Base Language - Bug #2183

fix resource save/restoration when used with string/handle functions

09/17/2013 12:22 PM - Constantin Asofiei

Status:	Closed	**Start date:**	09/18/2013
Priority:	Normal	**Due date:**	10/04/2013
Assignee:	Hynek Cihlar	**% Done:**	100%
Category:		**Estimated time:**	12.00 hours
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Feature #1781: add missing support for the WIDGET-...	**Closed**	**09/18/2013**	**10/04/2013**

History

#1 - 09/17/2013 12:24 PM - Constantin Asofiei

- Estimated time set to 12.00

Notes from #1639:
[GES]

> 3. Is it safe to use the hash code for the ID to resource mapping in handle? It seems like this could easily return back the wrong resource or have a resource overwritten because of conflicts. Is the hash code always unique for all resource types? Even if it is, someone could come along later and change the implementation without realizing that we have a dependency on it...

[CA]

> Last changes don't change how handle.hashCode used to work in P2J. resourceId (which relies on hashCode) saves the resource in the map, because at some point it might need to restore it. Also, the fromString and toString use the same approach (hash code key in the map).
> About collisions for the keys of the handle.WorkArea.map, yes, I too think your concerns are valid. More, I think there are a couple of other issues with it:
> - I don't think the map should hold handle instances, but actual resources (as a handle can change resource). A simple test shows that HANDLE function returns the resource, not the handle:

```
def var h as handle.
def var h2 as handle.

h = session.
ch = string(h).

h2 = handle(ch). /* this will not return h, but the resource which was saved via string(h) */
create socket h2.

message h h2. /* h and h2 have different IDs. if HANDLE returned h var, then h and h2 would have been the same
 */
```

> - to avoid hashCode as resourceId, we should add a counter for the ID, but this must track resource instances instead of handle instances

#2 - 09/18/2013 09:43 AM - Greg Shah

- Start date set to 09/18/2013

- Assignee set to Hynek Cihlar

- Due date set to 10/04/2013

Work this along with the work in #1781. They are closely related.

#3 - 10/15/2013 04:33 PM - Hynek Cihlar

#1781-15
Also, for security purposes, we should not make the resource ID algorithm a simple sequence. It should be randomized such that malicious code cannot anticipate the resource IDs that will be generated and thus cannot obtain access to a resource that they should not have.

To prevent the malicious code from guessing valid IDs they will have to be evenly distributed in a big enough range. I'm guessing that scanning the whole 64bit range (2^64 combinations) will be matter of days assuming a single validity test will take tens of CPU instructions. So if the security is real concern the IDs should be wider than 64 bits and generated by the Secure random number generator to be evenly distributed across the whole range. This leads me to Java UUID. The downside of using UUIDs of course is the higher memory and CPU impact. For example generating a random number with SecureRandom should be several tens times slower than Random.

#4 - 10/15/2013 04:37 PM - Greg Shah

The random 64-bit value is acceptable for now. The 4GL apps have so many other security holes that it isn't worth spending too much time on this right now.

#5 - 10/20/2013 05:04 PM - Hynek Cihlar

- Status changed from New to WIP

#6 - 10/31/2013 09:54 PM - Hynek Cihlar

- File hc_upd20131031a.zip added

- Status changed from WIP to Review

Attached are the changes to review. The changes passed regression tests.

The source code includes several TODO comments, describing issues to be resolved:

(1) "Zero" handles
Should the behavior of "zero" handles be replicated in the converted code? Zero handle is constructed
in the native Progress environment by calling the handle method supplying a formally valid
but made-up (not received by the string(handle) function) resource id value.
The handle constructed this way "holds" no resource and is not unknown but zero, example:
h = handle("123456789").
str = string(h).
now str equals "0"
I didn't find this behavior documented in the Progress reference manual.
Should we replicate this kind of behavior in the translated code?

(2) Resource ID deallocation
When handle.resourceId is called, new ID is allocated and stored in the context. To cleanup when the related resource is deleted, handle.removeResource must be called. Currently this happens from the resource's delete method. However it is a potential place for memory leak, one must not forget to deallocate the IDs explicitly (if not already called in the resource's delete method). Is a more transparent resource ID management desired?

(3) Resource ID together with its resource?
Should the resource id be stored in the resource itself (for example WrappedResource extended with getResourceId method)? Probably not, because resource ids are closely related to handles (one can get the resource id only by calling string(handle)).

(4) The default handle string format had to be extended to cover the Long IDs. I found one test case failing (testcases/uast) due to longer ID string.

Any other foreseeable issues with the new string format?

#7 - 11/01/2013 07:06 AM - Greg Shah

Overall, this is very good work.

Answers

(1) "Zero" handles

As a general rule, we always implement any language feature (documented or not) that can be relied upon or otherwise seen in 4GL application behavior. In fact, I would say that much if not most of the functionality we have implemented is undocumented.

Yes, please do implement this exactly as it works in the 4GL.

(2) Resource ID deallocation

Yes, we need to solve this for sure. Constantin is working on something related right now.

Constantin: what do you think?

(3) Resource ID together with its resource?

I suspect the resource ID is logically associated with the handle, not the resource. I wonder if you could have 2 handles with different IDs referring to the same resource?

(4) The default handle string format had to be extended to cover the Long IDs. I found one test case failing (testcases/uast) due to longer ID string. Any other foreseeable issues with the new string format?

Actually, I would like you to answer this question. In particular, I would like for you to explore and document exactly where the incompatibilities exist between our approach and the 4GL. Then we can assess the impact.

Code Review 1031a

1. You will need to merge with the latest rev in bzr.

2. In handle.removeResource():

```
      WorkArea wa = work.get();
      Long id = wa.resourceIds.get(resource);
      wa.idHandles.remove(id);
      wa.resourceIds.remove(resource);
```

I think it is more appropriately coded like this:

```
      if (resource == null)
         return;

      WorkArea wa = work.get();
      Long id = wa.resourceIds.remove(resource);

      if (id != null)
         wa.idHandles.remove(id);
```

The reason for this is that we don't know that the idHandles or resourceIds maps are null-safe.

3. To answer this (from handle.resourceId():

TODO: is it enough to state that removeResource is to be called? or should we have more transparent deallocation of the entries in the resourceIds and idHandles maps?

We must have completely transparent/automatic cleanup of the related maps.

4. handle.fromResourceId() seems to be able to return null. That is used to directly return values to converted code. I don't see how that is compatible with the 4GL behavior which has no concept of an NPE. The result has to be safe and compatible. I suspect there are use cases that are missing here. The "zero handle" problem is part of this, but I worry that there are other cases too.

5. In handle.resourceId():

```
} while(wa.idHandles.containsKey(id));
```

should be this:

```
}
while (wa.idHandles.containsKey(id));
```

6. handle.fromResourceId() needs javadoc.

7. ResourceIdHelper.idToString() should return UNKNOWN_STRING instead of the literal "?".

8. ResourceIdHelper.idFromString() should respond in a 4GL compatible way when the input is out of bounds, unknown or the empty string. We can't leave this unimplemented, nor can we use Java exceptions like IllegalArgumentException since "valid" 4GL code can actually trigger these cases. What does Progress do in this case?

9. ResourceIdHelper copyright range should be 2013 only instead of a range.

10. RemoteResourceImpl, ProxyProcedureWrapper are missing header entries.

11. ServerEvent has an unnecessary indent in its history entries, please eliminate this (in your entry and the one before).

#8 - 11/01/2013 07:32 AM - Constantin Asofiei

(2) Resource ID deallocation

Yes, we need to solve this for sure. Constantin is working on something related right now.

Constantin: what do you think?

The resource ID should be deallocated when the resource actually gets deleted (considering that this is the resource's ID and not the handle's - see my notes bellow).

(3) Resource ID together with its resource?

I suspect the resource ID is logically associated with the handle, not the resource. I wonder if you could have 2 handles with different IDs referring to the same resource?

Is worth mentioning here the 4GL bug (I don't think they did this on purpose) related to static QUERY resources. A static QUERY can't be deleted, but a handle referring a static QUERY can be deleted (see note 60 in #2120). This is the only case I've found in which the static resource and its associated handle don't behave the same.

In the static QUERY case, after the handle is deleted, obtaining the QUERY handle one more time will produce a different ID. I think this shows that the ID is associated with the handle. The problem is in P2J the resource and handle ID is the same thing. Unless we can prove that 2 handles with different IDs can refer the same resource, P2J can remain with the IDs linked with the resource.

#9 - 11/02/2013 05:24 PM - Hynek Cihlar

(1) "Zero" handles

As a general rule, we always implement any language feature (documented or not) that can be relied upon or otherwise seen in 4GL application behavior. In fact, I would say that much if not most of the functionality we have implemented is undocumented.

Yes, please do implement this exactly as it works in the 4GL.

Ok, will do.

(3) Resource ID together with its resource?

I suspect the resource ID is logically associated with the handle, not the resource. I wonder if you could have 2 handles with different IDs referring to the same resource?

I was trying to answer the same question, looking in the Progress reference docs. No definite answer. But I would say you cannot have two handles with different IDs referring to the same resource and this is the way handle.java is coded at the moment.

(4) The default handle string format had to be extended to cover the Long IDs. I found one test case failing (testcases/uast) due to longer ID string. Any other foreseeable issues with the new string format?

Actually, I would like you to answer this question. In particular, I would like for you to explore and document exactly where the incompatibilities exist between our approach and the 4GL. Then we can assess the impact.

Will try.

Code Review 1031a

1. You will need to merge with the latest rev in bzr.

Yes, will do and will run the regression again on the merged code.

2. In handle.removeResource():

[...]

I think it is more appropriately coded like this:

[...]

The reason for this is that we don't know that the idHandles or resourceIds maps are null-safe.

Yes. I didn't realize, as you as well point out in other points below, that error handling is also subjected to the conversion and compatibility conformance. In regular Java I am used to Illegal*Exceptions sometimes even NullPointerExceptions to be part of error handling. Of course Progress conversion is another case, I will keep this in mind.

Will fix.

3. To answer this (from handle.resourceId():

TODO: is it enough to state that removeResource is to be called? or should we have more transparent deallocation of the entries in the resourceIds and idHandles maps?

> We must have completely transparent/automatic cleanup of the related maps.

Will ask Constantin for a consultation on this.

> 4. handle.fromResourceId() seems to be able to return null. That is used to directly return values to converted code. I don't see how that is compatible with the 4GL behavior which has no concept of an NPE. The result has to be safe and compatible. I suspect there are use cases that are missing here. The "zero handle" problem is part of this, but I worry that there are other cases too.

Will fix.

5. - 11.

Will fix.

#10 - 11/15/2013 08:37 AM - Greg Shah

Please document the current status and upload the latest version for code review.

#11 - 11/15/2013 11:22 AM - Hynek Cihlar

- File hc_upd20131115a.zip added

See the attached hc_upd20131115a.zip with the latest code version. It contains changes and fixes from the previous code review resolving all the mentioned points.

The most interesting update is the resource deallocation. The idea is to keep the resource management simple and to rely on the Java garbage collector by not referencing Progress resources directly but through Java weak references (see the class WeakReference). In the code you will see two maps storing id-to-resource and resource-to-id mappings. The maps store WrappedResource instances as weak references. This way handle class will not block garbage collector from collecting Progress resources with unreferenced outstanding handles. Please note that this part of code is not yet documented and not regression-tested.

#12 - 11/15/2013 11:31 AM - Constantin Asofiei

Are you sure that WeakReference is the way to go? What about this scenario:

1. user creates a resource and saves it in a handle var

2. the handle's ID is saved in a char var
3. that var is initialized to something else
4. after a long time, the user wants to retrieve that handle via the handle function

The code would look like this:

```
def var h as handle.
def var ch as handle.

create sax-reader h. /* this is a resource which can be retreived only by its ID; no next/prev/first/last refe
rences are kept. */
ch = string(h).

pause 3600.

h = string(ch).
if not valid-handle(h) then message "The handle must be valid!".
```

Idea is, there might be cases where the resource must not be implicitly deleted by the GC. I want to rely only on explicit (or implicit) resource deletion. If the a resource is long-lived, then let it live, even if no one else is referring it.

#13 - 11/15/2013 12:26 PM - Greg Shah

Code Review 1115a

1. The explicit classes imported in handle.java should be imported with the wildcard.

2. handle.purgeStaleReferences() and the inner classes at the end of the class need javadoc (if they survive).

More broadly, I agree with Constantin's assessment. We can't rely upon weak references, the references need to be explicitly managed.

#14 - 11/15/2013 12:47 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Are you sure that WeakReference is the way to go? What about this scenario:
>
> 1. user creates a resource and saves it in a handle var
> 2. the handle's ID is saved in a char var
> 3. that var is initialized to something else
> 4. after a long time, the user wants to retrieve that handle via the handle function
>
> The code would look like this:
> [...]
> Idea is, there might be cases where the resource must not be implicitly deleted by the GC. I want to rely only on explicit (or implicit) resource

> deletion. If the a resource is long-lived, then let it live, even if no one else is referring it.

What is the boundary for the resource lifetime in this case? In other words, when should the fromString() return an invalid handle? Is it after the function exits, session ends, etc.? Is it the same for all resource types?

Are there any Progress resources subject to garbage collection? For example class instances?

#15 - 11/15/2013 12:48 PM - Hynek Cihlar

Greg Shah wrote:

> Code Review 1115a
>
> 1. The explicit classes imported in handle.java should be imported with the wildcard.

Fixed.

> 2. handle.purgeStaleReferences() and the inner classes at the end of the class need javadoc (if they survive).
>
> More broadly, I agree with Constantin's assessment. We can't rely upon weak references, the references need to be explicitly managed.

To be resolved.

#16 - 11/15/2013 01:09 PM - Constantin Asofiei

> What is the boundary for the resource lifetime in this case? In other words, when should the fromString() return an invalid handle? Is it after the function exits, session ends, etc.? Is it the same for all resource types?

All resources are alive at most as long as the session is alive, and are private to that session. You don't need to worry about who deletes a resource, for this task you need to make sure that if a resource gets deleted, it is removed from the user's registry of resources.

> Are there any Progress resources subject to garbage collection? For example class instances?

The class instances are not resources, thus they are not handled by handles.

#17 - 11/15/2013 01:37 PM - Hynek Cihlar

> What is the boundary for the resource lifetime in this case? In other words, when should the fromString() return an invalid handle? Is it after the function exits, session ends, etc.? Is it the same for all resource types?

> All resources are alive at most as long as the session is alive, and are private to that session. You don't need to worry about who deletes a resource, for this task you need to make sure that if a resource gets deleted, it is removed from the user's registry of resources.

When a resource is deleted (by all means) is its instance method Deletable.delete always called? Do we care if the user doesn't explicitly delete a resource?

> Are there any Progress resources subject to garbage collection? For example class instances?

> The class instances are not resources, thus they are not handled by handles.

Thanks for the clarifications!

#18 - 11/15/2013 01:45 PM - Constantin Asofiei

Hynek Cihlar wrote:

> When a resource is deleted (by all means) is its instance method Deletable.delete always called?

Yes, that will always be called. The callpaths you need to check are:

- HandleChain.delete - this is the entry point for deleting almost all resources

- ExternalProgramWrapper.delete
- ProxyProcedureWrapper.delete

All these cases already have a handle.removeResource call.

> Do we care if the user doesn't explicitly delete a resource?

No, if this is a non-static resource, then is user's responsibility (this is what widget pools are for, to allow mass deletion of non-static resources). For static resource cases, is P2J's responsibility (see #2196).

#19 - 11/15/2013 01:53 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > When a resource is deleted (by all means) is its instance method Deletable.delete always called?
>
> Yes, that will always be called. The callpaths you need to check are:
>
> - HandleChain.delete - this is the entry point for deleting almost all resources
> - ExternalProgramWrapper.delete
> - ProxyProcedureWrapper.delete
>
> All these cases already have a handle.removeResource call.
>
> > Do we care if the user doesn't explicitly delete a resource?
>
> No, if this is a non-static resource, then is user's responsibility (this is what widget pools are for, to allow mass deletion of non-static resources). For static resource cases, is P2J's responsibility (see #2196).

In that case the previously submitted sources (hc_upd20131031a.zip) already had all what is necessary for the resource deallocation. I will undo the weak references, regression test and submit for final review.

#20 - 11/17/2013 04:28 PM - Hynek Cihlar

- File hc_upd20131117a.zip added

Attached is the release candidate.

- I reverted the weak references introduced in hc_upd20131115a.zip, deallocation of resources should be now according the requirements.
- Also improved ResourceIdHelper.idFromString - parsing of integer string is now the responsibility of NumberType class. Finally, the parse semantics is in line with Progress semantics.
- Plus other smaller fixes and documentation improvements.

Regression test is still in progress.

#21 - 11/18/2013 02:30 AM - Constantin Asofiei

Hynek, the RemoteResource/RemoteResourceImpl/Agent changes are incorrect. The RemoteResource interface is used to send back to the other side the ID of a resource. When a appserver sends to the requester a handle instance, the requester receives a RemoteResource instance holding the resource ID as on the remote side; when the appserver receives from the requester a RemoteResource, it will use it to resolve the correct legacy resource. Thus, In Agent.preProcessArguments:1163, you can't use handle.resourceId, as the resource ID is already known - must be the ID specified by the RemoteResource.

The other changes look OK to me.

#22 - 11/18/2013 09:30 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, the RemoteResource/RemoteResourceImpl/Agent changes are incorrect. The RemoteResource interface is used to send back to the other side the ID of a resource. When a appserver sends to the requester a handle instance, the requester receives a RemoteResource instance holding the resource ID as on the remote side; when the appserver receives from the requester a RemoteResource, it will use it to resolve the correct legacy resource. Thus, In Agent.preProcessArguments:1163, you can't use handle.resourceId, as the resource ID is already known - must be the ID specified by the RemoteResource.
>
> The other changes look OK to me.

Thank you Constantin for the catch. This is a piece of code related to a version with resource ids being saved directly in the resources, I forgot to remove it.

I've run regression test and will post the fixed changes when the test finishes.

#23 - 11/19/2013 04:19 PM - Hynek Cihlar

- File hc_upd20131118a.zip added

Yesterday's regression test run ended with suspiciously high number of errors. I've started another control run. Meanwhile I am attaching the latest changes with the fixed RemoteResource/RemoteResourceImpl/Agent, please review.

#24 - 11/20/2013 09:36 AM - Constantin Asofiei

Hynek Cihlar wrote:

> ... I am attaching the latest changes with the fixed RemoteResource/RemoteResourceImpl/Agent, please review.

The update is OK, as you've backed out the Agent.java, ProxyProcedureWrapper.java, RemoteResourceImpl.java and RemoteResource.java changes.

#25 - 11/26/2013 05:32 AM - Hynek Cihlar

The code (hc_upd20131118a.zip) passed regression tests (the run 20131116) and was committed as revision 10416.

#26 - 11/26/2013 08:41 AM - Greg Shah

- Status changed from Review to Closed

- % Done changed from 0 to 100

#27 - 12/18/2013 04:50 AM - Constantin Asofiei

Some socket-related static proxy methods got bad after the 1118a.zip update, as some signatures are out-of-sync. I'm working on fixing them.

Greg: after I've added logging to the static proxy registration, I found these too:

> [12/18/2013 11:49:54 EET] (RemoteObject:SEVERE) Unable find matching static method public abstract boolean com.goldencode.p2j.util.RemoteErrorData.isError() in backing class com.goldencode.p2j.util.ErrorManager.
> [12/18/2013 11:49:54 EET] (RemoteObject:SEVERE) Unable find matching static method public abstract void com.goldencode.p2j.util.RemoteErrorData.setDontClean(boolean) in backing class com.goldencode.p2j.util.ErrorManager.
> [12/18/2013 11:49:54 EET] (RemoteObject:SEVERE) Unable find matching static method public abstract boolean com.goldencode.p2j.admin.AdminExports.changeUserGroups(java.lang.String,java.lang.String[],java.lang.String[]) in backing class com.goldencode.p2j.admin.AdminServerImpl.

These don't affect sockets, but they could pose problems in the feature.

#28 - 12/18/2013 06:22 AM - Constantin Asofiei

I should have caught this earlier. Why are negative resource ids ignored in ResourceIdHelper.idFromString?

```
      // we only consider positive integers valid resource ids
      int signum = num.signum();
      if (signum <= 0)
      {
         return ZERO;
      }
```

This is not consistent with the handle.resourceId implementation, as it is possible for this method to generate a negative resource ID.

#29 - 12/18/2013 08:20 AM - Constantin Asofiei

- File ca_upd20131218a.zip added

Fixed signatures of static proxy methods. Added logging to RemoteObject.registerStaticNetworkServer, to report any unresolved proxied methods (the problems at note 27 are still reported).
LE: fixed a socket problem too.

Hynek: about the negative resource IDs, please let me know your opinion on the following:

- I think ResourceIdHelper.getNext should never return negative values
- with this in mind, ResourceIdHelper.idFromString and ResourceIdHelper.idToString should throw an exception if a negative ID is encountered. This looks OK, because if we change getNext to return always positive values, then these methods should never received a negative ID (as the only way to compute a resource ID is via getNext).

#30 - 12/18/2013 03:20 PM - Greg Shah

Code Review 1218a

The changes all look good.

The LT will need merging with the one just checked in by VIG.

In regard to the errors reported in note 27, go ahead and fix those issues and include the change in your next update (it can be for another task if 1218a has already passed testing).

#31 - 12/18/2013 05:04 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Fixed signatures of static proxy methods. Added logging to RemoteObject.registerStaticNetworkServer, to report any unresolved proxied methods (the problems at note 27 are still reported).
> LE: fixed a socket problem too.
>
> Hynek: about the negative resource IDs, please let me know your opinion on the following:
>
> - I think ResourceIdHelper.getNext should never return negative values
> - with this in mind, ResourceIdHelper.idFromString and ResourceIdHelper.idToString should throw an exception if a negative ID is encountered. This looks OK, because if we change getNext to return always positive values, then these methods should never received a

> negative ID (as the only way to compute a resource ID is via getNext).

The idea behind the ids was to utilize the whole 64 bits in the Long. And because Progress always formats the ids as positive numbers so does the ResourceIdHelper.idToString. The cycle Long Id -> String Id -> Long Id works as long as you use the methods from ResourceIdHelper to format the id to a string and to parse it back to a long.

Example (using byte instead of long to preserve space):

```
byte newId = ResourceIdHelper.getNext()
// newId == -1
String formattedNewId = ResourceIdHelper.idToString(newId)
// formattedNewId == "255"
byte parsedNewId = ResourceIdHelper.idFromString(formattedNewId)
// parsedNewId == -1
```

This all was done to use a primitive type to represent the id (to make it less memory intensive and preserve some GC cycles) while utilizing the whole bit-range.

The biggest disadvantage of this scheme is the added complexity and a potential room for errors.

Constantin, your proposed solution will work ok, we just lose half of the id set.

#32 - 12/18/2013 05:16 PM - Hynek Cihlar

Constantin Asofiei wrote:

> I should have caught this earlier. Why are negative resource ids ignored in ResourceIdHelper.idFromString?

The reason for this is that Progress returns "zero" handle when negative number is passed to the HANDLE function. But I agree the implementation is confusing. This should probably be implemented in the handle class, there it would be more obvious.

#33 - 12/19/2013 02:40 AM - Constantin Asofiei

- File ca_upd20131219b.zip added

- File ca_upd20131219a.zip added

This is 1218a.zip merged with latest rev. 1218a.zip passed, 1219a.zip was committed to bzr revision 10423.

1219b.zip fixes the static proxy problems at note 27 and the negative resource ID problems.

> The reason for this is that Progress returns "zero" handle when negative number is passed to the HANDLE function.

Then, I think is reasonable for our resource ID generation to never return a negative number. Internally, P2J may save the resource ID as long (as returned by handle.resourceId), and a negative resource ID will not be allowed to be properly restored using handle.fromString.

Hynek: please review the ResourceIdHelper changes in the attached 1219b.zip update.

#34 - 12/19/2013 08:00 AM - Greg Shah

Code Review 1219b

I am OK with the changes. If Hynek is OK with them, then go ahead with testing.

#35 - 12/19/2013 08:54 AM - Hynek Cihlar

Constantin, I would remove the doc references about treating the long as unsigned number, now we use it as plain signed long, together with removing the ResourceIdHelper.TWO_64 since it is useless now.

Also when you call handle.fromString() with a negative number, an exception is thrown instead of "zero" handle being returned.

Otherwise it looks ok.

#36 - 12/19/2013 09:01 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin, I would remove the doc references about treating the long as unsigned number, now we use it as plain signed long, together with removing the ResourceIdHelper.TWO_64 since it is useless now.

OK, will do.

> Also when you call handle.fromString() with a negative number, an exception is thrown instead of "zero" handle being returned.

I'm not sure throwing an exception in this case is correct. As you noted, when a negative value is passed to a HANDLE function call, 4GL will generate the ZERO handle.

#37 - 12/19/2013 09:54 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Also when you call handle.fromString() with a negative number, an exception is thrown instead of "zero" handle being returned.

I'm not sure throwing an exception in this case is correct. As you noted, when a negative value is passed to a HANDLE function call, 4GL will generate the ZERO handle.

Sorry, I didn't express myself clearly. With your change, when "-1" is passed to handle.fromString(), IllegalArgumentException is thrown. The expected behavior is however that a "zero" handle instance is returned.

#38 - 12/19/2013 10:02 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Sorry, I didn't express myself clearly. With your change, when "-1" is passed to handle.fromString(), IllegalArgumentException is thrown. The expected behavior is however that a "zero" handle instance is returned.

Hmm... handle.fromString calls ResourceIdHelper.idFromString, and the version from 1219b.zip doesn't throw an exception. Only ResourceIdHelper.idToString throws one, which is called from handle.toString() (the equivalent of the STRING function).

#39 - 12/19/2013 11:59 AM - Hynek Cihlar

True, I was looking at the other method. Please ignore the comment about throwing an exception when calling handle.fromString("-1").

#40 - 12/19/2013 12:27 PM - Constantin Asofiei

OK, one more question about the TWO_64 usage. In ResourceIdHelper.idFromString, you have this code (btw, shouldn't have been modulo(TWO_64) instead of substract?):

```
      // out of range of positive long?
      if (num.compareTo(MAX_LONG) > 0)
      {
         // make the value fit into the 2s complement long
         num.subtract(TWO_64);
      }
```

I think the correct approach is to throw an exception if we receive a value outside of our known range (which is [1..java.lang.Long.MAX_VALUE]). If we allow it, converting it to java long may result in referencing an existing resource (not necessarily the correct one).

#41 - 12/19/2013 02:44 PM - Hynek Cihlar

AFAIK, the subtract is correct for the 2s complement, but the code is missing check for value greater than max unsigned long, i.e. the range check you proposed.

Also

- Progress reports the error "** Value too large for integer. (78)" when the string value doesn't fit into the range of an unsigned 64bit number.
- And it reports an additional error ("** Decimal number is too large. (536)\n** Value too large for integer. (78)") when the string exceeds 50 significant digits (i.e. not including numsign, white spaces, leading zeros, etc.).
- Plus when the length of the input string exceeds 255 characters, "ZERO" handle is returned.

I think, the proper errors should be returned instead of throwing exception.

#42 - 12/20/2013 06:27 AM - Constantin Asofiei

- File ca_upd20131220a.zip added

This version raises the "** Value too large for integer. (78)" if the given value can't be stored in a 64 bit integer. The other two cases are added as TODOs.

Greg/Hynek; If the changes are OK with you, I'll put this in testing.

#43 - 12/20/2013 09:21 AM - Greg Shah

Code Review 1220a

I am fine with the changes. I think that these errors are really things that are generically raised in the decimal/integer code in the 4GL. I am actually hitting some of the same error requirements when processing the output/return data from the native API calls. We should probably find the proper common location for the error processing.

But for this task, go with your current implementation.

#44 - 12/21/2013 03:31 AM - Constantin Asofiei

1220a.zip passed testing, was committed to bzr rev 10424.

#45 - 11/16/2016 12:07 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

Files

hc_upd20131031a.zip	265 KB	11/01/2013	Hynek Cihlar
hc_upd20131115a.zip	265 KB	11/15/2013	Hynek Cihlar
hc_upd20131117a.zip	264 KB	11/17/2013	Hynek Cihlar
hc_upd20131118a.zip	251 KB	11/19/2013	Hynek Cihlar
ca_upd20131218a.zip	70 KB	12/18/2013	Constantin Asofiei
ca_upd20131219a.zip	70.1 KB	12/19/2013	Constantin Asofiei
ca_upd20131219b.zip	28.7 KB	12/19/2013	Constantin Asofiei