte[] instances, I don't see how these can ever be the same ref. But I'll leave these in place.

Now I'm working on copy-lob from src to tar [overlay at n [trim]] cases, with these variations:

- unknown
- zero length
- 1 length
- "abc"
- length equal to target
- length greater than target
- length smaller than target

multiplied by:

- longchar to longchar
- longchar to clob
- longchar to blob
- clob to clob
- clob to longchar
- blob to blob
- blob to longchar

There are lots of issues, mainly for unknown and non-overlay. I haven't reached trim yet, will add these once I have the other cases working

#49 - 11/26/2019 01:25 PM - Eric Faulhaber

These are all runtime bugs, right? Both the conversion and runtime were meant to handle these permutations, but runtime testing was light.

#50 - 11/26/2019 05:07 PM - Constantin Asofiei

Eric Faulhaber wrote:

> These are all runtime bugs, right? Both the conversion and runtime were meant to handle these permutations, but runtime testing was light.

Yes, the fixes are for runtime.

And there's a stupid issue with COPY-LOB, clob source and OVERLAY - if the clob field was initialized to something before being set to unknown, then the target will be set to empty string instead of unknown... see this:

```
def temp-table tt1 no-undo field fc1 as clob field fc2 as clob field fb1 as blob field fb2 as blob.
create tt1.
// tt1.fc1 = "". // if this line exists, then the target will be set to "", and not unknown
tt1.fc1 = ?.
tt1.fc2 = ?.

copy-lob from tt1.fc1 to tt1.fc2 overlay at 1.

message string(tt1.fc2). // this can be either ? or empty string, depending on the line above
```

#51 - 11/26/2019 05:34 PM - Eric Faulhaber

First CLOB initialization is sticky, even if followed by a later assignment? Does that hold if the "" and ? initializations are reversed? That is, if tt1.fc1 = ? is followed by tt1.fc1 = "", will the COPY-LOB copy unknown value to the target?

#52 - 11/26/2019 05:36 PM - Constantin Asofiei

Eric Faulhaber wrote:

> First CLOB initialization is sticky, even if followed by a later assignment?

Not the initialization per-se - what matters is that the clob 'was prior initialized to something not-unknown, before setting it back to unknown'.

> Does that hold if the "" and ? initializations are reversed? That is, if tt1.fc1 = ? is followed by tt1.fc1 = "", will the COPY-LOB copy unknown value to the target?

No. See above - what matters is that the clob was at some point initialized 'to something'.

#53 - 12/03/2019 05:05 PM - Constantin Asofiei

The changes are in 3809e rev 11414. There are still some issues not addressed (see the TODOs in the code)m and #2135-52.

#54 - 01/03/2020 01:52 AM - Marian Edu

Eric Faulhaber wrote:

- What is the structure/size/layout of the BOM for file input/output?

There is no special structure for BOM used by Progress, it's just a single unicode character at the beginning of the file. As the doc say for input is interpreted and then skipped from import (if present), for output is written unless the output codepage is UTF-8.

```
The Byte Order Mark (BOM) or the encoding declaration is processed when a COPY-LOB operation reads from or wri
tes to a FILE. For example, when a COPY-LOB statement copies data from a FILE (with a BOM) to a CLOB, the stat
ement identifies and translates file content based on the BOM character but ignores the BOM character as part
of the copy operation. And, when a COPY-LOB operation copies data from a CLOB to the beginning of a FILE, the
statement prepends the BOM character, except when the BOM can be assumed; that is, the code page of the FILE t
arget is UTF-8.
```

- What is a "CLOBCP" mentioned in developer reference (in the matrix of code page handling in the COPY-LOB statement section)? How does it relate to a "CLOBDB"?

In Progress a CLOB field can have it's own codepage set (COLUMN-CODEPAGE) when it is referred as "CLOBCP" or it will defaults to the database codepage for database tables or temp-table codepage - which is the internal codepage (CPINTERNAL) in which case it is referred as "CLOBDB". If the clob field has a COLUMN-CODEPAGE set then CLOBCP is that codepage, otherwise if from a database table it will have the database's codepage and for temp-table the current client internal codepage and they call that CLOBDB.

#55 - 01/27/2020 01:39 PM - Greg Shah

Marian: 2 followup questions:

1. What is the unicode BOM character that is used?

2. Is it always the same unicode character as the BOM?

#56 - 01/30/2020 08:56 AM - Marian Edu

Greg Shah wrote:

> Marian: 2 followup questions:
>
> 1. What is the unicode BOM character that is used?
>
> 2. Is it always the same unicode character as the BOM?

I'm going to write test cases for this one but what Progress writes as BOM is:
UTF-16LE - FF FE
UTF-16BE - FE FF
UTF-32LE - FF FE 00 00
UTF-32BE - 00 00 FE FF

No BOM is used for UTF-8, the permitted sequence EF BB BF is not used but that doesn't seem to be neither required or recommended. Looks like reading from file using COPY-LOB that optional UTF-8 BOM is not skipped either so the longchar will also contain the BOM. For UTF-16 and UTF-32 the BOM is considered when reading if present and it is stripped out.

If the codepage used for copy-lob - either with CONVERT TARGET option or using CPSTREAM - does not specify the byte order (UTF-16, UTF-32) the BOM is written to the output, if the codepage includes the byte order (UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE) then no BOM is written.

#57 - 01/30/2020 09:37 AM - Greg Shah

Yes, test cases will be very useful. Thanks!

It seems that the 4GL implements a subset of the Byte Order Mark standard. I guess the 4GL does not have support for UTF-7, UTF-1, UTF-EBCDIC or the other possible encodings which were not mentioned in your note. So we can ignore those.

#58 - 07/15/2020 03:07 PM - Greg Shah

- % Done changed from 50 to 100

- Status changed from WIP to Closed

These changes are all in trunk. The remaining work will be tracked in #4768.

#59 - 07/15/2020 03:07 PM - Greg Shah

- Related to Feature #4761: I18N phase 3 added

#60 - 07/15/2020 03:07 PM - Greg Shah

- Related to deleted (Feature #4761: I18N phase 3)

#61 - 07/15/2020 03:07 PM - Greg Shah

- Related to Feature #4768: finish copy-lob support added