Base Language - Feature #1635

implement MEMPTR/RAW support

10/21/2012 11:21 AM - Greg Shah

Status:	Closed	**Start date:**	01/19/2013
Priority:	Normal	**Due date:**	05/03/2013
Assignee:	Greg Shah	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Runtime Support for Server Features		
billable:	No	**version:**	
vendor_id:	GCD		

Description

Subtasks:

Feature # 1963: implement conversion support for MEMPTR/RAW	**Closed**
Feature # 1964: finish runtime support for MEMPTR/RAW	**Closed**

Related issues:

Related to Base Language - Feature #1634: implement full native library (.so ...	**Closed**	**01/14/2013**	**06/06/2013**
Related to Base Language - Feature #2135: implement COPY-LOB language statement	**Closed**		
Related to Base Language - Bug #2375: fix the memptr initialization (in exten...	**Closed**		
Related to Base Language - Bug #2659: memptr in internal procedure address du...	**New**		
Related to Base Language - Bug #2657: appserver agent context reset does not ...	**Closed**		

History

#1 - 10/21/2012 11:22 AM - Greg Shah

The following have highest priority:

Built-in Functions: get-long(), get-pointer-value()
Statements: put-long, put-short, COPY-LOB

These are low priority:

Built-in Functions: get-byte-order(), get-bytes(), get-double(), get-float(), get-int64(), get-short(), get-unsigned-long(), get-unsigned-short(), get-bits()
Statements: set-byte-order, put-bytes, put-double, put-float, put-int64, put-short, put-unsigned-long, put-unsigned-short, set-pointer-value, put-bits

Really low priority:

Built-in Functions: raw()
Statements: raw

Possible undocumented stuff (there are references in the Programming Interfaces book but the referenced func/statement are not documented in the 4GL reference): get-raw() function and the put-raw language statement. These are suggested to be the equivalent of assigning a memptr from a raw (get-raw()) and assigning a raw from a memptr (put-raw).

The conversion rules can be found in convert/builtin_functions.rules and in convert/language_statements.rules.

The runtime code for all of these should go into com/goldencode/p2j/util/BinaryData.java. Follow the pattern of the getByte() and setByte() that is already there. The exceptions: COPY-LOB (we have to figure out where to put this) and get-bits/put-bits should be in integer.java.

#2 - 10/31/2012 01:39 PM - Greg Shah

- Target version set to Milestone 7

#3 - 01/24/2013 05:40 AM - Costin Savin

From what I've seen from documentation 4GL memptr memory can be shared with other variable:
For example according to documentation something like this can be possible:

```
DEFINE VARIABLE mptr AS MEMPTR.

DEFINE VARIABLE mpt AS MEMPTR.

SET-SIZE(mptr) = LENGTH("123456789") + 1.

SET-SIZE(mpt) = LENGTH("123456789") + 1.

PUT-STRING(mptr, 1) = "123456789".

SET-POINTER-VALUE(mpt) = GET-LONG(mptr,3).

MESSAGE GET-STRING(mpt,1).
```

Where get-long is supposed to return a signed 32 bit value at the specified memory location (INTEGER) , this probably means it represents the memory location which can be passed to SET-POINTER-VALUE to get the variable pointing to that memory location.

We can return type integer as specified but if it is used in instructions like SET-POINTER-VALUE which use it to share memory between MEMPTR variables, how will this cope with p2j implementation of memptr.

Does this kind of usage of get-long exist in practice?

Tested this instruction on win 4GL and it blocks probably because of memory problems (even though documentation provides a similar example as good).

#4 - 01/24/2013 07:11 AM - Greg Shah

You are mis-interpreting the documentation.

There are 2 serious problems with your example:

1. You should not use SET-SIZE before calling SET-POINTER-VALUE. We will have to see if it is safe (maybe it is), but what you are doing is you are allocating a buffer of size 10 bytes. Then you are overwriting the pointer to that buffer, making it an "orphan". I hope that the 4GL is smart

enough to de-allocate the original allocated 10 bytes buffer for mpt when SET-POINTER-SIZE is executed, but who knows?

2. SET-POINTER-VALUE is used to force a MEMPTR to point to a buffer allocated for your process by NON-4GL code. It is generally used after you call a DLL and that DLL has returned back a data structure that contains a pointer inside it. The GET-LONG will read 4 bytes directly from the memory location at the 3rd byte of the memory pointed to by mptr. The contents of that mptr are as follows (in hexidecimal):

```
31 32 33 34 35 36 37 38 39 00

      ^
      |
index 3
```

Of course, the 4 bytes starting at index 3 are 0x33343536 but when read using GET-LONG on a little endian system, they will be returned as 0x36353433. In ASCII, this is the text "6543", but when read by GET-LONG it is the decimal number 909456435.

This means you are reading your ASCII text "3456" and treating it like a pointer to real memory. Of course, if it is a pointer to actually allocated memory in your process, it would just be random luck. In all likelihood, it is not allocated memory at all. Of course, as soon as you try to use unallocated memory, the system will have severe problems, since that is an illegal access by the CPU and the process should stop or crash or hang. If you are "lucky" enough to have memory allocated at that address, then reading and writing it will corrupt data in your process, which may destroy your database, some other data or it may stop/crash/hang your process.

Don't do this.

GET-LONG just reads the CONTENTS of the memory. It DOES NOT return back the address of that memory.

SET-POINTER-VALUE needs a VALID address created by other code (in a DLL) and then written into a buffer as DATA. Then you can read those CONTENTS and treat it like an address.

#5 - 01/24/2013 10:33 AM - Costin Savin

I understand now, the correct case to exemplify what I wanted to show looks like this:

```
DEFINE VARIABLE mptr AS MEMPTR.
DEFINE VARIABLE mpt AS MEMPTR.
DEFINE VARIABLE mpts AS MEMPTR.
DEFINE VARIABLE lon AS INT.

SET-SIZE(mptr) =11.
SET-SIZE(mpt) = 11.
PUT-STRING(mptr, 1) = "123456789".

/*get the pointer to mptr as 32 bit integer*/
lon = GET-POINTER-VALUE(mptr).
```

```
/**put the pointer inside mpt var at position 3/
PUT-LONG ( mpt , 3)=lon.

/*set the pointer to the 32 bit integer from position 3 of mpt variable*/
SET-POINTER-VALUE(mpts) = GET-LONG(mpt,3).

/*mpts will point to mptr and contain it's data*/
MESSAGE GET-STRING(mpts,1).
```

GET-POINTER-VALUE returns an integer representing the pointer to that address, when we try to SET-POINTER-VALUE how will it get the integer value it receives to point to the correct mptr variable in Java? Should memptr variables be tracked by a 32 bit id which can be used as "pointer" to access them?

At the moment GET-POINTER-VALUE is implemented as .assign() this will need to be changed but it depends on what approach is best for this problem

#6 - 01/24/2013 11:07 AM - Greg Shah

In this example, you are just making 2 MEMPTR vars point to the same buffer. I don't think you can make one MEMPTR point to a memory location INSIDE another MEMPTR buffer.

That means that in Java, this is just 2 memptr instances that have the SAME REFERENCE to the byte[] value member. In this case, we may have to use a special form of assign() for SET-POINTER-VALUE since normally we duplicate the byte array.

```
At the moment GET-POINTER-VALUE is implemented as .assign()
```

I think you meant SET-POINTER-VALUE here.

```
GET-POINTER-VALUE returns an integer representing the pointer to that address, when we try to SET-POINTER-VALU
E how will it get the integer value it receives to point to the correct mptr variable in Java? Should memptr v
ariables be tracked by a 32 bit id which can be used as "pointer" to access them?
```

If we do the tracking thing, I think it is the value member (the reference to the byte[]) that should be tracked in this manner.

While that would work well for this "internal" case that you have written, your case is not the common case. Usually GET-POINTER-VALUE/SET-POINTER-VALUE are used for dealing with DLL calls. That means that the common case has a **real** 32-bit pointer that is 4 bytes in size and is cast to an integer. We will have to support those cases too. That means that when GET-POINTER-VALUE is called, we will have to back the array using a real memory buffer so that it can be passed to API calls. Or more likely, we will modify our memptr standard approach and simply back it with memory all the time. Maybe we can use something like a subclass of NIO ByteBuffer to do the trick, we will see about that later.

For now:

1. Have GET-POINTER-VALUE emit as a getPointer() instance method in memptr.

2. Have SET-POINTER-VALUE emit as a setPointer() instance method in memptr.

#7 - 01/28/2013 08:56 AM - Costin Savin

Hi, should raw built-in function/statement be present in the conversion support?
I'm asking because I saw it's low priority and it may take some time to implement it as both built-in function and statement

#8 - 01/28/2013 09:44 AM - Greg Shah

No, don't work on RAW and RAW stmt for now.

#9 - 01/28/2013 10:08 AM - Costin Savin

- File cs_upd20130128a.zip added

Submited proposed update which supports conversion for the items on list except copy-lob and raw (merged with the most recent updates), commited the test files used for conversion to bazaar

#10 - 01/28/2013 11:05 AM - Greg Shah

The work is decent. Feedback:

1. I know that I said that get-bits/put-bits should be in integer.java, but looking carefully at the docs, it is clear that this can work on both integer and int64 data. For this reason, the various setBits() and getBits() methods should be in the NumberType superclass. Create an abstract worker method (e,g, protected abstract void setBitsWorker(NumberType, NumberType, NumberType)) that will be implemented as a stub in the integer and int64 classes. The decimal class needs this too, but it will probably just raise an error (to match whatever the 4GL does when you call PUT-BITS/GET-BITS on a decimal).

2. The stubs for setBits/getBits should use "double" instead of "int" for the primitive parameters. The reason is that in the calling code, there may be usage of int, long and double literals and all 3 can be suitably represented inside the 80-bit double.

3. The stubs for setBits/getBits should use NumberType instead of integer for all your non-primitive parameters. The reason is that it can be called with multiple possible inputs (integer and int64 at a minimum) and this reduces the number of variants.

4. The stubs for setBits/getBits cannot call intValue() without some unknown protection. Solve this by just wrapping any double parameters in a new int64((int) double_parm). The real worker can centralize all the unknown value processing before calling intValue(); This must be done for getBits() too.

5. In BinaryData, the javadoc for the byte-order constants is too similar to the 4GL documentation. Please rewrite it in your own words.

6. In BinaryData, make the same NumberType and double parameter type changes as mentioned above. You should be able to eliminate some variants (e.g. NumberType handles all 3 subclasses, so you don't need variants for each one).

#11 - 01/28/2013 01:07 PM - Costin Savin

- File cs_upd20130128b.zip added

Added proposed update

#12 - 01/28/2013 01:43 PM - Greg Shah

Feedback:

1. The update is missing the NumberType.java file.

2. In BinaryData, please use the same approach to unknown value processing as mentioned in the previous feedback for getBits() and setBits(). In other words, the stubs for set*/get* methods cannot call intValue() (et al) without some unknown protection. Solve this by just wrapping any double parameters in a wrapper constructor. The real worker for each setter/getter should be the form that has all wrapper constructors (e.g. setFloat(NumberType, NumberType). That is the place where we will centralize all the unknown value processing before calling intValue().

#13 - 01/29/2013 05:37 AM - Costin Savin

- File cs_upd20130129a.zip added

Added proposed update.

#14 - 01/29/2013 10:31 AM - Greg Shah

Feedback:

1. Please help me understand why your code formatting is still not matching our coding standards. From BinaryData:

```
      if (order!=null)
      {
         byteOrder = order.intValue();
      }
```

At this point, I expect all of your code to be written to the standard from the first moment. It is my impression that you are coding it with little regard to the standard and then you clean it up later. Please do not do this. It takes longer to get the job done and it takes much more of my time to be constantly explaining the same things over and over.

2. Just as above, I have explained on multiple occasions that you CANNOT call intValue() without unknown checking. This code:

```
      if (order!=null)
      {
         byteOrder = order.intValue();
      }
```

should look like this:

```
      if (order == null || order.isUnknown())
      {
         // unknown case, usually this will set the instance to be unknown, but it must
         // simply do whatever the 4GL behavior is
      }
      else
      {
         byteOrder = order.intValue();
      }
```

I expect your future work to take this into account from the first code submission.

3. More code formatting issues (in NumberType):

```
public abstract integer getBitsWorker(NumberType pos , NumberType len);
public abstract void setBitsWorker(NumberType data,NumberType pos,
```

```
                                        NumberType len);
```

should be:

```
public abstract integer getBitsWorker(NumberType pos, NumberType len);
public abstract void setBitsWorker(NumberType data,
                                   NumberType pos,
                                   NumberType len);
```

4. Please do not refer to the data types in the @param or @return portions of the JavaDoc. The javadoc processor will automatically list the type in the documentation. So it is redundant to put your own comment in about the type. And it makes it harder to maintain because when someone changes the type later, the javadoc must also change or else it will be wrong.

5. GET-BITS can return either an integer or an int64, depending on the number of bits requested. This means the getBits()/getBitsWorker() methods must have a signature that returns a NumberType. This also solves the problem with child class references in return types (see 6 below).

6. It is a bad practice to refer to child classes from code inside a parent class. Your setBits(), getBits(), setBitsWorker() and getBitsWorker() are filled with references. Propose a solution to get rid of these and let's discuss. Consider how this addition to NumberType could solve the problem:

```
   /**
    * Creates a new instance of the same type that represents the
    * given value.
    *
    * @return   An new instance that represents the given value.
    */
   public abstract NumberType instantiateNumber(double num);
```

#15 - 01/29/2013 12:29 PM - Costin Savin

1 It is my impression that you are coding it with little regard to the standard and then you clean it up later.
This is because of a very bad habit I've picked up working (for some years), with defined code formatting settings for the editor and only handle any formatting problems at the end by auto-formatting. I'll eliminate any slipping to this habit.

6.
For the method

```
  /**
    * Creates a new instance of the same type that represents the
    * given value.
    *
    * @return   An new instance that represents the given value.
    */
   public abstract NumberType instantiateNumber(double num);
```

If we instantiate the correct NumberType implementation according to the value of num, ii is going to be the same code for integer int64 and decimal.
Wouldn't it be better to create this as a static public method of MathOps?

#16 - 01/29/2013 12:38 PM - Greg Shah

```
If we instantiate the correct NumberType implementation according to the value of num, ii is going to be the s
ame code for integer int64 and decimal.
Wouldn't it be better to create this as a static public method of MathOps?
```

No, that is not the idea here. What this would do is to instantiate the same type as the child class (if called on an integer.class instance, then num is wrapped in new integer(num) but on int64, it is returned as new int64(num). I'm not sure this always solves our problem, but it is the common case.

Another (possibly better) idea: fully handle the unknown processing in the NumberType class and translate everything into primitives. Then the worker methods would only operate on the primitives.

#17 - 01/29/2013 12:58 PM - Costin Savin

I thought that the method will get the type depending on the size/type of the variable.
The second idea seems simpler and more secure.

#18 - 01/29/2013 01:14 PM - Greg Shah

I agree. Go with it.

#19 - 01/29/2013 01:38 PM - Costin Savin

- File cs_upd20130129b.zip added

Added proposed update

#20 - 01/29/2013 02:01 PM - Greg Shah

This looks OK. Regression test this for conversion only. I think the runtime portions should be safe enough to include with other code being runtime tested later.

#21 - 01/30/2013 09:02 AM - Costin Savin

Conversion tests finally passed (I had some problems with ant and antlr failing and it took some time to figure out what was happening and fix it, also some differences because of majic code that was not up to date , but other than that conversion is ok)

#22 - 01/30/2013 10:41 AM - Greg Shah

Check it in to bzr and distribute the update. I will apply it to staging.

#23 - 04/10/2013 02:39 PM - Greg Shah

A more detailed look at NIO ByteBuffer makes it clear that it is not a suitable solution for our problem. In particular, it is based on Buffer which is designed to batch writes and then "flip" or "clear" the buffer and run a batch of reads. This is great for stream processing but really poor for truly random access/intermixed read/write usage.

Because of inherent problems with pinning/duplicate copying of arrays of primitives in JNI, we will not try to implement using GetByteArrayRegion/SetByteArrayRegion or with GetByteArrayElements/ReleaseByteArrayElements. Both of these JNI APIs are really designed for JNI code to temporarily work with Java heap memory. The idea of a memptr is really that the Java code is accessing a native buffer that has a real/fixed memory address AND which can be passed to API calls at the OS level. The JNI methods for accessing arrays don't handle that use case cleanly.

The best solution is the following:

1. A new class will be written, named NativeBuffer. This class will be backed by native methods to allocate a contiguous buffer of bytes using the C runtime heap. Native methods will be required to allocate (e.g. malloc), de-allocate (e.g. free), read and write to this buffer. These native methods will be a set of primitives that are used by NativeBuffer convenience methods which will allow reading/writing a wider range of types.

2. Creating an instance of the new class will naturally allocate a buffer of a given size. The memory address will be stored in a field in NativeBuffer. Read/write calls down to the JNI layer will take that address and some parameters to specify the data sink/source to be read/written and the offset and length of the portion to read/write. Use of the instance should always be protected by a call to free the buffer. Normally, this is done via a finally block, but since these are long lived, we will probably need some other approach, possibly scope based (e.g. some TransactionManager facility).

3. The direct use of byte[] will be removed from BinaryData. Instead, it will rely upon abstract or protected workers to read/write the data and each subclass will implement this minimum worker set using the data type that is a best fit. raw will directly use byte[] and memptr will use NativeBuffer.

4. The implementation of GET-BYTE-ORDER and SET-BYTE-ORDER will have its most important backing features implemented in NativeBuffer. The actual constants and so forth will be in memptr, but the behavior of honoring a particular type or read/write will be in NativeBuffer.

5. Some research will be needed to fully understand the SET-SIZE/GET-SIZE and GET-POINTER-VALUE/SET-POINTER-VALUE functionality. On the surface, the documentation seems reasonable, but it must be proven to be correct. The good news is that it is all supporting 64-bit addresses, so the most concerning part of the solution is straightforward.

#24 - 04/10/2013 03:03 PM - Greg Shah

The error behavior of all the BinaryData.get/set*() methods will need to be researched before implementation. If all goes well, the same error processing and helper methods can be used as are currently used for get/setString() and get/setByte(). But for the current methods, the behavior was found to be a bit tricky, so this may require more time to get right than initially expected.

#25 - 04/16/2013 09:08 AM - Greg Shah

- Status changed from New to WIP

- Assignee set to Greg Shah

Something to highlight about this implementation: it must be implemented in the client. The real memory backed MEMPTR implementation is primarily used to integrate with native APIs, passing real memory buffers to APIs via real 64-bit pointers. The native API implementation will have to be on the client (see #1634). As such, the memptr implementation will also need to be on the client so that the pointers are referencing memory in the correct address space/process.

The implication of this is that there will need to be a remote interface used to access the low level NativeBuffer methods and the client will have to export this, just like it exports process launching and file-system access.

#26 - 05/28/2013 09:18 AM - Greg Shah

- File ges_upd20130523a.zip added

- Status changed from WIP to Closed

This is a complete re-write of the memptr and raw data types. Since most of the implementation of these types is shared, the BinaryData class is where the majority of the implementation still resides. The new memptr design uses native (JNI) methods for the direct allocation, deallocation, reading and writing of memory. This means that this memory is outside of the Java heap and the 4GL can directly access pointers in memory and pass pointers to API calls. This is necessary for SET-POINTER-VALUE/GET-POINTER-VALUE and more importantly, for RUN of procedures defined using EXTERNAL. This is how 4GL code accesses native library functions (e.g. in a .DLL or .so). This set of requirements means that the byte[] implementation that used to be in BinaryData was not sufficient. The raw data type is still backed by a byte[] (on the Java heap), but the memptr type is now backed by direct memory access via JNI. The BinaryData parent class has been enhanced to add a much more extensive set of abstract methods to allow all of the real reading/writing to be delegated to the child classes, while the real logic (and error handling etc...) remains in BinaryData. This update also substantially completes the memptr and raw implementation. All features are now present except for the RAW statement and RAW built-in function. The error processing, boundary conditions and all strange behavior has been duplicated faithfully except for a couple of cases noted in the Javadoc. This has all been extensively tested (see testcases/uast/memptr/*.p).

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

This has passed regression testing. The results are saved in shared/clients/timco/majic_test_results/10353_620e9a1_20130528_ges.zip.

This is not yet applied to staging. It is committed to bzr as revision 10355.

The conversion changes do make one small change to Majic, which is expected and is OK.

#27 - 08/08/2014 11:51 AM - Constantin Asofiei

There is a bug in the BinaryData.assign code; if a raw/memptr value is set to unknown and then assigned to something else, the unknown state doesn't get cleared. To reproduce:

```
def var r as raw.
def var m as memptr.

set-size(m) = 11.
put-string(m, 1) = "0123456789".

r = ?.
r = m.

def var ch as char.
ch = get-string(r, 1).

message ch.
```

The r = m assignment doesn't clear the unknown flag.

Question: if the data is of length 0, does it need to be set to unknown? Or raw/memptr values with no data are considered not-unknown?

#28 - 08/08/2014 02:00 PM - Greg Shah

> Or raw/memptr values with no data are considered not-unknown?

raw/memptr has a concept of "uninitialized". For raw, this means that the value is null. This can occur in cases like assigning an unknown raw instance to another raw instance, but by default raw instances are not uninitialized. For memptr, the default state is uninitialized and it is uninitialized until backing memory is allocated (its size is set to something non-zero and usually a positive number). In other words, it is uninitialized so long as the addr is 0. After memory has been allocated, it can be reset to uninitialized by either setting the size of the instance to 0 or by assigning it data of length 0.

Neither type defaults to an unknown value. You can only get instances to be unknown by explicitly assigning from an unknown (literal or var), by passing an unknown value as a parameter to certain setters (byte-order or length) or by assigning in some code path that yields a null byte[].

> Question: if the data is of length 0, does it need to be set to unknown?

No.

For both raw and memptr, being uninitialized is the same has having data of length 0. Here is an example that shows it actually clears the unknown value:

```
def var m as memptr.
```

```
def var r as raw.
def var m2 as memptr.
def var r2 as raw.
def var m3 as memptr.
def var r3 as raw.

m = ?.
m2 = ?.
m = r3.
m2 = m3.

r = ?.
r2 = ?.
r = r3.
r2 = m3.

if m eq ? or r eq ? or m2 eq ? or r2 eq ? then message "Assigning from uninitialized vars should have cleared
unknown value.".
else message "OK".
```

I think we get this wrong (we don't call clearUnknown()). I would be tempted to do it in deallocate(), except that when you set the size of an unknown memptr instance to 0 (or the length of a raw to 0), it still reports as unknown:

```
def var m as memptr.
def var r as raw.

m = ?.
set-size(m) = 0.

r = ?.
length(r) = 0.

if m ne ? or r ne ? then message "Setting the length of raw or size of memptr to 0 does not clear unknown valu
e.".
else message "OK".
```

I think the better place to call clearUnknown() is in replaceContents(). This will also solve the original reported issue.

#29 - 08/11/2014 12:38 PM - Constantin Asofiei

- File ca_upd20140811a.zip added

I have some weird unknown equality test using memptr... See this test:

```
def var r1 as raw.
def var m1 as memptr.
def var r2 as raw.
def var m2 as memptr.

procedure proc0.
   def output param r as raw.
   def output param m as memptr.

   message string(r) string(m).

   if (r eq ?) ne (not(r ne ?))
      then message "a unknown problem raw:" (r eq ?) (r ne ?) (not(r ne ?)).
   if (r ne ?) ne (not(r eq ?))
      then message "b unknown problem raw:" (r ne ?) (r eq ?) (not(r eq ?)).
   if (m eq ?) ne (not(m ne ?))
      then message "c unknown problem memptr" (m eq ?) (m ne ?) (not(m ne ?)).
   if (m ne ?) ne (not(m eq ?))
      then message "d unknown problem memptr" (m ne ?) (m eq ?) (not(m eq ?)).
end.

message "case 1".
run proc0(output r1, output m1).

message "case 2".
r2 = ?.
m2 = ?.
run proc0(output r2, output m2).
```

the output is:

```
case 1
020000 ?
case 2
020000 ?
c unknown problem memptr yes yes no
d unknown problem memptr yes yes no
```

Even if in both cases the string representation of the memptr is the same, in second case, when memptr is set to unknown, the unknown equality test is computed different: it's like Schrodinger's cat, the memptr is both unknown and "not unknown"... This doesn't make sense.

Attached is an update which fixes the raw problem in MAJIC; their case was like this:

```
r = ?.
run something(output r).
```

#30 - 08/11/2014 01:17 PM - Greg Shah

Code Review 0811a

The changes look good. When it passes testing, please check it in and distribute it.

#31 - 08/11/2014 01:34 PM - Greg Shah

> when memptr is set to unknown, the unknown equality test is computed different: it's like Schrodinger's cat, the memptr is both unknown and "not unknown"... This doesn't make sense.

How Progress creates so many quirky/unexplainable implementations is hard to understand. I often wonder if they could create a greater number of buggy programs if they were intentionally trying to do so. I suspect not.

Interestingly, the problem seems to be specific to parameter processing:

```
def var mp1 as memptr.
def var mp2 as memptr.
def var mp3 as memptr.

procedure bogus:
   def input parameter m1 as memptr.
   def input-output parameter m2 as memptr.
   def output parameter m3 as memptr.

   message (m1 eq ?) (m1 ne ?) (m2 eq ?) (m2 ne ?) (m3 eq ?) (m3 ne ?).
end.

mp1 = ?.
mp2 = ?.
mp3 = ?.

message (mp1 eq ?) (mp1 ne ?) (mp2 eq ?) (mp2 ne ?) (mp3 eq ?) (mp3 ne ?).
run bogus (input mp1, input-output mp2, output mp3).
```

Output:

```
yes no yes no yes no
yes yes yes yes yes yes
```

Thoughts on how to accommodate this?

#32 - 08/11/2014 04:26 PM - Constantin Asofiei

Greg Shah wrote:

> Thoughts on how to accommodate this?

Looks like this special state is preserved until the memptr var gets assigned or its size changed (even to 0), and it can be leaked outside of the procedure where the memptr was used as output param. More, this is not limited to the ? literal:

```
def var mp1 as memptr.
def var mp2 as memptr.
def var mp3 as memptr.
def var mp as memptr.
mp = ?.

procedure bogus:
   def input parameter m1 as memptr.
   def input-output parameter m2 as memptr.
   def output parameter m3 as memptr.

   message (m1 eq mp) (m1 ne mp) (m2 eq mp) (m2 ne mp) (m3 eq mp) (m3 ne mp).
end.

mp1 = mp.
mp2 = mp.
mp3 = mp.

message (mp1 eq mp) (mp1 ne mp) (mp2 eq mp) (mp2 ne mp) (mp3 eq mp) (mp3 ne mp).
run bogus (input mp1, input-output mp2, output mp3).
message (mp1 eq mp) (mp1 ne mp) (mp2 eq mp) (mp2 ne mp) (mp3 eq mp) (mp3 ne mp).
```

output will be the same as in your case:

```
yes no yes no yes no
yes yes yes yes yes yes
yes no yes yes yes yes
```

For the comparison operator issues, I think we can get away without quirky conversion rules. We need to treat this special memptr state in comparison operators, and this can be done directly in the runtime code. All comparison operators should be checked.

Now, how to track this special state of memptr vars. First of all, instead of using the BinaryData.unknown boolean field, we need a memptr.unknown int field with 3 states:

- -1 for false/negative
- 1 for true/positive
- 0 for the "undetermined" state

Conversion rules are needed in the init block of any proc/function:

```
   public void bogus(final memptr _m1, final memptr m2, final memptr m3)
   {
      internalProcedure(new Block()
      {
         memptr m1 = memptr.inputParamteter(_m1);  /* instead of "memptr m1 = new memptr(_m1);" */

         public void init()
         {
            TransactionManager.deregister(new Undoable[]
            {
               m2,
               m3
            });
            m2.inputOuputParameter(m2);  /* this is new code */
            m3.outputParameter(m3); /* instead of m3.assign(new memptr()); */
         }
```

Depending on the existing state of the received arguments and the parameter type, the memptr parameters will initialize accordingly. This will either default to normal logic (if the argument is not unknown) or it will set the memptr in the undetermined state, in which it will be kept until it gets assigned to something else or its size gets changed. The CompareOps will need to check if an operand is a memptr, and if is in undetermined state, it needs to act accordingly.

#33 - 08/12/2014 05:03 AM - Constantin Asofiei

Greg Shah wrote:

> Code Review 0811a
>
> The changes look good. When it passes testing, please check it in and distribute it.

Committed to bzr rev 10597.

#34 - 08/12/2014 09:08 AM - Greg Shah

I'm fine with the proposed approach. My only concern is that I don't want us to spend too much time fixing this, as we have so much other work that needs to get done. If you can get this done in 4 hours or so, then go ahead now. Otherwise, create a task and we'll do the work later.

#35 - 11/16/2016 11:42 AM - Greg Shah

- Target version changed from Milestone 7 to Runtime Support for Server Features

Files

cs_upd20130128a.zip	70 KB	01/28/2013	Costin Savin
cs_upd20130128b.zip	91 KB	01/28/2013	Costin Savin
cs_upd20130129a.zip	112 KB	01/29/2013	Costin Savin
cs_upd20130129b.zip	112 KB	01/29/2013	Costin Savin
ges_upd20130523a.zip	545 KB	05/28/2013	Greg Shah
ca_upd20140811a.zip	31.5 KB	08/11/2014	Constantin Asofiei